عنوان: EF Code First #1
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۰:۰۷:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در ادامه بحث ASP.NET MVC می‌شود به ابزاری به نام MVC Scaffolding اشاره کرد. کار این ابزار که توسط یکی از اعضای تیم ASP.NET MVC به نام استیو اندرسون تهیه شده، تولید کدهای اولیه یک برنامه کامل ASP.NET MVC از روی مدل‌های شما می‌باشد. حجم بالایی از کدهای تکراری آغازین برنامه را می‌شود توسط این ابزار تولید و بعد سفارشی کرد. MVC Scaffolding حتی قابلیت تولید کد بر اساس الگوی Repository و یا نوشتن Unit tests مرتبط را نیز دارد. بدیهی است این ابزار جای یک برنامه نویس را نمی‌تواند پر کند اما کدهای آغازین یک سری کارهای متداول و تکراری را به خوبی می‌تواند پیاده سازی و ارائه دهد. زیر ساخت این ابزار، علاوه بر ASP.NET MVC، آشنایی با Entity framework code first است.
در طی سری ASP.NET MVC که در این سایت تا به اینجا مطالعه کردید من به شدت سعی کردم از ابزارگرایی پرهیز کنم. چون شخصی که نمی‌داند مسیریابی چیست، اطلاعات چگونه به یک کنترلر منتقل یا به یک View ارسال می‌شوند، قراردادهای پیش فرض فریم ورک چیست یا زیر ساخت امنیتی یا فیلترهای ASP.NET MVC کدامند، چطور می‌تواند از ابزار پیشرفته Code generator ایی استفاده کند، یا حتی در ادامه کدهای تولیدی آن‌را سفارشی سازی کند؟ بنابراین برای استفاده از این ابزار و درک کدهای تولیدی آن، نیاز به یک پیشنیاز دیگر هم وجود دارد: «Entity framework code first»
امسال دو کتاب خوب در این زمینه منتشر شده‌اند به نام‌های:
Programming Entity Framework: DbContext , ISBN: 978-1-449-31296-1
Programming Entity Framework: Code First , ISBN: 978-1-449-31294-7
که هر دو به صورت اختصاصی به مقوله EF Code first پرداخته‌اند.

در طی روزهای بعدی EF Code first را با هم مرور خواهیم کرد و البته این مرور مستقل است از نوع فناوری میزبان آن؛ می‌خواهد یک برنامه کنسول باشد یا WPF یا یک سرویس ویندوز NT و یا ... یک برنامه وب.
البته از دیدگاه مایکروسافت، M در MVC به معنای EF Code first است. به همین جهت MVC3 به صورت پیش فرض ارجاعی را به اسمبلی‌های آن دارد و یا حتی به روز رسانی که برای آن ارائه داده نیز در جهت تکمیل همین بحث است.

مروری سریع بر تاریخچه Entity framework code first

ویژوال استودیو 2010 و دات نت 4، به همراه EF 4.0 ارائه شدند. با این نگارش امکان استفاده از حالت‌های طراحی database first و model first مهیا است. پس از آن، به روز رسانی‌های EF خارج از نوبت و به صورت منظم، هر از چندگاهی ارائه می‌شوند و در زمان نگارش این مطلب، آخرین نگارش پایدار در دسترس آن 4.3.1 می‌باشد. از زمان EF 4.1 به بعد، نوع جدیدی از مدل سازی به نام Code first به این فریم ورک اضافه شد و در نگارش‌های بعدی آن، مباحث DB migration جهت ساده سازی تطابق اطلاعات مدل‌ها با بانک اطلاعاتی، اضافه گردیدند. در روش Code first، کار با طراحی کلاس‌ها که در اینجا مدل داده‌ها نامیده می‌شوند، شروع گردیده و سپس بر اساس این اطلاعات، تولید یک بانک اطلاعاتی جدید و یا استفاده از نمونه‌ای موجود میسر می‌گردد.
پیشتر در روش database first ابتدا یک بانک اطلاعاتی موجود، مهندسی معکوس می‌شد و از روی آن فایل XML ایی با پسوند EDMX تولید می‌گشت. سپس به کمک entity data model designer ویژوال استودیو، این فایل نمایش داده شده و یا امکان اعمال تغییرات بر روی آن میسر می‌شد. همچنین در روش دیگری به نام model first نیز کار از entity data model designer جهت طراحی موجودیت‌ها آغاز می‌گشت.
اما با روش Code first دیگر در ابتدای امر مدل فیزیکی و یک بانک اطلاعاتی وجود خارجی ندارد. در اینجا EF تعاریف کلاس‌های شما را بررسی کرده و بر اساس آن، اطلاعات نگاشت‌های خواص کلاس‌ها به جداول و فیلدهای بانک اطلاعاتی را تشکیل می‌دهد. البته عموما تعاریف ساده کلاس‌ها بر این منظور کافی نیستند. به همین جهت از یک سری متادیتا به نام ویژگی‌ها یا اصطلاحا data annotations مهیا در فضای نام System.ComponentModel.DataAnnotations برای افزودن اطلاعات لازم مانند نام فیلدها، جداول و یا تعاریف روابط ویژه نیز استفاده می‌گردد. به علاوه در روش Code first یک API جدید به نام Fluent API نیز جهت تعاریف این

ویژگی‌ها و روابط، با کدنویسی مستقیم نیز درنظر گرفته شده است. نهایتا از این اطلاعات جهت نگاشت کلاس‌ها به بانک اطلاعاتی و یا برای تولید ساختار یک بانک اطلاعاتی خالی جدید نیز می‌توان کمک گرفت.

مزایای EF Code first

- مطلوب برنامه نویس‌ها! : برنامه نویس‌هایی که مدتی تجربه کار با ابزارهای طراح را داشته باشند به خوبی می‌دانند این نوع ابزارها عموما demo-ware هستند. چندجا کلیک می‌کنید، دوبار Next، سه بار OK و ... به نظر می‌رسد کار تمام شده. اما واقعیت این است که عمری را باید صرف نگهداری و یا پیاده سازی جزئیاتی کرد که انجام آن‌ها با کدنویسی مستقیم بسیار سریعتر، ساده‌تر و با کنترل بیشتری قابل انجام است.
- سرعت: برای کار با EF Code first نیازی نیست در ابتدای کار بانک اطلاعاتی خاصی وجود داشته باشد. کلاس‌های خود را طراحی و شروع به کدنویسی کنید.
- سادگی: در اینجا دیگر از فایل‌های EDMX خبری نیست و نیازی نیست مرتبا آن‌ها را به روز کرده یا نگهداری کرد. تمام کارها را با کدنویسی و کنترل بیشتری می‌توان انجام داد. به علاوه کنترل کاملی بر روی کد نهایی تهیه شده نیز وجود دارد و توسط ابزارهای تولید کد، ایجاد نمی‌شوند.
- طراحی بهتر بانک اطلاعاتی نهایی: اگر طرح دقیقی از مدل‌های برنامه داشته باشیم، می‌توان آن‌ها را به المان‌های کوچک و مشخصی، تقسیم و refactor کرد. همین مساله در نهایت مباحث database normalization را به نحوی مطلوب و با سرعت بیشتری میسر می‌کند.
- امکان استفاده مجدد از طراحی کلاس‌های انجام شده در سایر ORMهای دیگر. چون طراحی مدل‌های برنامه به بانک اطلاعاتی خاصی گره نمی‌خورند و همچنین الزاما هم قرار نیست جزئیات کاری EF در آن‌ها لحاظ شود، این کلاس‌ها در صورت نیاز در سایر پروژه‌ها نیز به سادگی قابل استفاده هستند.
- ردیابی ساده‌تر تغییرات: روش اصولی کار با پروژه‌های نرم افزاری همواره شامل استفاده از یک ابزار سورس کنترل مانند SVN، Git، مرکوریال و امثال آن است. به این ترتیب ردیابی تغییرات انجام شده به سادگی توسط این ابزارها میسر می‌شوند.
- ساده‌تر شدن طراحی‌های پیچیده‌تر: برای مثال پیاده سازی ارث بری، ایجاد کلاس‌های خود ارجاع دهنده و امثال آن با کدنویسی ساده‌تر است.

دریافت آخرین نگارش EF

برای دریافت و نصب آخرین نگارش EF نیاز است از NuGet استفاده شود و این مزایا را به همراه دارد:
به کمک NuGet امکان با خبر شدن از به روز رسانی جدید صورت گرفته به صورت خودکار درنظر گرفته شده است و همچنین کار دریافت بسته‌های مرتبط و به روز رسانی ارجاعات نیز در این حالت خودکار است. به علاوه توسط NuGet امکان دسترسی به کتابخانه‌هایی که مثلا در گوگل‌کد قرار دارند و به صورت معمول امکان دریافت آن‌ها برای ما میسر نیست، نیز بدون مشکل فراهم است (برای نمونه ELMAH، که اصل آن از گوگل‌کد قابل دریافت است؛ اما بسته نیوگت آن نیز در دسترس می‌باشد).
پس از نصب NuGet، تنها کافی است بر روی گره References در Solution explorer ویژوال استودیو، کلیک راست کرده و به کمک NuGet آخرین نگارش EF را نصب کرد. در گالری آنلاین آن، عموما EF اولین گزینه است (به علت تعداد بالای دریافت آن).
حین استفاده از NuGet جهت نصب Ef، ابتدا ارجاعاتی به اسمبلی‌های زیر به برنامه اضافه خواهند شد:

System.ComponentModel.DataAnnotations.dll

System.Data.Entity.dll

EntityFramework.dll

بدیهی است بدون استفاده از NuGet، تمام این کارها را باید دستی انجام داد.
سپس در پوشه‌ای به نام packages، فایل‌های مرتبط با EF قرار خواهند گرفت که شامل اسمبلی آن به همراه ابزارهای DB Migration است. همچنین فایل packages.config که شامل تعاریف اسمبلی‌های نصب شده است به پروژه اضافه می‌شود. NuGet به کمک این فایل و شماره نگارش درج شده در آن، شما را از به روز رسانی‌های بعدی مطلع خواهد ساخت:

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="EntityFramework" version="4.3.1" />
</packages>
```

همچنین اگر به فایل app.config یا web.config برنامه نیز مراجعه کنید، یک سری تنظیمات ابتدایی اتصال به بانک اطلاعاتی در آن ذکر شده است:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=4.3.1.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
EntityFramework">
      <parameters>
        <parameter value="Data Source=(localdb)\v11.0; Integrated Security=True;
MultipleActiveResultSets=True" />
      </parameters>
    </defaultConnectionFactory>
  </entityFramework>
</configuration>
```

همانطور که ملاحظه می‌کنید بانک اطلاعاتی پیش فرضی که در اینجا ذکر شده است، LocalDB می‌باشد. این بانک اطلاعاتی را از این آدرس نیز می‌توانید دریافت کنید.

البته ضرورتی هم به استفاده از آن نیست و از سایر نگارش‌های SQL Server نیز می‌توان استفاده کرد ولی خوب ... مزیت استفاده از آن برای کاربر نهایی این است که «نیازی به یک مهندس برای نصب، راه اندازی و نگهداری ندارد». تنها مشکل آن این است که از ویندوز XP پشتیبانی نمی‌کند. البته SQL Server CE 4.0 این محدودیت را ندارد.
ضمن اینکه باید درنظر داشت EF به فناوری میزبان خاصی گره نخورده است و مثال‌هایی که در اینجا بررسی می‌شوند صرفا تعدادی برنامه کنسول معمولی هستند و نکات عنوان شده در آن‌ها در تمام فناوری‌های میزبان موجود به یک نحو کاربرد دارند.

قراردادهای پیش فرض EF Code first

عنوان شد که اطلاعات کلاس‌های ساده تشکیل دهنده مدل‌های برنامه، برای تعریف جداول و فیلدهای یک بانک اطلاعات و همچنین مشخص سازی روابط بین آن‌ها کافی نیستند و مرسوم است برای پر کردن این خلاء از یک سری متادیتا و یا Fluent API مهیا نیز استفاده گردد. اما در EF Code first یک سری قرار داد توکار نیز وجود دارند که مطلع بودن از آن‌ها سبب خواهد شد تا حجم کدنویسی و تنظیمات جانبی این فریم ورک به حداقل برسند. برای نمونه مدل‌های معروف بلاگ و مطالب آن‌را درنظر بگیرید:

```
namespace EF_Sample01.Models
{
    public class Post
    {
        public int Id { set; get; }
        public string Title { set; get; }
        public string Content { set; get; }
        public virtual Blog Blog { set; get; }
    }
}
```

```
using System.Collections.Generic;

namespace EF_Sample01.Models
{
```

```
    public class Blog
    {
        public int Id { set; get; }
        public string Title { set; get; }
        public string AuthorName { set; get; }
        public IList<Post> Posts { set; get; }
    }
}
```

یکی از قراردادهای EF Code first این است که کلاس‌های مدل شما را جهت یافتن خاصیتی به نام Id یا ClassId مانند BlogId، جستجو می‌کند و از آن به عنوان primary key و فیلد identity جدول بانک اطلاعاتی استفاده خواهد کرد.
همچنین در کلاس Blog، خاصیت لیستی از Posts و در کلاس Post خاصیت virtual ایی به نام Blog وجود دارند. به این ترتیب روابط بین دو کلاس و ایجاد کلید خارجی متناظر با آن‌را به صورت خودکار انجام خواهد داد.
نهایتا از این اطلاعات جهت تشکیل database schema یا ساختار بانک اطلاعاتی استفاده می‌گردد.
اگر به فضاهای نام دو کلاس فوق دقت کرده باشید، به کلمه Models ختم شده‌اند. به این معنا که در پوشه‌ای به همین نام در پروژه جاری قرار دارند. یا مرسوم است کلاس‌های مدل را در یک پروژه class library مجزا به نام DomainClasses نیز قرار دهند. این پروژه نیازی به ارجاعات اسمبلی‌های EF ندارد و تنها به اسمبلی System.ComponentModel.DataAnnotations.dll نیاز خواهد داشت.

EF Code first چگونه کلاس‌های مورد نظر را انتخاب می‌کند؟

ممکن است ده‌ها و صدها کلاس در یک پروژه وجود داشته باشند. EF Code first چگونه از بین این کلاس‌ها تشخیص خواهد داد که باید از کدامیک استفاده کند؟ اینجا است که مفهوم جدیدی به نام DbContext معرفی شده است. برای تعریف آن یک کلاس دیگر را به پروژه برای مثال به نام Context اضافه کنید. همچنین مرسوم است که این کلاس را در پروژه class library دیگری به نام DataLayer اضافه می‌کنند. این پروژه نیاز به ارجاعی به اسمبلی‌های EF خواهد داشت. در ادامه کلاس جدید اضافه شده باید از کلاس DbContext مشتق شود:

```
using System.Data.Entity;
using EF_Sample01.Models;

namespace EF_Sample01
{
    public class Context : DbContext
    {
        public DbSet<Blog> Blogs { set; get; }
        public DbSet<Post> Posts { set; get; }
    }
}
```

سپس در اینجا به کمک نوع جنریکی به نام DbSet، کلاس‌های دومین برنامه را معرفی می‌کنیم. به این ترتیب، EF Code first ابتدا به دنبال کلاسی مشتق شده از DbContext خواهد گشت. پس از یافتن آن، خواصی از نوع DbSet را بررسی کرده و نوع‌های متناظر با آن‌را به عنوان کلاس‌های دومین درنظر می‌گیرد و از سایر کلاس‌های برنامه صرفنظر خواهد کرد. این کل کاری است که باید انجام شود.
اگر دقت کرده باشید، نام کلاس‌های موجودیت‌ها، مفرد هستند و نام خواص تعریف شده به کمک DbSet، جمع می‌باشند که نهایتا متناظر خواهند بود با نام جداول بانک اطلاعاتی تشکیل شده.

تشکیل خودکار بانک اطلاعاتی و افزودن اطلاعات به جداول

تا اینجا بدون تهیه یک بانک اطلاعاتی نیز می‌توان از کلاس Context تهیه شده استفاده کرد و کار کدنویسی را آغاز نمود. بدیهی

است جهت اجرای نهایی کدها، نیاز به یک بانک اطلاعاتی خواهد بود. اگر تنظیمات پیش فرض فایل کانفیگ برنامه را تغییر ندهیم، از همان defaultConnectionFactory یاده شده استفاده خواهد کرد. در این حالت نام بانک اطلاعاتی به صورت خودکار تنظیم شده و مساوی «EF_Sample01.Context» خواهد بود.
برای سفارشی سازی آن نیاز است فایل app.config یا web.config برنامه را اندکی ویرایش نمود:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
...
  </configSections>
  <connectionStrings>
    <clear/>
    <add name="Context"
         connectionString="Data Source=(local);Initial Catalog=testdb2012;Integrated Security = true"
          providerName="System.Data.SqlClient"
           />
  </connectionStrings>
...
</configuration>
```

در اینجا به بانک اطلاعاتی testdb2012 در وهله پیش فرض SQL Server نصب شده، اشاره شده است. فقط باید دقت داشت که تگ configSections باید در ابتدای فایل قرار گیرد و مابقی تنظیمات پس از آن.
یا اگر علاقمند باشید که از SQL Server CE استفاده کنید، تنظیمات رشته اتصالی را به نحو زیر مقدار دهی نمائید:

```
<connectionStrings>
              <add name="MyContextName"
                         connectionString="Data Source=|DataDirectory|\Store.sdf"
                         providerName="System.Data.SqlServerCe.4.0" />
</connectionStrings>
```

در هر دو حالت، name باید به نام کلاس مشتق شده از DbContext اشاره کند که در مثال جاری همان Context است.
یا اگر علاقمند بودید که این قرارداد توکار را تغییر داده و نام رشته اتصالی را با کدنویسی تعیین کنید، می‌توان به نحو زیر عمل کرد:

```
public class Context : DbContext
{
    public Context()
      : base("ConnectionStringName")
    {
    }
```

البته ضرورتی ندارد این بانک اطلاعاتی از پیش موجود باشد. در اولین بار اجرای کدهای زیر، به صورت خودکار بانک اطلاعاتی و جداول Blogs و Posts و روابط بین آن‌ها تشکیل می‌گردد:

```
using EF_Sample01.Models;

namespace EF_Sample01
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            using (var db = new Context())
            {
                db.Blogs.Add(new Blog { AuthorName = "Vahid", Title = ".NET Tips" });
                db.SaveChanges();
            }
        }
    }
}
```

در این تصویر چند نکته حائز اهمیت هستند:

الف) نام پیش فرض بانک اطلاعاتی که به آن اشاره شد (اگر تنظیمات رشته اتصالی قید نگردد).

ب) تشکیل خودکار primary key از روی خواصی به نام Id

ج) تشکیل خودکار روابط بین جداول و ایجاد کلید خارجی (به کمک خاصیت virtual تعریف شده)

د) تشکیل جدول سیستمی به نام dbo.__MigrationHistory که از آن برای نگهداری سابقه به روز رسانی‌های ساختار جداول کمک گرفته خواهد شد.

ه) نوع و طول فیلدهای متنی، nvarchar از نوع max است.

تمام این‌ها بر اساس پیش فرض‌ها و قراردادهای توکار EF Code first انجام شده است.

در کدهای تعریف شده نیز، ابتدا یک وهله از شیء Context ایجاد شده و سپس به کمک آن می‌توان به جدول Blogs اطلاعاتی را افزود و در آخر ذخیره نمود. استفاده از using هم دراینجا نباید فراموش شود، زیرا اگر استثنایی در این بین رخ دهد، کار پاکسازی منابع و بستن اتصال گشوده شده به بانک اطلاعاتی به صورت خودکار انجام خواهد شد.
در ادامه اگر بخواهیم مطلبی را به Blog ثبت شده اضافه کنیم، خواهیم داشت:

```
using EF_Sample01.Models;

namespace EF_Sample01
{
    class Program
    {
        static void Main(string[] args)
        {
            //addBlog();
            addPost();
        }

        private static void addPost()
        {
            using (var db = new Context())
            {
                var blog = db.Blogs.Find(1);
                db.Posts.Add(new Post
                {
                    Blog = blog,
                    Content = "data",
                    Title = "EF"
                });
                db.SaveChanges();
            }
        }

        private static void addBlog()
        {
            using (var db = new Context())
            {
                db.Blogs.Add(new Blog { AuthorName = "Vahid", Title = ".NET Tips" });
                db.SaveChanges();
            }
        }
    }
}
```

متد db.Blogs.Find، بر اساس primary key بلاگ ثبت شده، یک وهله از آن‌را یافته و سپس از آن جهت تشکیل شیء Post و افزودن آن به جدول Posts استفاده می‌شود. متد Find ابتدا Contxet جاری را جهت یافتن شیءایی با id مساوی یک جستجو می‌کند (اصطلاحا به آن first level cache هم گفته می‌شود). اگر موفق به یافتن آن شد، بدون صدور کوئری اضافه‌ای به بانک اطلاعاتی از اطلاعات همان شیء استفاده خواهد کرد. در غیراینصورت نیاز خواهد داشت تا ابتدا کوئری لازم را به بانک اطلاعاتی ارسال کرده و اطلاعات شیء Blog متناظر با id=1 را دریافت کند. همچنین اگر نیاز داشتیم تا تنها با سطح اول کش کار کنیم، در EF Code first می‌توان از خاصیتی به نام Local نیز استفاده کرد. برای مثال خاصیت db.Blogs.Local بیانگر اطلاعات موجود در سطح اول کش می‌باشد.
نهایتا کوئری Insert تولید شده توسط آن به شکل زیر خواهد بود (لاگ شده توسط برنامه SQL Server Profiler):

```
exec sp_executesql N'insert [dbo].[Posts]([Title], [Content], [Blog_Id])
values (@0, @1, @2)
select [Id]
from [dbo].[Posts]
where @@ROWCOUNT > 0 and [Id] = scope_identity()',
N'@0 nvarchar(max) ,@1 nvarchar(max) ,@2 int',
@0=N'EF',
```

```
@1=N'data',
@2=1
```

این نوع کوئرهای پارامتری چندین مزیت مهم را به همراه دارند:
الف) به صورت خودکار تشکیل می‌شوند. تمام کوئری‌های پشت صحنه EF پارامتری هستند و نیازی نیست مرتبا مزایای این امر را گوشزد کرد و باز هم عده‌ای با جمع زدن رشته‌ها نسبت به نوشتن کوئری‌های نا امن SQL اقدام کنند.
ب) کوئرهای پارامتری در مقابل حملات تزریق اس کیوال مقاوم هستند.
ج) SQL Server با کوئری‌های پارامتری همانند رویه‌های ذخیره شده رفتار می‌کند. یعنی query execution plan محاسبه شده آن‌ها را کش خواهد کرد. همین امر سبب بالا رفتن کارآیی برنامه در فراخوانی‌های بعدی می‌گردد. الگوی کلی مشخص است. فقط پارامترهای آن تغییر می‌کنند.
د) مصرف حافظه SQL Server کاهش می‌یابد. چون SQL Server مجبور نیست به ازای هر کوئری اصطلاحا Ad Hoc رسیده یکبار execution plan متفاوت آن‌ها را محاسبه و سپس کش کند. این مورد مشکل مهم تمام برنامه‌هایی است که از کوئری‌های پارامتری استفاده نمی‌کنند؛ تا حدی که گاهی تصور می‌کنند شاید SQL Server دچار نشتی حافظه شده، اما مشکل جای دیگری است.

مشکل! ساختار بانک اطلاعاتی تشکیل شده مطلوب کار ما نیست.

تا همینجا با حداقل کدنویسی و تنظیمات مرتبط با آن، پیشرفت خوبی داشته‌ایم؛ اما نتیجه حاصل آنچنان مطلوب نیست و نیاز به سفارشی سازی دارد. برای مثال طول فیلدها را نیاز داریم به مقدار دیگری تنظیم کنیم، تعدادی از فیلدها باید به صورت not null تعریف شوند یا نام پیش فرض بانک اطلاعاتی باید مشخص گردد و مواردی از این دست. با این موارد در قسمت‌های بعدی بیشتر آشنا خواهیم شد.

نظرات خوانندگان

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۰:۵۴:۲۱

با سلام و خسته نباشید. امید است این سری مطالب هم مانند مطالب MVC فراتر از مقالات و کتب موجود باشد.
سه سوال:
1- چطور می توان با Code First برخی از موارد ابتدایی ایجاد بانک مانند Collation و Compatibility Level و Schema و User و Role را به DBMS ارسال کرد.
2- اگر از پروایدرهای دیگر مثلا برای MySQL یا Oracle استفاده شود، آیا Code First قادر است بدون هیچ تغییری نسبت به SQL Server کد را به بانکهای دیگر نگاشت کند؟ در مورد بانک های NOSQL چطور؟
3- آیا اگر این پروژه Code First را در یک هاست اشتراکی Deploy کنیم و در آن هاست برنامه Start شود (مثلا یک پروژه MVC)، آیا پروژه قادر خواهد بود به طور خودکار بانک را تولید نماید و دیگر نخواهیم بصورت دستی بانک و یوزر را در کنترل پنل هاست تعریف کنیم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۱:۵۹:۴۸

1 و 3 - در انتهای بحث عرض کردم در قسمت‌های بعدی خیلی از موارد رو توضیح خواهم داد. این قسمت اول و فقط یک «مقدمه» ابتدایی بود.
2 - EF با بانک‌های اطلاعاتی NoSQL کار نمی‌کند. ضمنا هستند بانک‌های اطلاعاتی NoSQL ایی که برای دات نت نوشته شده‌اند و از همان روز اول با کلاس‌ها و LINQ کار می‌کنید مانند RavenDB . طراحی فوق العاده‌ای داره (^) .
استفاده از EF Code first با سایر بانک‌های اطلاعاتی بجز مشتقات SQL Server نیز میسر است. برای آن‌ها نیاز به پروایدر مخصوص وجود دارد؛ مثلا: (^)

نویسنده: محمدی
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۲:۲۳:۲۳

تغییرات در کدها (دیتابیس) چگونه مدیریت می شوند(بروزرسانی) ؟ یکی از کارهای سخت بروز رسانی دیتابیس مشتریه امیدوارم EF راه مناسبی برای این موضوع داشته باشه.
مقادیر پیش فرض در دیتابیس کی و چگونه مدیریت می شوند؟
این سوالات تو ذهنم پیش اومد اینجا نوشتم که در قسمت های بعد جوابشون رو بگیرم .
مطئنم مثل همیشه چیزهای زیادی اینجا یاد میگیرم ، مرسی

نویسنده: MehdiPayervand
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۲:۴۱:۳۱

با عرض سلام و خسته نباشید، جناب مهندس با وجود اینکه معرفیتون در سطح خیلی بالایی هست ولی جای خالی مقایسه با NHibernate رقیب کهنه کار و اپن سورس EF خالیه، باز هم ممنون

نویسنده: علی قمشلویی
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۴:۵۹:۵۵

با سلام و تشکر در مورد استفاده از POCO نیز لطف کنید مطلب بزارید

نویسنده: Iman Amirdarabi
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۶:۲۲:۵۲

با سلام

ممنون از مطالب مفید شما.
من کلا با اصل code first مشکل دارم.
صرف وقت کم درسته برنامه سریع تر بالا می یاد ولی از طرف دیگه باید وقت بیشتری صرف توسعه مواردی کرد که توسط نویسنده فریم ورک پیش بینی نشده.
مشکل دومی که وجود داره در code first خیلی از ویژگی هایی که در هر دیتابیس وجو داره از دست داده می شه مثل استفاده از sp
در آخر یک سوال آیا شما حاضری یک برنامه پیچیده تحت وب با این مدل پیاده سازی کنی به عنوان یک مهندس صاحب نظر؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۶:۳۶:۴۵

- یکی از اهدف ORMها این است که برنامه رو مستقل از بانک اطلاعاتی پیاده سازی کنند؛ تا بتوان در صورت نیاز راحت به یک بانک اطلاعاتی دیگر سوئیچ کرد. من اینجا وقت نگذاشتم در مورد «چرا باید از ORM استفاده کرد» توضیح بدم مانند (^) یا مانند (^) و ... باز هم جستجو کنید هست.
- بله. عدم استفاده از یک ORM در پروژه این روزها اشتباه محض است.

نویسنده: Naser Tahery
تاریخ: ۱۳۹۱/۰۲/۱۴ ۱۹:۳۸:۱۴

بی صبرانه منتظر قسمت های بعدی هستم.
ممنون از شما

نویسنده: ZB
تاریخ: ۱۳۹۱/۰۲/۱۴ ۲۳:۲۷:۳۴

سلام آقای نصیری
ممنون بخاطر مطالب مفیدتون. EF یک ORM قوی هست ولی بعضی مواردش هست که به نظر بنده نوعی در حد این ORM نیست.که دو موردش رو در اینجا ذکر میکنم :
1) در برنامه های چند لایه اگر لایه ها بصورت پروژه های جداگانه در نظر گرفته شده باشند باید Connection String رو در لایه BL ، UI و DAL قرار داد که به نظر من از لحاظ امنیتی درست نیست. این بهتره که در UI ما اصلا ندونیم COnnection String چی هست. البته این مورد با کد نویسی قابل حل هست ولی در کل مناسب نیست
2) من در کی از پروژه های گروهی به این مسئله برخوردم. ما در قسمت DAL فلدرهای مختلف داشتیم و هر کسی در فلدر قسمتی که کار میکرد میخواست یک دیتا مدل داشته باشه. این کار باعث میشد که به تعداد دیتا مدلها ما Connection String داشته باشیم و اگر میخواستیم از یک Connection String استفاده کنیم اسم کلاسی که تولید میشد برای همه یکسان میبود (اما در Name Space های مختلف) .

به نظر من EF از لحاظ Connection String یک مقدار ضعف داره.تا نظر دوستان چه باشه. ممنون و موفق باشید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۴ ۲۳:۴۹:۴۵

Connection String کاری به دیتامدل نداره. پیش فرض آن نام کلاسی است که از DbContext مشتق می‌شود (در مطلب فوق توضیح دادم: «در هر دو حالت، name باید به نام کلاس مشتق شده از DbContext اشاره کند که در مثال جاری همان Context است.»).
نیازی هم نیست در سراسر پروژه تکرار شود. یکبار باید در فایل کانفیگ برنامه تعریف شود.
اطلاع داشتن از این قراردادهای توکار از اتلاف وقت جلوگیری می‌کند.

نویسنده: وحید نصیری

تاریخ: ۲۳:۵۱:۵۳ ۱۳۹۱/۰۲/۱۴

ضمن اینکه زمانیکه از ORM استفاده می‌کند لایه DAL همان ORM است و نیازی نیست کار اضافه‌تری انجام دهید. این لایه خودبخود لحاظ شده است.

نویسنده: رضا
تاریخ: ۱۰:۳۲:۲۰ ۱۳۹۱/۰۲/۱۵

بسیار عالی. من در مورد code first در خود سایت asp.enet خوندم . اما توضیحات شما بخصوص چون به زبان فارسی است خیلی در درک اونچه درست نفهمیده بودم کمک کرد . منتظر بخش های بعدی هستیم .

نویسنده: Salehi
تاریخ: ۱۱:۵۳:۱۹ ۱۳۹۱/۰۲/۱۵

برای اینکه بانک اطلاعاتی به طور خودکار تشکیل بشه،اجرای هر دستوری که به بانک مربوط میشه کافیه؟مثلا دستور select ؟ بعد از اون به ازای هربار اجرای یک دستور، وجود یا عدم وجود DB چک میشه؟

نویسنده: وحید نصیری
تاریخ: ۱۲:۰۳:۳۲ ۱۳۹۱/۰۲/۱۵

در قسمت دوم این سری تحت عنوان «استراتژی‌های مقدماتی تشکیل بانک اطلاعاتی در EF Code first» توضیح دادم.

نویسنده: Iman Amirdarabi
تاریخ: ۱۴:۳۸:۳۲ ۱۳۹۱/۰۲/۱۵

سلام استاد
وقتی قرار برنامه مستقل از دیتابیس باشه ویژگی های هر دیتابیس چی می شه
مثل sp ها ufn ها یا type هایی که مختص یک دیتابیس و ....

نویسنده: وحید نصیری
تاریخ: ۱۶:۱۶:۵۷ ۱۳۹۱/۰۲/۱۵

- این ویژگی‌ها رو می‌تونید فراموش کنید. چون مثلا SP در SQL Server CE وجود خارجی ندارد. اما برنامه‌ی نوشته شده با EF به راحتی می‌تونه با انواع و اقسام بانک‌های اطلاعاتی که پروایدر EF برای آن‌ها مهیا باشد، کار کند. برنامه شما به دیتابیس خاصی گره نمی‌خوره. اگر لازم بود راحت می‌تونید با تغییر پروایدر و تغییر کانکشن استرینگ، بدون نیازی به تغییر در کدهای خود، از یک بانک اطلاعاتی دیگر استفاده کنید.
- در EF Code first امکان استفاده از SP و امثال آن هم وجود دارد (در جای خودش توضیح خواهم داده به چه نحوی). البته در این حالت برنامه فقط مختص به SQL Server خواهد شد.

نویسنده: ZB
تاریخ: ۱۴:۰۰:۴۳ ۱۳۹۱/۰۲/۱۶

با تشکر از پاسختون اما باید عرض کنم همونطور که مطلع هستید Connection String در EF مثل Linq 2 SQL نیست که به یک رشته خلاصه باشه بلکه مسیرهای CSDL, SSDL, MSL را هم لازم دارد. بنابراین اگر چند دیتا مدل داشته باشیم مجبوریم که چند Connection String ذخیره کنیم. دومین مطلبی که باید عرض کنم اینه که شما براحتی به مشکل خورد برنامه در فقدان Connection String رو در لایه های بالاتر میتوانید تست کنید. البته شما استاد هستید برای دوستان تازه کار عرض میکنم که در یک Solution دو پروژه اضافه کنید یکی برای دیتا مدل و یکی هم برای واسط کاربر.چنانچه از پروژه واسط بخواهید توابعی رو از دیتا مدل صدا بزنید به شما ارور برخواهد گشت و شما باید Connection String رو به برنامه واسط اضافه کنید. با تشکر

نویسنده: وحید نصیری

تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۴:۲۲:۴۱

بحث من در اینجا EF Code first است. در اینجا شما دیگر با SSDL و غیره کاری ندارید.
در مقدمه عرض کردم روش‌های database first و model first هم بودند و هستند. این فرق می‌کنه. code first هست. بحث چیز دیگری است.

نویسنده: آرش
تاریخ: ۱۳۹۱/۰۲/۱۶ ۲۲:۲۲:۳۶

با درود و سپاس از شما - اگر مقدور بود لطفا یک توضیح کوچک در مورد LocalDBو تفاوت اون با sql ce بفرمایید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۷ ۰۰:۰۱:۴۹

یک جدول مقایسه‌ای اینجا هست در این مورد: (^)

نویسنده: Sirwan Afifi
تاریخ: ۱۳۹۱/۰۲/۲۲ ۰۰:۳۰:۳۰

سلام استاد خیلی ممنون بابت آموزشهاتون
یه سوال :

همونطور که توضیح دادید در کل ما سه نوع پروژه لازم داریم : 1- Domain Classes که حاوی Model های ما هست 2- DataLayer که حاوی کلاس Context می باشد و در نهایت پروژه خودمان

حال مشکل من اینجاست که در داخل کلاس Context که ایجاد کرده ام کلاس DbContext و رفرنس EF_Sample01.Models (نام پروژه رو همون EF_Sample01 گذاشتم یعنی داخل یک Solution این سه نوع پروژه رو دارم)رو نمی شناسه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۲ ۰۰:۳۷:۴۶

DbContext نیاز به ارجاعی به اسمبلی EF دارد که باید به این class libraryهای دیگر هم اضافه شود.

نویسنده: مشعل
تاریخ: ۱۳۹۱/۰۴/۰۴ ۱۱:۲۰

با سلام
تشکیل خودکار بانک اطلاعاتی و جداول برای من انجام نمی‌شود. در واقع چون دیتابیس مورد نظر که در Connection String نام برده شده وجود ندارد، برنامه من اصلا به دیتابیس کانکت نمی‌شود و با خطای زیر هنگام اجرای متد SaveChanges مواجه میشوم:

```
Cannot open database "EFTest" requested by the login. The login failed.
```

لطفا راهنمایی بفرمائید
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۴ ۱۱:۵۹

در مورد کانکشن استرینگ، ایجاد بانک اطلاعاتی و غیره در قسمت‌های بعدی بیشتر توضیح داده شده. اگر تعاریف رشته اتصالی شما به این نحو باشد:

```
<connectionStrings>
```

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

به این معنا است که کلاس Context شما به نحو زیر تعریف شده است:

```
public class ProductContext : DbContext
```

بنابراین جزو قرار دادهای توکار EF Code first است که name ذکر شده در قسمت تعریف رشته اتصالی در فایل کانفیگ، با نام کلاس مشتق شده از DbContext یکی باشد.
با این تعاریف باید برنامه کار کند (البته بر اساس نام کلاس‌های برنامه شما).
ضمنا login failed به این معنا است که رشته اتصالی اشتباه تعریف شده است. رشته فوق به یک بانک اطلاعاتی sql server و به وهله پیش فرض آن اشاره می‌کند و از نوع windows authentication است. این موارد را باید بر اساس تنظیمات سیستم خودتون تغییر بدید.

نویسنده: torisoft
تاریخ: ۱۳۹۱/۰۴/۱۰ ۲۳:۵۹

سلام جناب نصیری
سیلور 5 از code first پشتیبانی نمیکنه ؟ موقع نصب از nuget پیغام میده که

```
Could not install package 'EntityFramework 4.3.1'. You are trying to install this package into a
project that targets 'Silverlight,Version=v5.0', but the package does not contain any assembly
references that are compatible with that framework.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۰:۷

سیلورلایت به صورت مستقیم با هیچ نوع ORM ایی کار نمی‌کند (چون یک فناوری سمت کاربر است). اما شما در سمت سرور می‌تونید به کمک یک WCF سرویس و مشتقات مشابه آن با EF یا NH و غیره کار کنید و سپس نتیجه را در یک برنامه سیلورلایت مصرف کنید.

نویسنده: mina
تاریخ: ۱۳۹۱/۰۴/۲۸ ۱۳:۱۴

سلام ممنون از مطالب مفیدتون

من تازه 1 روزه شروع کردم code first کار کنم

تو یه مقاله داشتم می‌خوندم برای تعریف entity ها این association ها رو تو هر دو طرف virtual تعریف کرده بود

مثلا برای blog این طور نوشته بود

```
public virtual IList<post> posts { set; get; }
```

خودم هم این طور نوشتم تا این جا که فرقی ندیدم می‌شه توضیح بدید چه فرقی دارن

باز هم ممنونم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۸ ۱۳:۳۷

مراجعه کنید به قسمت 10 این سری، بحث lazy loading آن.

نویسنده: هوشنگ
تاریخ: ۱۳۹۱/۰۵/۰۱ ۱۲:۵۹

سلام ،
جناب نصیری آیا من میتونم از EF Code First با دیتابیس Oracle استفاده کنم یا خیر ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۱ ۱۳:۳۲

بله. نیاز به پروایدر خود شرکت اوراکل را دارد و کار می‌کند.

نویسنده: فربد صالحی
تاریخ: ۱۳۹۱/۰۵/۱۶ ۱۳:۲۶

با سلام. از بین دو کتابی که در ابتدا معرفی کردید، من code first رو مطالعه کردم. با توجه به اینکه قصد دارم دوره شما رو هم کامل مطالعه کنم، نیاز به مطالعه کتاب DbContext هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۶ ۱۳:۵۵

هرچقدر بیشتر مطالعه کنید بهتر است.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۱/۰۵/۱۷ ۰:۳۴

چه پارامترهایی در انتخاب Code First، Model First, DataBase First برای یک پروژه وجود دارد؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۵/۱۷ ۲:۵۶

اگر قبلا نخواندید ، خواندن این سوال و جواب خالی از لطف نیست.
چه دلیل قانع کننده ای وجود داره برای پروژه ای که از صفر شروع می‌شود از روشی غیر از Code First استفاده شود ؟ حتی برای پروژه هایی که پایگاه داده‌ی آنها وجود دارد هم ابزارهای مهندسی معکوس (جهت تولید خودکار موجودیت ها) وجود دارد.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۱/۰۵/۱۸ ۱:۱۲

اشاره‌ی خوبی بود، ولی بیشتر یه محک میخواستم که امکان تشخیص این وجود داشته باشه که برای چه پروژه‌ای از چه رهیافتی استفاده کنیم. البته مطالب سوال و جواب هم تا حدی کمک میکنه

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۰۷ ۱۵:۱۷

سلام مهندس نصیری، چرا این کد توی EF5 خطای کلید خارجی میده؟
کدش از کتاب Code First که معرفی کردین استفاده کردم اما کد خودتون خطا نداره

```
using System;
using System.Collections.Generic;
namespace ChapterOneProject
{
public class Patient
    {
        public Patient()
        {
            Visits = new List<Visit>();
        }

        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime BirthDate { get; set; }
        //[ForeignKey("AnimalTypeId")]

        public AnimalType AnimalType { get; set; }
        //public int AnimalTypeId { get; set; }

        public DateTime FirstVisit { get; set; }
        public List<Visit> Visits { get; set; }
    }

public class Visit
    {
        [Key]
        public int Id { get; set; }
        public DateTime Date { get; set; }
        public String ReasonForVisit { get; set; }
        public String Outcome { get; set; }
        public Decimal Weight { get; set; }

        //[ForeignKey("PatientId")]
        //public virtual Patient Patient { get; set; }
        public int PatientId { get; set; }
    }

public class AnimalType
    {
        public int Id { get; set; }
        public string TypeName { get; set; }
    }
}
```

کد کانتکست

```
public class VetContext : DbContext
    {
        public DbSet<Patient> Patients { get; set; }
        public DbSet<Visit> Visits { get; set; }
        //public DbSet<AnimalType> AnimalTypes { get; set; }
    }
```

و در تابع Main برنامه Console این نوشته شده اما خطا میده و ثبت نمی‌شه

```
var dog = new AnimalType { TypeName = "Dog" };
            var visit = new List<Visit>
                            {
                                new Visit
                                    {
                                        Date = new DateTime(2011, 9, 1),
                                        Outcome = "Test",
                                        ReasonForVisit = "Test",
                                        Weight = 32,
```

```
                                }
                            };
            var patient = new Patient
                                {
                                    Name = "Sampson",
                                    BirthDate = new DateTime(2008, 1, 28),
                                    AnimalType = dog,
                                    Visits = visit,
                                };
            using (var context = new VetContext())
            {
                context.Patients.Add(patient);
                context.SaveChanges();
            }
```

کدهای دیگه تست کردم مشکلی نداشت اما این مورد ؟

با profiler چک کردم خطای عدم توانایی در تبدیل نوع datetime2 به datetime میده

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۲/۰۷ ۱۵:۴۷

اشکالش پیدا کردم توی کتاب برای شی patient چون خصوصیت FirstVisit مقداردهی نشده و نباید تهی باشد بنابراین زمان اجرا نمی‌تواند تاریخ پیش فرض را تبدیل کند. با اضافه کردن خط زیر

```
....

BirthDate = new DateTime(2008, 1, 28),
<<< FirstVisit = new DateTime(2008,1,12), >>>
AnimalType = dog,

.....
```

مشکل حل شد.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۷ ۱۶:۳۸

DateTime در دات نت value type هست یعنی نال پذیر نیست مگر اینکه Nullable تعریف شود.

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۳:۲۲

چطوری من جداول رو به دیتابیس اضافه کنم

کلاس رو نوشتم و وب کانفیگ رو هم ست کردم مثل ef اسکریپت نداره چطوری ادد کنم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۳:۳۳

- جداول رو خودش اضافه می‌کنه به صورت خودکار؛ در اولین بار اجرای برنامه.
- برای سفارشی سازی یا تهیه اسکریپت، در قسمت‌های 4 و 5 این سری به تفصیل بحث شده.

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۴:۵۷

زمان اجرا این خطا رو میده

.The type initializer for 'System.Data.Entity.Internal.AppConfig' threw an exception

مشکل از کجاست

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۶:۴۸

- حداقل دو علت می‌تونه داشته باشه:

الف) از پروژه‌ای استفاده می‌کنید که از چند ماژول تشکیل شده. اولی به EF نگارش A ارجاع دارد دومی به EF نگارش B. همه این‌ها رو باید یک دست کنید.

ب) EF رو به روز کردید اما تعریف آن‌را در فایل کانفیگ به روز نکردید. برای مثال این تعریف قدیمی در فایل کانفیگ شما هست

```
<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=4.3.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
```

که در آن به EF 4.3 اشاره شده. بعد پروژه رو به EF 5 آپدیت کردید اما این مورد به روز نشده که باید به صورت زیر تغییر کند:

```
<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
```

ج) تعریف ConnectionStrings در فایل کانفیگ باید بعد از ConfigSections باشد و نه قبل از آن.

- ضمنا تمام مثال‌های این سری از اینجا قابل دریافت است.

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۸:۲۶

اقای نصیری مرسی که واقعا وقت میزاری.واقعا زکات علم تو میدی.

sectionname="entityFramework"type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,>
</"EntityFramework, Version=4.3.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089 خط بالا در app.config
استفاده میشه که مال ویندوزه من asp.net کار میکنم پس اضافه بود پاکش کردم درست شد
بازم تشکر میکنم

نویسنده: امیر
تاریخ: ۱۳۹۲/۰۱/۰۷ ۱۳:۵

با سلام
من میخام پی دی اف EF Code frist رو تا اخرین درس داشته باشم چیکار باید کرد لینکشو بی زحمت میزارین مثل Asp.net mvc

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۰۷ ۱۳:۹

لطفا مراجعه کنید به ابتدای گروه EF در سایت. لینک دریافت PDF کلیه مطالب گروه به صورت یکجا قرار دارد. این نکته در مورد سایر گروه‌های سایت هم صادق است.

نویسنده: محمد
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۲:۳۷

سلام

وقتی کانکش استرینگو به این صورت تعریف می‌کنم :

```
<configuration>
<configSections>
</configSections>
<connectionStrings>
<clear/>
<add name="Context" connectionString="Data Source=localhost;Initial Catalog=test;Integrated Security =
true" providerName="System.Data.SqlClient"/>
</connectionStrings>
<system.web>
<compilation debug="true"/></system.web>
</configuration>
```

این erorr میده :

An error occurred while getting provider information from the database. This can be caused by Entity Framework using an incorrect connection string. Check the inner exceptions for details and ensure that the connection string is correct.

علتش چی می‌تونه باشه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۹ ۱۲:۵۷

خودش گفته چکار کنی. باید به inner exceptions مراجعه کنید. این تازه سطح اول خطا است. این خطا تو در تو است و تمام خطاهای EF تو در تو طراحی شدن.

مابقی رو هم در یک انجمن پیگیری کنید. نیاز هست کدت باشه. نیاز هست مشخص باشه سطح دسترسی‌ات به کانکشن استرینگی که تعریف کردی برقرار است یا نه و خیلی از مسایل دیگر.

نویسنده: debuger
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۳:۱۹

سلام

با توجه به خطایی که گذاشته شده به نظر مشکل از ConnectionString هست و اگر مثال این قسمت رو انجام دادی و instance شما به غیر از (local) است هنگام نوشتن نام DataSource بایستی پرانتزها رو برداری

```
<add name="Context"
         connectionString="Data Source=.\sql2012;Initial Catalog=testdb2012;Integrated Security = true"
          providerName="System.Data.SqlClient"
           />
```

موفق باشید

نویسنده: ر.م
تاریخ: ۱۳۹۲/۰۲/۰۲ ۰:۵

سلام. اصلا یاد گرفتن code first خوب هست یا نه؟ آیا پیاده سازی‌های اون تو پروژه مشکلی ایجاد نمیکنه؟

در مقابل روش db first چه مزیتها و معایبی داره؟ بهترین منبع یادگیری برای ef کجاست؟

راستی مفهوم change tracking تو ef رو میتونید توضیح بدید؟

متشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۲ ۰:۳۵

- از اون بهتر نیست .
- قدم به قدم. عجله نکن. در قسمت 14 این سری بهش پرداخته شده.

نویسنده: ر.م
تاریخ: ۱۳۹۲/۰۲/۰۳ ۱۰:۵۱

سلام.
آقا ممنون از جواب .
ببینید من مطلب لینک رو خوندم . پرداخته به انتخاب orm.
اما حالا من صحبتم متمرکز هست روی خود ef. من منظورم اینه که کجا باید codeFirst استفاده بشه کجا dbFirst. آیا کار کردن روی یک پروژه بصورت codeFirst در آینده یعنی وقتی حجم دیتا و ارتباطات زیاد شد مشکلی نخواهد ساخت؟ سرعت کدام بهتر است؟ آیا با وجود قابلیتهای lambda , linq نیازی به ساخت storedProcedure سمت دستابیس اصلا داریم یا نه؟
متشکرم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۳ ۱۱:۱۵

یک موتور اصلی EF بیشتر وجود نداره. برای کار کردن با این موتور اصلی سه روش حداقل تدارک دیده شده:
الف) database first: مربوط به زمانیکه یک دیتابیس از پیش طراحی شده با ساختار جداول و ارتباطات آن موجود است. این روش، ساختار بانک اطلاعاتی شما رو مهندسی معکوس کرده و یک سری کد و دیاگرام را برای استفاده توسط موتور اصلی EF تولید می‌کنه.
ب) روش model first: در VS.NET می‌تونید طراحی‌های مرتبط با جداول، کلاس‌ها و روابط رو انجام بدید. کدهای اضافی برای کار با موتور EF و همچنین به روز رسانی بانک اطلاعاتی رو به صورت خودکار انجام خواهد داد.
ج) روش code first: در این روش دیگر خبری از طراح بصری نیست. کار با کدنویسی و طراحی کلاس‌ها و ایجاد روابط بین آن‌ها توسط برنامه نویس شروع می‌شود. نهایتا این کلاس‌ها توسط موتور EF استفاده خواهند شد. امکان تبدیل این کلاس‌ها به بانک اطلاعاتی متناظر و همچنین به روز رسانی خودکار بانک اطلاعاتی با تغییر ساختار کلاس‌ها هم پیش بینی شده. روش code first بهترین حالت است برای کسانی که نمی‌خواهند از انبوهی از کدهای تولید شده به صورت خودکار (حاصل از مهندسی معکوس یک بانک اطلاعاتی موجود) استفاده کنند و می‌خواهند کنترل بیشتری بر روابط و اختصاصی سازی آن‌ها داشته باشند. در این حالت می‌تونید بدون نیاز به یک بانک اطلاعاتی یک برنامه را کامل کنید (منهای مباحث تست سیستم).
روش code first در حال حاضر روشی است که بیشتر توسط تیم EF تبلیغ می‌شود و در حال توسعه است. مابقی رو هم کم کم دارند تبدیل می‌کنند به پوسته‌ای برای حالت code first. مثلا ابزار تهیه کردند برای مهندسی معکوس یک بانک اطلاعاتی موجود به روش code first. کد نهایی تمیزتری داره؛ چون کلاس‌ها را خودتان طراحی می‌کنید و توسط ابزارها به صورت خودکار تولید نمی‌شوند، کنترل بیشتر و نهایتا کیفیت بالاتری دارند. ساده است؛ درگیر نگهداری edmx model‌ها نخواهید بود. به روز رسانی بانک اطلاعاتی آن هم می‌تواند کاملا خودکار شود.

برای اطلاعات بیشتر در مورد مزایای این روش یا تاریخچه EF متن قسمت جاری را یکبار مطالعه کرده و قسمت‌های «مروری سریع بر تاریخچه Entity framework code first» و «مزایای EF Code first» را بررسی کنید.
+ کل قسمت EF از اینجا به صورت یک فایل PDF قابل دریافت است. در مورد اکثر مواردی که عنوان کردید به صورت مجزا بحث شده و توضیحات کافی ارائه شدن.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۶:۸

ممنون از مطلب خوبتون.
برای جدول‌های زیاد که توی یک DbContext اضافه شدند،برای اولین لود و ذخیره زمان خیلی زیادی گرفته میشه و البته pre-generating views هم فایده نداشته!و از طرف دیگه توی برنامه مجبورم برای اینکه دیتا رو از دیتابیس بگیرم یکبار

Reload (DbEntityEntry<TEntity>(entityToRefresh).Reload()) کنم.من نمی‌فهمم یعنی همه‌ی دیتا یکبار توی شروع برنامه از دیتابیس دریافت میشن که دفعه‌های بعدی از Cash خونده میشن؟!ممنون میشم اگر یکم در این مورد توضیح بدید و جایی رو معرفی کنید که بشه مقدار بیشتر در مورد first level cache خوند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۶:۲۹

سایت جاری از EF Code first استفاده می‌کنه. مشکلی هم با کارآیی آن وجود ندارد.
برای مسایل شخصی نیاز به بررسی کدهای شما، بررسی best practices، بررسی‌های ویژه توسط EF Profilers و همچنین code review هست. به عبارتی نیاز به مشاور خصوصی دارید.
موفق باشید

نویسنده: ahmadb7
تاریخ: ۱۳۹۲/۰۲/۳۱ ۸:۲۰

با سلام کدام را شما ترجیح می‌دهید EF یا NH و برای شروع کدام بهتر است با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۳۱ ۹:۳

توضیحات در اینجا

نویسنده: احمد احمدی
تاریخ: ۱۳۹۲/۰۴/۰۶ ۲۳:۳

من هم به همین مشکل خوردم. یکی از دلایلش نال بودن مقدار DataTime مدل هست. DbUpdateException : MVC/EF Code First : SaveChanges

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۷ ۰:۳۰

در EF تا Inner exception را بررسی نکنید، دلیل اصلی را مشاهده نخواهید کرد و نهایتا با سعی و خطا و حدس و گمان، پیش خواهید رفت.

نویسنده: محمدیوسف میرجلیلی
تاریخ: ۱۳۹۲/۰۵/۱۵ ۱۴:۵۷

سلام. در درس ششم کلاس‌های کانفیگ را در فضای‌نام Mappings تعریف کردیم. اگر پروژه شامل چند اسمبلی باشه (DomainClasses و DataLayer)، فضای‌نام Mappings و کلاس‌های مرتبط با اون را بهتره در کدوم یک از پروژه‌ها ایجاد کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۵ ۱۵:۱۱

در قسمت 12 این سری با مثال قابل دریافت توضیح داده شده.

نویسنده: امیر خلیلی
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۱:۲۱

یک سوال ابتدایی
آیا سرعت کار با دیتابیس و فراخوانی دیتا با استفاده از EF نسبت به Ado.Net و یا همان DataSet و DataReader بیشتر است ؟
یا فقط به خاطر یک سری مزیت‌های دیگه باید رو بیاریم به این تکنولوژی ؟
راستش من فقط سرعت کار برام مهمه !

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۱:۳۱

تمام ORMهای دات نتی در سطح پایین خودشون از ADO.NET استفاده می‌کنند. بنابراین پشت صحنه این‌ها، استفاده از Data Reader و امثال آن است، به همراه یک سری مزیت جانبی دیگر مانند: « 5 دلیل برای استفاده از یک ابزار ORM » و جلوگیری از اختراع چرخ‌هایی به شدت ناقص و معیوب مانند: « مروری بر کدهای کلاس SqlHelper » و همچنین امنیت توکار و پیش فرض لحاظ شده در آن‌ها مانند: « امنیت در LINQ to SQL »

نویسنده: محمد رضا خزائی
تاریخ: ۱۳۹۲/۱۰/۱۳ ۲۱:۱۸

با سلام؛ منم این مشکل رو دارم .برای بار اول دیتابیس رو نمیسازه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۳ ۲۱:۲۵

« وادار کردن EF Code first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه »

نویسنده: XPlan
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۱:۵۵

سلام
-1می خواستم بدونم برای مثال در کلاس Blog شما

```
public class Blog
    {
        public int Id { set; get; }
        public string Title { set; get; }
        public string AuthorName { set; get; }
        public IList<Post> Posts { set; get; }
    }
```

EF دقیقا چه زمانی (و با فراخوانی چه متد هایی) از اکسسورهای set و چه زمانی از get استفاده می‌کند؟
-2 اگر در همین کلاس Blog به هر دلیل نیاز باشد که از اکسسورهای خودکار #C استفاده نکنیم کلاس Blog چگونه خواهد شد؟لطفا این کلاس را بدون اکسسورهای خودکار باز نویسی کنید

```
get
{
return ?
}
set
{
//push calculated private field to db ?
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۲:۲۵

- با استفاده از امکانات Reflection دات نت، در زمان خواندن اطلاعات از بانک اطلاعاتی، از set و زمان دریافت اطلاعات از کاربر

و تشکیل کوئری SQL نهایی از get استفاده خواهد کرد.
- در قسمت سوم این سری، در مورد فیلدهای محاسباتی بحث شده « 6) NotMapped و DatabaseGenerated »

نویسنده: منصور جعفری
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۲:۵۲

سلام؛ آیا روش Code First برای ویندوز اپلیکشن هم استفاده میشه...؟
ممنونم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۳:۱۸

در هر فناوری مرتبط با دات نت که کل دات نت فریم ورک در دسترس باشد، قابل استفاده است. از WPF تا WinForms تا WCF و انواع و اقسام برنامه‌های وب؛ از ویندوز سرویس تا یک برنامه‌ی کنسول ساده.

نویسنده: ح مراداف
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۶:۵

سلام،
منظور شما اینه که باید با Nuget رفرنس Entity FrameWork رو روی هر سه پروژه (Domain Classes و و DataLayer و پروژه اصلی) نصب کنم ؟

من وب اپلیکیشن تازه داره کار می‌کنم و تا الان همش وب سایت کار می‌کردم، آیا بصورت پیش فرض EntityFrameWork توی پروژه‌ها وجود نداره و حتما باید با Nuget رفرنس اونو به پروژه اضافه کنیم ؟
(یعنی این dll با نصب ویژوال استودیو نصب نمیشه ؟! و باید از نوگت دانولدش کنیم ؟)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۶:۴۱

قدیمی هست نمونه موجود در آن. مگر اینکه از آخرین نگارش VS.NET به همراه آخرین به روز رسانی آن استفاده کنید. اطلاعات بیشتر در مطالب به روز رسانی به EF 6 : اینجا و اینجا
همچنین این پروژه به علت ذات سورس باز آن هر از چندگاهی مستقل از VS.NET به روز می‌شود. بنابراین همیشه آخرین نگارش آن‌را بهتر است از نیوگت دریافت کنید .

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۱۸:۱

با سلام
اگه یه پروژه باشه که دارای چند Class Library با نام های DataLayer و DomainClasses و ServiceLayer و Models باشه چطوری با Package Manager Console می‌تونم EF را برای هر پروژه نصب کنم؟ و اینکه چطوری می‌تونم دستور Install-Package EntityFramework.Migrations -Version 0.9.0.0 را فقط در پروژه DataLayer نصب کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۱۸:۲۶

پروژه پیش فرض را در کنسول پاورشل نیوگت باید تغییر بدید:

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۲۱:۵۰

با سلام
آیا باید برای هر پروژه از دستور Install-Package EntityFramework استفاده کنم؟
راهی نداره که نخوایم برای هر پروژه ای اونرا دانلود و نصب نماییم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۲۲:۱

نیوگت خیلی بهینه عمل می‌کند. بار اول که درخواست نصب بسته‌ای را می‌دهید، ابتدا یک کوئری ساده برای دریافت شماره آخرین نگارش موجود در مخزن سایت رسمی آن ارسال می‌کند. بعد این شماره نگارش را با کش محلی سیستم (فایل‌های قبلی دریافت شده آن) مقایسه می‌کند. اگر یکی بود، از کش محلی استفاده می‌شود (چیزی مجددا دریافت نخواهد شد)؛ در غیراینصورت بسته‌ی جدید را دریافت خواهد کرد.

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۲۲:۱۰

وقتی سعی می‌کنم که از دستور زیر استفاده کنم با خطای زیر روبه رو می‌شوم. PM> Install-Package EntityFramework

```
Install-Package : Could not connect to the feed specified at 'https://www.nuget.org/api/v2/'. Please
verify that the package source
(located in the Package Manager Settings) is valid and ensure your network connectivity.
At line:1 char:1
+ Install-Package EntityFramework
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Install-Package], InvalidOperationException
    + FullyQualifiedErrorId :
NuGetCmdletUnhandledException,NuGet.PowerShell.Commands.InstallPackageCommand
```

راه حل چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۲۲:۱۵

اینجا بحث شده: « خلاصه‌ای در مورد روش‌های دریافت فایل از سایت NuGet »

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۰:۴۰

همانطور که شما فرمودید که کلاس Context را در یک ClassLibrary جداگانه قرار بدیم و نیاز به ارجاعی به اسمبلیهای EF خواهد داشت. این یعنی من باید برای این ClassLibrary هم باید EF را بوسیله Nuget نصب کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۰:۴۳

کمی بالاتر در نظرات با تصویر پاسخ دادم.

نویسنده: مرتضی احمدی
تاریخ: ۱۳۹۲/۱۲/۲۸ ۱۵:۱۹

سلام

اگر در سازنده DbContext با مقدار ثابتی مثل base("connectionName") مقداردهی کنیم به کانکشن موردنظر وصل می‌شود ولی وقتی که این مقدار را Runtime ست می‌کنم عمل نمی‌کند و خطا میدهد کانکشن پیدانشد. در حالی که معادل نامی که RunTime ست شده است کانکشن استرینگی تعریف شده است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۸ ۱۷:۱۰

تنظیم رشته اتصالی در زمان اجرا:

```
var ctx = new MyContext();
ctx.Database.Connection.ConnectionString = "...";
```

نویسنده: حمید حسین وند
تاریخ: ۱۳۹۳/۰۱/۲۴ ۱۳:۲۱

سلام
میخوام بدونم فرق دو تا دستور زیر چیه با هم؟

```
public IList<Post> Posts { set; get; }
```

و

```
public ICollection<Post> Posts { set; get; }
```

و اینکه تا جایی که می‌دونم نباید فیلد اضافی به اسم این Propertyها در table ایجاد بشه اما توی کدهایی که من نوشتم (عین همین دو مورد) برای هر کدوم فیلد اضافی توی جدولم ایجاد میشه و مقدارش null هست.
مگر نه اینکه از این دو مورد برای دریافت اطلاعات اضافی از جدول مثلا Post استفاده میشه و لزومی برای درج اطلاعات هنگام ثبت وبلاگ جدید نیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۴ ۱۳:۳۱

در قسمت‌های بعدی به این مباحث پرداخته شده:
- فرق List و کالکشن و موارد مشابه در اینجا
- بررسی رابطه one-to-many در قسمت هفتم

نویسنده: رشوند
تاریخ: ۱۳۹۳/۰۱/۳۱ ۱۹:۴۷

سلام و با عرض تبریک روز زن به همه زنان ایران زمین.
با پیشنهادتان برای ارتباط با دیتابیس، این سری از آموزش‌ها رو شروع کردم.

سوال:
در قسمت *تشکیل خودکار بانک اطلاعاتی و افزودن اطلاعات به جداول*
1- مقدار Data Source و Initial Catalog رو از کجا باید پیدا کرد؟ یا بهتر بگویم کانکشن استرینگ رو چطوری می‌شه از SQL

بدست آورد؟

این کانکشن بعد از نصب SQL Server 2008 Enterprise :

-2 ما در ابتدای اموزش ی اد گرفتیم که برای شروع کار یک کنترلر و بعد اکشن و بعد ویو ایجاد کنیم و سپس پروژه رو اجرا کنیم (در حال کلی). حالا می‌خواستم بپرسم که برای اجرای (*در اولین بار اجرای کدهای زیر*) کلاس Program رو چگونه (کجا و چگونه بنویسیم) اجرا بگیریم تا دیتابیس ایجاد شود..؟

سپاس.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۳۱ ۲۰:۴۵

- رشته اتصالی به SQL Server حالت‌های مختلفی می‌تواند داشته باشد. اطلاعات بیشتر
Data Source آن معمولا نام کامپیوتر جاری است یا IP Server. چون در تصویر شما instance name خالی است، از همان وهله‌ی پیش فرض استفاده می‌شود. اگر مقدار داشت می‌شد computer_name/instance_name
Initial Catalog نام بانک اطلاعاتی مدنظر است که قرار است به آن متصل شوید (یا در اینجا به صورت خودکار ساخته شود).
Integrated Security = true به معنای استفاده از اعتبارسنجی ویندوزی است برای اتصال به SQL Server. یعنی کاربر جاری لاگین کرده به سیستم باید دسترسی لازم را برای کار با SQL Server داشته باشد.
- برای فراگیری یک فناوری جدید از برنامه‌های کنسول استفاده کنید و نه ASP.NET. این مباحث عمومی است بین فناوری‌های مختلف استفاده کننده از آن. در یک برنامه‌ی کنسول آغاز کار از متد Main است؛ در یک برنامه‌ی وب از متد Application_Start فایل global.asax.cs خواهد بود.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۲۵ ۱۹:۴۱

در بعضی موارد هنگام کار با EF چنین خطایی رخ میده

```
Validation failed for one or more entities. See 'EntityValidationErrors' property for more details
```

البته اگر من از طریق زیر عمل کنم هیچ اتفاق نمیفته و درج انجام میشه

```
db.Entry(message).State = EntityState.Added;
 db.SaveChanges();
```

حالا تفاوت این دو روش db.messages.add با db.entry چه تفاوتی داره؟
یکی سری کدها هم داخل نت برای catch کردن DbEntityValidationException وجود داره ولی هیچ وقت وارد catch نمیشه ، مثل اینکه نوع استثنا رخ داده متفاوته

نمونه کدهای موجود

```
try
{
    // Your code...
    // Could also be before try if you know the exception occurs in SaveChanges

    context.SaveChanges();
}
catch (DbEntityValidationException e)
{
    foreach (var eve in e.EntityValidationErrors)
    {
        Console.WriteLine("Entity of type \"{0}\" in state \"{1}\" has the following validation
errors:",
            eve.Entry.Entity.GetType().Name, eve.Entry.State);
        foreach (var ve in eve.ValidationErrors)
        {
            Console.WriteLine("- Property: \"{0}\", Error: \"{1}\"",
                ve.PropertyName, ve.ErrorMessage);
        }
    }
    throw;
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۲۵ ۲۰:۵۲

البته این مطلب اول هست. در مطالب بعدی در مورد «اعتبارسنجی» بیشتر بحث شده:

InnerException و Entity Framework

استثناهایی که باید حین استفاده از EF Code first بررسی شوند

اعتبارسنجی در Entity framework Code first

همچنین نیاز است با « رفتار متصل و غیر متصل در EF » آشنا شوید. این مورد در مطلب « کار با کلیدهای اصلی و خارجی در EF Code first » هم بیشتر بحث شده‌است.

تمام این‌ها در مسیر آموزشی EF Code first به ترتیب لیست شده‌اند؛ قسمت‌های «اعتبارسنجی و بررسی استثناءها» و «ردیابی تغییرات».

عنوان: **ارتقاء به Entity framework 6 و استفاده از بانک‌های اطلاعاتی غیر از SQL Server**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۲۶ ۱۷:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

برای ارتقاء برنامه‌های قدیمی به EF 6 (که با دات نت 4 به بعد سازگار است) دو حالت استفاده از نیوگت را در حین افزودن ارجاعات لازم به کتابخانه‌های مرتبط با EF باید مدنظر داشت:

الف) از نیوگت استفاده کرده‌اید

در این حالت فقط کافی است کنسول پاورشل نیوگت را در VS.NET گشوده و دستور update-package را صادر کنید. (1) به صورت خودکار آخرین نگارش EF دریافت شده و (2) همچنین فایل کانفیگ برنامه برای افزودن و به روز رسانی تعاریف مرتبط با نگارش 6 به روز گردیده و (3) همچنین اسمبلی اضافی و قدیمی System.Data.Entity.dll نیز حذف خواهد شد.

ب) اگر از نیوگت استفاده نکرده‌اید

ابتدا یک فایل متنی ساده را به نام packages.config با محتوای ذیل به پروژه خود اضافه کنید.

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="EntityFramework" version="5.0.0" targetFramework="net40" />
</packages>
```

سپس بر روی نام Solution در VS.NET کلیک راست کرده و گزینه فعال سازی Restore بسته‌های نیوگت را فعال کنید (انتخاب گزینه Enable NuGet Package Restore). در ادامه یکبار برنامه را Build کنید تا پوشه packages به صورت خودکار از اینترنت دریافت و بازسازی شود. اکنون دستور update-package را در کنسول پاورشل نیوگت صادر کنید. همان مراحل قسمت الف تکرار خواهند شد.

لازم به ذکر است، اگر پروژه شما از چندین زیر پروژه تشکیل شده است که هر کدام نیز ارجاعی را به اسمبلی EF دارند، باید فایل packages.config فوق را به این زیر پروژه‌ها نیز اضافه کنید. دستور update-package، زیر پروژه‌ها را نیز اسکن کرده و تمام ارجاعات لازم را به صورت خودکار به روز می‌کند. همچنین اسمبلی‌های قدیمی اضافی را نیز حذف خواهد کرد. به این ترتیب با تداخل نگارش‌های قدیمی و جدید EF مواجه نخواهید شد.

مشکلاتی که ممکن است با آن‌ها مواجه شوید:

الف) برنامه کامپایل نمی‌شود

تنها تغییری که جهت کامپایل برنامه باید انجام دهید، استفاده از فضاهای نام جدید بجای فضاهای قدیمی موجود در اسمبلی منسوخ و حذف شده System.Data.Entity.dll است. خود VS.NET قابلیت یافتن فضاهای نام مرتبط را دارد و یا اگر از Resharper نیز استفاده می‌کنید، این قابلیت بهبود یافته است. در کل مثلا System.Data.EntityState شده است System.Data.Entity.EntityState و امثال آن که به روز رسانی آن‌ها نکته خاصی ندارد .

ب) پروایدر بانک اطلاعاتی مورد استفاده یافت نمی‌شود

به صورت پیش فرض فقط پروایدر SQL Server به همراه بسته EF 6 است. حتی پروایدر SQL Server CE نیز با آن ارائه نمی‌شود .
اگر از SQL Server CE استفاده کرده‌اید، باید دستور ذیل را نیز پس از نصب EF 6 صادر کنید:

```
PM> Install-Package EntityFramework.SqlServerCompact
```

تا با خطای ذیل مواجه نشوید:

```
No Entity Framework provider found for the ADO.NET provider with invariant name
'System.Data.SqlServerCe.4.0'.
Make sure the provider is registered in the 'entityFramework' section of the application config file.
See http://go.microsoft.com/fwlink/?LinkId=260882 for more information.
```

استفاده از نیوگت به روشی که عنوان شد، فایل کانفیگ برنامه شما را جهت افزودن تعاریف پروایدرهای لازم، به روز می‌کند و این مورد در EF 6 الزامی است (حتما باید تعریف پروایدر در فایل کانفیگ موجود باشد).

ج) خطای عدم وجود کلید خارجی جدول Migration را دریافت می‌کنید

```
The foreign key constraint does not exist. [ PK_dbo.__MigrationHistory ]
```

تا EF 5 نام کلید اصلی جدول MigrationHistory به صورت PK___MigrationHistory می‌باشد.

```
ALTER TABLE [__MigrationHistory] ADD CONSTRAINT [PK___MigrationHistory] PRIMARY KEY ([MigrationId]);
```

در EF 6 این نام شده است PK_dbo.__MigrationHistory
برای حل این مشکل تنها کافی است دستورات ذیل را یکبار بر روی بانک اطلاعاتی خود صادر کنید تا نام مورد نظر به عنوان کلید اصلی جدول migration اضافه شود؛ در غیراینصورت اصلا برنامه اجرا نخواهد شد:

```
ALTER TABLE [__MigrationHistory] drop CONSTRAINT [PK___MigrationHistory];
ALTER TABLE [__MigrationHistory] ADD CONSTRAINT [PK_dbo.__MigrationHistory] PRIMARY KEY (MigrationId);
```

البته یکبار برنامه را اجرا کنید. اگر خطای نبود کلید اصلی یاد شده صادر شد، آنگاه دو دستور فوق را اجرا نمائید.

نظرات خوانندگان

نویسنده: rezaei
تاریخ: ۱۳۹۲/۰۷/۲۷ ۰:۶

با سلام
توی یک برنامه سه لایه که edmx تو لایه DAL وجود داره به چه صورت میشه فضاهای نام رو update کرد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۲۷ ۰:۵۷

- توضیحات فوق مربوط به EF Code first بود و حتی با VS 2010 نیز قابل پیاده سازی و استفاده است.
- برای حالت database first نیاز به *VS 2013* دارید تا از کلیه امکانات EF 6 استفاده کنید (یا باید به روز رسانی خاصی را برای VS 2012 نصب کنید). حالت Code first مستقل است از IDE (یک مزیت دیگر).
Entity Framework Designer همراه با VS 2012 فقط از EF 5 پشتیبانی می‌کند (در حالت پیش فرض) و از تغییرات انجام شده در EF 6 آگاه نیست. به همین جهت اگر با VS 2012 بخواهید با EF6 کار کنید (در حالت database first) باز هم همان اسمبلی قدیمی System.Data.Entity.dll را مورد استفاده قرار می‌دهد که در EF 6 اصلا کاربردی ندارد و با آن یکی شده است.
البته در کل می‌تونید با VS 2012 هم در حالت database first با EF 6 کار کنید ولی نیاز به یک سری تغییرات دستی خواهید داشت (EF قدیمی و همچنین اسمبلی اضافی یاد شده را باید دستی حذف کنید) و به علاوه از بهبودهای جدید آن (در حالت پیش فرض و بدون به روز رسانی) محروم خواهید بود.
Entity Framework Designer سورس باز نیست (برخلاف هسته EF) و جزئی از VS.NET است. قرار است در آینده این افزونه را هم سورس باز کنند تا بتوانند مستقل از چرخه طول عمر VS.NET، خود EF Database first را نیز به روز کنند.
ولی در کل اگر از Code first استفاده می‌کنید، EF6 حتی با VS 2010 هم سازگار است.
- زمانیکه با یک ORM کار می‌کنید (فرقی نمی‌کند به چه اسمی)، لایه DAL همان ORM هست. (دست به اختراع لایه‌های اضافی نزنید)

نویسنده: یک دوست
تاریخ: ۱۳۹۲/۰۷/۲۷ ۱۹:۵

البته برای استفاده کامل از امکانات entity6 در حالت‌های Model first و Database first در vs2012 می‌توانید vs2012 خود را با پکیج زیر به روز کنید: http://www.microsoft.com/en-us/download/details.aspx?id=40762
ماخذ

نویسنده: محسن شفیعی
تاریخ: ۱۳۹۲/۰۷/۳۰ ۹:۵۸

با تشکر از وقتی که میزارین برا این مقالات ارزنده .
من وقتی به Entity framework 6 ارتقاء داردم مهمترین مشکلم این بود که t4 palmate هایی که برا scaffolding استفاده میکردم دیگه جواب نمیداد . مضمون اورش این بود که این ورژن از Entity framework رو ساپورت نمیکنه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۳۰ ۱۰:۱۸

بله. مطابق توضیحات رسمی که دادند، فقط با VS 2013 و MVC 5 این گزینه خاص کار می‌کند. اگر نیاز دارید، اینجا رای بدید .

نویسنده: امیر خلیلی
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۷:۶

با سلام؛ من تازه 2 روزه با Entity Framework آشنا شدم. حالا به روی VS 2012 و از طریق manage nuget pachages و گزینه 6

Entity Framework را نصب کردم و برای قدم اول یک کلاس , یک data layer , و کانکشن استرینگ را هم طبق اون تنظیم کردم.
اما در زمان اجرا خطای Could not find schema را برای کانفیگ در کانکشن استرینگ میده. از دوستان کسی میتونه راهنمایی کنه؟
ممنون میشم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۸:۲۵

- برای اینکه از راه دور کسی بتواند به سؤالات شما پاسخ دهد، یک سری نکات را باید رعایت کنید . (برای مثال مشخص نیست تنظیمات رشته اتصالی شما چی هست؟ کجا و به چه صورت و ترتیبی تعریف شده. کدهای شما چطور تعریف شده‌اند که به این خطا رسیدید؟ اصل خطا، به صورت کامل و دقیق، به همراه stack trace آن چی هست؟ (ذکر اصل کامل خطا، مهم‌ترین قسمت پرسش شما باید باشد))
- این خطا عموما زمانی حاصل می‌شود که محل تعریف connectionStrings در فایل کانفیگ قبل از configSections باشد. ترتیب این‌ها مهم است.
- به علاوه صرفا تعریف یک کلاس لایه داده و رشته اتصالی *کافی نیست* . نیاز است مباحث **migration** را هم اضافه کنید. مراجعه کنید به سری EF Code first سایت و **5** قسمت اول آن‌را مطالعه کنید .

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۸/۰۲ ۱۲:۱۷

یه نکته حاشیه ای: "ناگت" به تلفظ اصلی نزدیک تره تا نوگت یا نیوگت

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۲ ۱۲:۲۳

«نیوگت» هست مطابق نظر خالقین اصلی آن .

نویسنده: رضایی
تاریخ: ۱۳۹۲/۰۹/۲۳ ۲۳:۵۳

با سلام؛ در صورت امکان مثالی از نحوه ایجاد بانک در اوراکل توسط code first رو قرار بدید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۴ ۰:۱۳

اصول کلی آن اینجا بحث شده: « EF Code first و بانک‌های اطلاعاتی متفاوت »

نویسنده: م ش
تاریخ: ۱۳۹۳/۰۳/۰۸ ۱۸:۵۸

با سلام،
من یک پروژه مطابق ساختاری که در این مطلب ذکر شده ایجاد کرده بودم که البته با EF5 ایجاد شده بود. پس از ارتقاء پروژه به EF6 مشکلی که بوجود آمده این است که در حین اجرای آزمون‌ها، بانک اطلاعاتی بهمراه جدول migration ایجاد می‌شود ولی جداول در بانک اطلاعاتی ساخته نمی‌شود و EF تلاش می‌کند که migrationها را بر روی جداول اعمال کند، در صورتیکه عملاً جداول ایجاد نمی‌شوند.
آیا از دوستان کسی با این مشکل مواجه شده است؟

نویسنده: سام میرزاقراچه
تاریخ: ۱۳۹۳/۰۵/۲۸ ۲۳:۴۸

با سلام؛ من از VS2013 و EF 6.1 استفاده می‌کنم ، مشکلی که وجود دارد این است که ، با توجه به اینکه از Migrations در لایه Datalayer.Migrations اسفاده شده در زمان شروع برنامه در لایه Web که MVC می‌باشد در قسمت Application_Start() از

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyDbContext, Configuration>());
```

استفاده شده که در ارسال هرگونه Query به لایه Serivce این error :

An exception occurred while initializing the database. See the **InnerException** for details. به وجود می‌آید و البته دیتابیس در SQL Server ایحاد شده و فقط در زمان ارسال هر گونه کوری به لایه سرویس این مورد پیش می‌آید .

اما با جایگرین شدن در قسمت Application_Start() از

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyDbContext, Configuration>());
```

به :

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

مشکل حل می‌شود و ارسال هر گونه کوری به لایه سرویس بدون مشکل کار کرده .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۲۹ ۰:۲۲

- InnerException و Entity Framework
- بررسی خطاهای متداول عملیات Migration در حین به روز رسانی پروژه‌های EF Code First

عنوان: **بروز رسانی استفاده از SqlServer Compact در Entityframework 6.0**
نویسنده: مهدی پایروند
تاریخ: ۱۳۹۲/۰۸/۱۵ ۱۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, NuGet, SQL Server CE, Entity Framework 6, Update-Package

تغییراتی در Entity framework 6 صورت گرفته و در ذیل لیستی از موارد آن آمده است. همچنین پیشتر در همین سایت نیز به آن‌ها اشاره‌ای شده که باز تولید پروایدرها برای نسخه جدید Entity framework یکی از آن‌ها می‌باشد:

```
Rebuilding EF Providers for EF6
Updating Applications to use EF6
EF Tools: adding EF6 support & enabling out-of-band releases
Async Query and Save
Connection Resiliency
Code-Based Configuration
Dependency Resolution
Interception/SQL logging
Custom implementations of Equals or GetHashCode on entity classes
Custom Code First Conventions
Code First Mapping to Insert/Update/Delete Stored Procedures
Configurable Migrations History Table
Multiple Contexts per Database
```

اکنون برای به‌روز رسانی به نسخه جدید، جهت ادامه استفاده از SqlServer Compact مواردی باید لحاظ شود که به آنها اشاره خواهیم کرد و قبل از آنها رعایت یک سری از پیشنیازها لازم است. برای مثال در وب کانفیگ برای استفاده از پروایدر SqlServer Compact بعنوان پروایدر پیش فرض باید قسمت مربوطه را به نحو ذیل تغییر داده باشیم:

```
<entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlCeConnectionFactory,
EntityFramework">
      <parameters>
        <parameter value="System.Data.SqlServerCe.4.0" />
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlServerCe.4.0"
type="System.Data.Entity.SqlServerCompact.SqlCeProviderServices, EntityFramework.SqlServerCompact" />
    </providers>
  </entityFramework>
```

حالا در کنسول نیوگت دستور زیر را برای به‌روزرسانی فقط Entity Framework وارد و اجرا میکنیم:

```
Update-Package EntityFramework
```

پیغام موفقیت آمیز بودن به‌روز رسانی در خروجی نیوگت ظاهر می‌شود

```
PM> Update-Package EntityFramework
Updating 'EntityFramework' from version '5.0.0' to '6.0.1' in project 'Hamid.Core.Model'.
Removing 'EntityFramework 5.0.0' from Hamid.Core.Model.
Successfully removed 'EntityFramework 5.0.0' from Hamid.Core.Model.
```

و نیز تاییدی برای اعمال تغییرات به‌روز رسانی Entity framework انجام میشود تا فایل کانفیگ پروژه را تغییر دهد:

این تغییرات شامل موارد ذیل می‌باشند (در صورت به‌روز رسانی دستی، منظور کپی پکیج بصورت دستی، اعمال تغییرات در کانفیگ‌ها مورد نیاز است):

```
<!-- 1. Change in <configuration><configSections> -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=4.4.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
<!-- 2. Add in <entityFramework><providers> -->
      <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
```

بعد از به‌روز رسانی Entityframework باید پکیج EntityFramework.SqlServerCompact برای ادامه استفاده از پروایدر نصب شود که با دستور نیوگت زیر این امر نیز میسر است:

```
PM> Install-Package EntityFramework.SqlServerCompact
```

حالا بدون مشکل می‌توان از پروژه بیلد گرفت و کار توسعه را ادامه داد.

عنوان: EF Code First #4
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۷ ۱۳:۴۹:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

آشنایی با Code first migrations

ویژگی Code first migrations برای اولین بار در EF 4.3 ارائه شد و هدف آن سهولت هماهنگ سازی کلاس‌های مدل برنامه با بانک اطلاعاتی است؛ به صورت خودکار یا با تنظیمات دقیق دستی.

همانطور که در قسمت‌های قبل نیز به آن اشاره شد، تا پیش از EF 4.3، پنج روال جهت آغاز به کار با بانک اطلاعاتی در EF code first وجود داشت و دارد:

1) در اولین بار اجرای برنامه، در صورتیکه بانک اطلاعاتی اشاره شده در رشته اتصالی وجود خارجی نداشته باشد، نسبت به ایجاد خودکار آن اقدام می‌گردد. اینکار پس از وهله سازی اولین DbContext و همچنین صدور یک کوئری به بانک اطلاعاتی انجام خواهد شد.

2) DropCreateDatabaseAlways : همواره پس از شروع برنامه، ابتدا بانک اطلاعاتی را drop کرده و سپس نمونه جدیدی را ایجاد می‌کند.

3) DropCreateDatabaseIfModelChanges : اگر EF Code first تشخیص دهد که تعاریف مدل‌های شما با بانک اطلاعاتی مشخص شده توسط رشته اتصالی، هماهنگ نیست، آن‌را drop کرده و نمونه جدیدی را تولید می‌کند.

4) با مقدار دهی پارامتر متد System.Data.Entity.Database.SetInitializer به نال، می‌توان فرآیند آغاز خودکار بانک اطلاعاتی را غیرفعال کرد. در این حالت شخص می‌تواند تغییرات انجام شده در کلاس‌های مدل برنامه را به صورت دستی به بانک اطلاعاتی اعمال کند.

5) می‌توان با پیاده سازی اینترفیس IDatabaseInitializer، یک آغاز کننده بانک اطلاعاتی سفارشی را نیز تولید کرد.

اکثر این روش‌ها در حین توسعه یک برنامه یا خصوصا جهت سهولت انجام آزمون‌های خودکار بسیار مناسب هستند، اما به درد محیط کاری نمی‌خورند؛ زیرا drop یک بانک اطلاعاتی به معنای از دست دادن تمام اطلاعات ثبت شده در آن است. برای رفع این مشکل مهم، مفهومی به نام «Migrations» در EF 4.3 ارائه شده است تا بتوان بانک اطلاعاتی را بدون تخریب آن، بر اساس اطلاعات تغییر کرده‌ی کلاس‌های مدل برنامه، تغییر داد. البته بدیهی است زمانیکه توسط NuGet نسبت به دریافت و نصب EF اقدام می‌شود، همواره آخرین نگارش پایدار که حاوی اطلاعات و فایل‌های مورد نیاز جهت کار با «Migrations» است را نیز دریافت خواهیم کرد.

تنظیمات ابتدایی Code first migrations

در اینجا قصد داریم همان مثال قسمت قبل را ادامه دهیم. در آن مثال از یک نمونه سفارشی سازی شده DropCreateDatabaseAlways استفاده شد.

نیاز است از منوی Tools در ویژوال استودیو، گزینه Library package manager آن، گزینه package manager console را انتخاب کرد تا کنسول پاورشل NuGet ظاهر شود.

اطلاعات مرتبط با پاورشل EF، به صورت خودکار توسط NuGet نصب می‌شود. برای مثال جهت مشاهده آن‌ها به مسیر packages\EntityFramework.4.3.1\tools در کنار پوشه پروژه خود مراجعه نمائید.

در ادامه در پایین صفحه، زمانیکه کنسول پاورشل NuGet ظاهر می‌شود، ابتدا باید دقت داشت که قرار است فرامین را بر روی چه پروژه‌ای اجرا کنیم. برای مثال اگر تعاریف DbContext را به یک اسمبلی و پروژه class library مجزا انتقال داده‌اید، گزینه Default project را در این قسمت باید به این پروژه مجزا، تغییر دهید.

سپس در خط فرمان پاور شل، دستور enable-migrations را وارد کرده و دکمه enter را فشار دهید.

پس از اجرای این دستور، یک سری اتفاقات رخ خواهد داد:

الف) پوشه‌ای به نام Migrations به پروژه پیش فرض مشخص شده در کنسول پاورشل، اضافه می‌شود.
ب) دو کلاس جدید نیز در آن پوشه تعریف خواهند شد به نام‌های Configuration.cs و یک نام خودکار مانند number_InitialCreate.cs
ج) در کنسول پاور شل، پیغام زیر ظاهر می‌گردد:

```
Detected database created with a database initializer. Scaffolded migration
'201205050805256_InitialCreate'
corresponding to current database schema. To use an automatic migration instead, delete the Migrations
folder and re-run Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
```

با توجه به اینکه در مثال قسمت سوم، از آغاز کننده سفارشی سازی شده DropCreateDatabaseAlways استفاده شده بود، اطلاعات آن در جدول سیستمی dbo.__MigrationHistory در بانک اطلاعاتی برنامه موجود است (تصویری از آن‌را در قسمت اول این سری مشاهده کردید). سپس با توجه به ساختار بانک اطلاعاتی جاری، دو کلاس خودکار زیر را ایجاد کرده است:

```
namespace EF_Sample02.Migrations
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
    using System.Linq;

    internal sealed class Configuration : DbMigrationsConfiguration<EF_Sample02.Sample2Context>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = false;
        }

        protected override void Seed(EF_Sample02.Sample2Context context)
        {
            //  This method will be called after migrating to the latest version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension method
            //  to avoid creating duplicate seed data. E.g.
            //
            //    context.People.AddOrUpdate(
            //      p => p.FullName,
            //      new Person { FullName = "Andrew Peters" },
            //      new Person { FullName = "Brice Lambson" },
            //      new Person { FullName = "Rowan Miller" }
            //    );
            //
        }
    }
}
```

```
namespace EF_Sample02.Migrations
{
    using System.Data.Entity.Migrations;

    public partial class InitialCreate : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "Users",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Name = c.String(),
                        LastName = c.String(),
                        Email = c.String(),
                        Description = c.String(),
                        Photo = c.Binary(),
                        RowVersion = c.Binary(nullable: false, fixedLength: true, timestamp: true,
storeType: "rowversion"),
                        Interests_Interest1 = c.String(maxLength: 450),
```

```
                        Interests_Interest2 = c.String(maxLength: 450),
                        AddDate = c.DateTime(nullable: false),
                    })
                .PrimaryKey(t => t.Id);

            CreateTable(
                "Projects",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Title = c.String(maxLength: 50),
                        Description = c.String(),
                        RowVesrion = c.Binary(nullable: false, fixedLength: true, timestamp: true,
storeType: "rowversion"),
                        AddDate = c.DateTime(nullable: false),
                        AdminUser_Id = c.Int(),
                    })
                .PrimaryKey(t => t.Id)
                .ForeignKey("Users", t => t.AdminUser_Id)
                .Index(t => t.AdminUser_Id);

        }

        public override void Down()
        {
            DropIndex("Projects", new[] { "AdminUser_Id" });
            DropForeignKey("Projects", "AdminUser_Id", "Users");
            DropTable("Projects");
            DropTable("Users");
        }
    }
}
```

در این کلاس خودکار، نحوه ایجاد جداول بانک اطلاعاتی تعریف شده‌اند. در متد تحریف شده Up، کار ایجاد بانک اطلاعاتی و در متد تحریف شده Down، دستورات حذف جداول و قیود ذکر شده‌اند.

به علاوه اینبار متد Seed را در کلاس مشتق شده از DbMigrationsConfiguration، می‌توان تحریف و مقدار دهی کرد.

علاوه بر این‌ها جدول سیستمی dbo.__MigrationHistory نیز با اطلاعات جاری مقدار دهی می‌گردد.

فعال سازی گزینه‌های مهاجرت خودکار

برای استفاده از این کلاس‌ها، ابتدا به فایل Configuration.cs مراجعه کرده و خاصیت AutomaticMigrationsEnabled را true کنید:

```
internal sealed class Configuration : DbMigrationsConfiguration<EF_Sample02.Sample2Context>
{
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
        }
```

پس از آن EF به صورت خودکار کار استفاده و مدیریت «Migrations» را عهده‌دار خواهد شد. البته برای این منظور باید نوع آغاز کننده بانک اطلاعاتی را از DropCreateDatabaseAlways قبلی به نمونه جدید MigrateDatabaseToLatestVersion نیز تغییر دهیم:

```
//Database.SetInitializer(new Sample2DbInitializer());
Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample2Context,
Migrations.Configuration>());
```

یک نکته:

کلاس Migrations.Configuration که باید در حین وهله سازی از MigrateDatabaseToLatestVersion قید شود (همانند کدهای فوق)، از نوع internal sealed معرفی شده است. بنابراین اگر این کلاس را در یک اسمبلی جداگانه قرار داده‌اید، نیاز است فایل را ویرایش کرده و internal sealed آن‌را به public تغییر دهید.

روش دیگر معرفی کلاس‌های Context و Migrations.Configuration، حذف متد Database.SetInitializer و استفاده از فایل app.config یا web.config است به نحو زیر (در اینجا حرف ` اصطلاحا back tick نام دارد. فشردن دکمه ~ در حین تایپ انگلیسی):

```
<entityFramework>
    <contexts>
      <context type="EF_Sample02.Sample2Context, EF_Sample02">
        <databaseInitializer
           type="System.Data.Entity.MigrateDatabaseToLatestVersion`2[[EF_Sample02.Sample2Context,
EF_Sample02],
                [EF_Sample02.Migrations.Configuration, EF_Sample02]], EntityFramework"
        />
      </context>
    </contexts>
  </entityFramework>
```

آزمودن ویژگی مهاجرت خودکار

اکنون برای آزمایش این موارد، یک خاصیت دلخواه را به کلاس Project به نام public string SomeProp اضافه کنید. سپس برنامه را اجرا نمائید.

در ادامه به بانک اطلاعاتی مراجعه کرده و فیلدهای جدول Projects را بررسی کنید:

```
CREATE TABLE [dbo].[Projects](
---...
 [SomeProp] [nvarchar](max) NULL,
---...
```

بله. اینبار فیلد SomeProp بدون از دست رفتن اطلاعات و drop بانک اطلاعاتی، به جدول پروژه‌ها اضافه شده است.

عکس العمل ویژگی مهاجرت خودکار در مقابل از دست رفتن اطلاعات

در ادامه، خاصیت public string SomeProp را که در قسمت قبل به کلاس پروژه اضافه کردیم، حذف کنید. اکنون مجددا برنامه را اجرا نمائید. برنامه بلافاصله با استثنای زیر متوقف خواهد شد:

```
Automatic migration was not applied because it would result in data loss.
```

از آنجائیکه حذف یک خاصیت مساوی است با حذف یک ستون در جدول بانک اطلاعاتی، امکان از دست رفتن اطلاعات در این بین بسیار زیاد است. بنابراین ویژگی مهاجرت خودکار دیگر اعمال نخواهد شد و این مورد به نوعی یک محافظت خودکار است که درنظر گرفته شده است.

البته در EF Code first این مساله را نیز می‌توان کنترل نمود. به کلاس Configuration اضافه شده توسط پاورشل مراجعه کرده و خاصیت AutomaticMigrationDataLossAllowed را به true تنظیم کنید:

```
internal sealed class Configuration : DbMigrationsConfiguration<EF_Sample02.Sample2Context>
{
        public Configuration()
        {
            this.AutomaticMigrationsEnabled = true;
            this.AutomaticMigrationDataLossAllowed = true;
        }
```

این تغییر به این معنا است که خودمان صریحا مجوز حذف یک ستون و اطلاعات مرتبط به آن‌را صادر کرده‌ایم.
پس از این تغییر، مجددا برنامه را اجرا کنید. ستون SomeProp به صورت خودکار حذف خواهد شد، اما اطلاعات رکوردهای موجود تغییری نخواهند کرد.

استفاده از Code first migrations بر روی یک بانک اطلاعاتی موجود

تفاوت یک دیتابیس موجود با بانک اطلاعاتی تولید شده توسط EF Code first در نبود جدول سیستمی dbo.__MigrationHistory است.
به این ترتیب زمانیکه فرمان enable-migrations را در یک پروژه EF code first متصل به بانک اطلاعاتی قدیمی موجود اجرا می‌کنیم، پوشه Migration در آن ایجاد خواهد شد اما تنها حاوی فایل Configuration.cs است و نه فایلی شبیه به number_InitialCreate.cs .
بنابراین نیاز است به صورت صریح به EF اعلام کنیم که نیاز است تا جدول سیستمی dbo.__MigrationHistory و فایل number_InitialCreate.cs را نیز تولید کند. برای این منظور کافی است دستور زیر را در خط فرمان پاورشل NuGet پس از فراخوانی enable-migrations اولیه، اجرا کنیم:

```
add-migration Initial -IgnoreChanges
```

با بکارگیری پارامتر IgnoreChanges، متد Up در فایل number_InitialCreate.cs تولید نخواهد شد. به این ترتیب نگران نخواهیم بود که در اولین بار اجرای برنامه، تعاریف دیتابیس موجود ممکن است اندکی تغییر کند.
سپس دستور زیر را جهت به روز رسانی جدول سیستمی dbo.__MigrationHistory اجرا کنید:

```
update-database
```

پس از آن جهت سوئیچ به مهاجرت خودکار، خاصیت AutomaticMigrationsEnabled = true را در فایل Configuration.cs همانند قبل مقدار دهی کنید.

مشاهده دستوارت SQL به روز رسانی بانک اطلاعاتی

اگر علاقمند هستید که دستورات T-SQL به روز رسانی بانک اطلاعاتی را نیز مشاهده کنید، دستور Update-Database را با پارامتر Verbose آغاز نمائید:

```
Update-Database -Verbose
```

و اگر تنها نیاز به مشاهده اسکریپت تولیدی بدون اجرای آن‌ها بر روی بانک اطلاعاتی مدنظر است، از پارامتر Script باید استفاده کرد:

```
update-database -Script
```

نکته‌ای در مورد جدول سیستمی dbo.__MigrationHistory

تنها دلیلی که این جدول در SQL Server البته (ونه برای مثال در SQL Server CE) به صورت سیستمی معرفی می‌شود این است که «جلوی چشم نباشد»! به این ترتیب در SQL Server management studio در بین سایر جداول معمولی بانک اطلاعاتی قرار نمی‌گیرد. اما برای EF تفاوتی نمی‌کند که این جدول سیستمی است یا خیر.
همین سیستمی بودن آن ممکن است بر اساس سطح دسترسی کاربر اتصالی به بانک اطلاعاتی مساله ساز شود. برای نمونه ممکن است schema کاربر متصل dbo نباشد. همینجا است که کار به روز رسانی این جدول متوقف خواهد شد.
بنابراین اگر قصد داشتید خواص سیستمی آن‌را لغو کنید، تنها کافی است دستورات T-SQL زیر را در SQL Server اجرا نمائید:

```
SELECT * INTO [TempMigrationHistory]
FROM [__MigrationHistory]
DROP TABLE [__MigrationHistory]
EXEC sp_rename [TempMigrationHistory], [__MigrationHistory]
```

ساده سازی پروسه مهاجرت خودکار

کل پروسه‌ای را که در این قسمت مشاهده کردید، به صورت ذیل نیز می‌توان خلاصه کرد:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Data.Entity.Migrations.Infrastructure;
using System.IO;

namespace EF_Sample02
{
    public class Configuration<T> : DbMigrationsConfiguration<T> where T : DbContext
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
    }

    public class SimpleDbMigrations
    {
        public static void UpdateDatabaseSchema<T>(string SQLScriptPath = "script.sql") where T :
DbContext
        {
            var configuration = new Configuration<T>();
            var dbMigrator = new DbMigrator(configuration);
            saveToFile(SQLScriptPath, dbMigrator);
            dbMigrator.Update();
        }

        private static void saveToFile(string SQLScriptPath, DbMigrator dbMigrator)
        {
            if (string.IsNullOrWhiteSpace(SQLScriptPath)) return;

            var scriptor = new MigratorScriptingDecorator(dbMigrator);
            var script = scriptor.ScriptUpdate(sourceMigration: null, targetMigration: null);
            File.WriteAllText(SQLScriptPath, script);
            Console.WriteLine(script);
        }
```

```
    }
}
```

سپس برای استفاده از آن خواهیم داشت:

```
SimpleDbMigrations.UpdateDatabaseSchema<Sample2Context>();
```

در این کلاس ذخیره سازی اسکریپت تولیدی جهت به روز رسانی بانک اطلاعاتی جاری در یک فایل نیز درنظر گرفته شده است.

تا اینجا مهاجرت خودکار را بررسی کردیم. در قسمت بعدی Code-Based Migrations را ادامه خواهیم داد.

نظرات خوانندگان

نویسنده: Naser Tahery
تاریخ: ۱۳۹۱/۰۲/۱۷ ۲۳:۰۲:۳۳

سلام و بسیار ممنون.
مفهومی به نام «Migrations» در EF 4.3 ارائه شده است . آیا این EF 4.3 توسط دات نت 4 پشتیبانی میشود؟
چون در زمینه ی وب ، هاست ها بیشتر از NET 4. را پشتیبانی نمیکنند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۸ ۰۰:۱۳:۲۱

- بله. این شماره نگارش کتابخانه است، نه خود دات نت. برای اینکه بتوانند به روز رسانی‌ها را سریعتر کنند، آن‌را از پروسه به روز رسانی‌های کل دات نت خارج کرده‌اند.
- هاست‌ها فعلا از دات نت 4.5 پشتیبانی نمی‌کنند. چون نگارش بعدی دات نت دراین تاریخ در مرحله بتا است.

نویسنده: peyman
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۳:۴۷

سلام آقای نصیری . مشکلی که دارم اینه که فقط Configuration.cs ایجاد میشه و اون یکی فایل ایجاد نمیشه ! مشکل از چی میتونه باشه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۵ ۱۴:۱۸

حتما تغییراتی رو تشخیص نداده. مراحلی که طی شده، کدهای شما، ساختار بانک اطلاعاتی و اطلاعات جدول migration باید بررسی شوند.

نویسنده: میثم
تاریخ: ۱۳۹۱/۰۴/۱۳ ۱۲:۵۹

سلام و بسیار ممنون از مطالب آموزشی زیبا و واضح شما . امیدوارم خداوند در هر دو جهان پاداش این کار خیر شما را بدهد.
در مورد اینکه هاست‌ها از دات نت 4.5 پشتیبانی نمی‌کنند و ما نمی‌توانیم به عنوان مثال از «Migrations» یا برخی امکانت دیگر استفاده کنیم . آیا راه حلی برای این مساله وجود دارد یا فعلا باید این امکانات را در برنامه‌های تحت وب استفاده نکنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۳ ۱۳:۲

شماره نگارش EF با شماره نگارش دات نت یکی نیست و مدتی هست که برای انتشار ارائه‌های منظم و در فواصل زمانی کمتر این سیاست رو در پیش گرفتن. EF 4.3 برای مثال مبتنی بر دات نت 4 است و نه دات نت 4 و نیم که در این زمان در نگارش نهایی قرار ندارد.

نویسنده: میثم
تاریخ: ۱۳۹۱/۰۴/۱۳ ۱۴:۱

ممنون از توضیح شما
من درک درستی از کامنت اول این پست و پاسخ شما نداشتم الان همه چیز روشن شد
تشکر فراوان

نویسنده: فربد صالحی
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۱:۳

اینطور که من متوجه شدم ، تا قبل از EF 4.3 ، جدولی به اسم EdmMetadata ساخته می‌شد که مدل رو به صورت hash نگهدای میکرد و فقط مشخص می‌شد که آیا مدل با بانک اطلاعاتی منطبق هست یا نه. اما چون نیاز به نگهداری اطلاعات بیشتری برای migration بود، الان جدول __MigrationHistory تولید میشه.

نویسنده: davmszd
تاریخ: ۱۳۹۱/۰۷/۱۴ ۱۹:۸

با سلام؛
من بعد از این دستورات رو تو پاور شل زدم به همچین خطایی برخوردم، جریان چیه ؟

```
The term 'enable-migrations' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, veri
fy that the path is correct and try again.
At line:1 char:18
+ enable-migrations <<<<
    + CategoryInfo          : ObjectNotFound: (enable-migrations:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۴ ۱۹:۲۱

فایل‌های مرتبط با migrations فقط از طریق NuGet به همراه بسته EF دریافت می‌شوند. بنابراین داشتن فایل‌های DLL مربوط به EF کافی نیست. بعد از آن، این فرامین از طریق پاورشل خود NuGet در vs.net باید اجرا شوند.
در کل اگر می‌خواهید بدانید این بسته درست نصب شده یا نه، دستور زیر را در پاورشل خود NuGet اجرا کنید:
Get-Package

نویسنده: davmszd
تاریخ: ۱۳۹۱/۰۷/۱۶ ۱۶:۴

ممنون از پاسختون با این سرعت :)
فک کنم مشکل از اینجا بود که من DLL ها EF رو خودم دستی اضافه کرده بودم و ....
یه بار تمام ارجاع‌ها ش تو پروژه رو حذف کردم و بااستفاده از NUGET دوباره نصبش کردم البته NUGET Manager رو هم با استفاده از Extension Manager به روز رسانی کردم
بازم ممنون از زحماتتون جناب نصیری

نویسنده: حسین
تاریخ: ۱۳۹۱/۰۷/۲۲ ۱۵:۱۹

واقعا عالی بود ممنون

نویسنده: نوید
تاریخ: ۱۳۹۱/۱۲/۰۳ ۱۲:۵۸

سلام
ابتدا از مطالب مفیدتون سپاسگزارم.
من مدل هارو در یک Class Library جداگانه و context رو هم در یک Class Library جداگانه قرار دادم و یک Reference از اون‌ها به پروژه اصلی اضافه کردم.
الان وقتی در Package manager Console دستور enable-migrations رو وارد میکنم خطای 'No context type was found in the assembly 'Online_Store.

رو میده که Online_Store نام پروژه اصلیم است.
ممنون میشم راهنماییم کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۳ ۱۳:۲۹

در متن توضیح دادم:
«... ابتدا باید دقت داشت که قرار است فرامین را بر روی چه پروژه‌ای اجرا کنیم. برای مثال اگر تعاریف DbContext را به یک اسمبلی و پروژه class library مجزا انتقال داده‌اید، گزینه *Default project* را در این قسمت (Nuget package manager console) باید به این پروژه مجزا، تغییر دهید . .. »

نویسنده: نوید
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۱:۳۰

ممنونم از راهنماییتون.
با این کار فایل Configuration ایجاد میشه ولی اون فایل دوم ایجاد نمیشه . وقتی بقیه دستورات رو هم اجرا میکنم(Update-datebase و ...) هم خطای
An error occurred while getting provider information from the database. This can be caused by Entity Framework using an incorrect connection string. Check the inner exceptions for details and ensure that the connection string is correct.

رو میده .
تو فایل app.config، هم Connection string رو اضافه کردم ولی باز هم همین خطا رو میده!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۱:۴۸

قسمت اول را در مورد نحوه صحیح تعریف رشته اتصالی مجددا مطالعه کنید .

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۳:۹

سلام من هر کاری میکنم نمیتونم مشکلم حل کنم این خطا رو میده دوباره نصب کردم باز این خطا رو میده اگه می‌تونید یه کمکی بکنید
((The parameter is incorrect. (Exception from HRESULT: 0x80070057 (E_INVALIDARG

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۳:۲۶

اینجا انجمن نیست. من از راه دور نمی‌تونم به شما کمک کنم. نمی‌دونم چکار کردی، تنظیماتت چی هست.
اگر در حین کار با enable-migrations این خطا رو گرفتید، سعی کنید دقیق‌تر کار کنید:

```
enable-migrations -StartUpProjectName "prj name" -ContextTypeName "ctx name"
```

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۹:۶

با سلام
اینو خطا میده

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<Context,Migrations.Configuration>);
```

Migrations.Configuration,
نمیشناسه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۹:۱۳

سورس‌های این سری رو دریافت کنید . کلاس Migrations.Configuration یکی از کلاس‌های سفارشی تعریف شده در sample02 است.

نویسنده: Hamid NCH
تاریخ: ۱۳۹۲/۰۴/۲۳ ۱۲:۵۲

یه مشکلی که من برای بروزرسانی دیتابیسم توسط این روش دارم اینه که وقتی برای بار اول دستور Update-database رو اجرا میکنم دیتابیس بدون هیچ مشکلی ساخته میشه.اما اگه برای بار دوم و بیشتر این دستور اجرا بشه با خطای زیر مواجه میشم:

Sequence contains more than one element

حالا چه کلاسهام رو تغییر بدم چه ندم و این در صورتیه که پیکریندیم به این روش هست:

```
internal sealed class Configuration : DbMigrationsConfiguration<GlucosanContext>
  {
   public Configuration()
   {
    AutomaticMigrationsEnabled = true;
    AutomaticMigrationDataLossAllowed = true;
   }
```

و همچنین تو کلاس Program این دستور رو نوشتم:

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<GlucosanContext,
Migrations.Configuration>());
```

پروژه بنده ویندوز فرم هست.باتشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۳ ۱۲:۵۷

زمانیکه از روش AutomaticMigrationsEnabled به همراه AutomaticMigrationDataLossAllowed استفاده می‌کنید (تنظیم شده به true البته)، نیازی نیست هیچ کار اضافه‌تری انجام بدید؛ همه چیز خودکار است. به روز رسانی ساختار بانک اطلاعاتی، کم و زیاد کردن فیلدها و غیره همگی خودکار است.
بنابراین اصلا نیازی نیست دستورات پاورشل را اجرا کنید و اگر قبلا اینکار انجام شده و یک سری فایل اضافی migration دارید، همه رو حذف کنید تا تداخل ایجاد نکنند.

نویسنده: Hamid NCH
تاریخ: ۱۳۹۲/۰۴/۲۳ ۱۳:۲۲

خیلی ممنون؛ اما بفرض مثال من یه فیلد به یکی از جدولهام اضافه می‌کنم.طبعا باید دیتابیس دوباره بروزرسانی بشه.و این عمل بروزرسانی اگه درست متوجه شده باشم طبق فرمایش شما با ست کردن AutomaticMigrationsEnabled به true باید انجام بشه.که نمیشه.یا اینکه باید دوباره دستور enable-database -force رو تو پاورشل اجرا کنم که این هم منجر به خطایی که عرض کردم میشه.
در کل بنده هربار که تغییری تو دیتابیسم میدم با اینکه دارم از Migration استفاده می‌کنم مجبورم که دیتابیس رو از sql server پاک کنم و دوباره ایجادش کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۳ ۱۴:۷

- شما نباید دستی تغییری در دیتابیس ایجاد کنید. این روش Code first است. تغییرات باید شامل افزودن خاصیت به کلاس‌ها باشند.
- نباید دستور پاورشلی رو اجرا کنید اگر AutomaticMigrationsEnabled فعال است؛ چون سبب بروز تداخل می‌شود.
- روش Code first، کار به روز رسانی بانک اطلاعاتی رو تا زمان اجرای اولین کوئری به تاخیر می‌اندازد (اینطوری طراحی شده تا آغاز برنامه سریع به نظر برسد). روش دیگری هم وجود داره تا این مساله رو تغییر داد:

« وادار کردن EF Code first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه »

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۵/۰۶ ۱۵:۴۶

سلام آقای نصیری؛

منم همین پیام را دریافت می‌کنم، ولی من EF نسخهٔ 5.0 را استفاده می‌کنم و اونو هم با NuGet به پروژه اضافه کرده‌م. ویژوال استدیو نسخهٔ 2012 هست و با دات‌نت 4.5 برنامه را ایجاد کرده‌م. برنامه تا آخر درس قبل کاملاً همونطور که انتظار می‌رفت اجرا شد و مشکلی هم نداشت.

اما به محض باز کردن package manager console از منوی Tools، بعد از معرفی نسخهٔ کنسول Package Manager Console Host Version 2.6.40627.9000 خطوط زیر را با زمینهٔ قرمز می‌نویسه:

```
Test-ModuleManifest : The specified module 'D:\Entity Framework Samples\EF Sample
02\packages\EntityFramework.5.0.0\tools\EntityFramework.psd1' was not loaded because no valid module
file was found in any module directory.
At D:\Entity Framework Samples\EF Sample 02\packages\EntityFramework.5.0.0\tools\init.ps1:14 char:34
+ $thisModule = Test-ModuleManifest <<<<  (Join-Path $toolsPath $thisModuleManifest)
    + CategoryInfo          : ResourceUnavailable: (D:\Entity Framework
Samples\E...yFramework.psd1:String) [Test-ModuleManifest], FileNotFoundException
    + FullyQualifiedErrorId :
Modules_ModuleNotFound,Microsoft.PowerShell.Commands.TestModuleManifestCommand

Import-Module : Cannot bind argument to parameter 'Name' because it is null.
At D:\Entity Framework Samples\EF Sample 02\packages\EntityFramework.5.0.0\tools\init.ps1:31 char:18
+     Import-Module <<<<  $thisModule
    + CategoryInfo          : InvalidData: (:) [Import-Module], ParameterBindingValidationException
    + FullyQualifiedErrorId :
ParameterArgumentValidationErrorNullNotAllowed,Microsoft.PowerShell.Commands.ImportModuleCommand
```

برای دستور enable-migrations هم دقیقاً همون چیزی را می‌نویسه که در کامنت اول **davmszd** بیان شده، یعنی اینو:

```
The term 'enable-migrations' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, verify that the path is
correct and try again.
At line:1 char:18
+ enable-migrations <<<<
    + CategoryInfo          : ObjectNotFound: (enable-migrations:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

و هنگامی هم که دستور Get-Packageرا اجرا می‌کنم، اینو می‌نویسه:

```
Id                              Version              Description/Release Notes
--                              -------              -------------------------
EntityFramework                 5.0.0                Entity Framework is Microsoft's recommended data
access technology for new applications.
```

ضمناً در پوشهٔ packages در کنار پروژه، فولدری بنام EntityFramework.5.0.0 و داخل اون هم فولدر tools با این فایل‌ها وجود داره:

```
about_EntityFramework.help.txt
```

```
EntityFramework.PowerShell.dll
EntityFramework.PowerShell.Utility.dll
EntityFramework.PS3.psd1
EntityFramework.psd1
EntityFramework.psm1
init.ps1
install.ps1
migrate.exe
Redirect.config
Redirect.VS11.config
```

ممنون میشم اگر منو راهنمایی بفرمایید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۶ ۱۷:۲۱

powershell ویندوز را اجرا کنید (خارج از ویژوال استودیو) . بعد در خط فرمان آن دستور زیر را وارد نمائید:

```
$psversiontable.psversion
```

اگر Major آن مساوی 2 بود، یعنی از نگارش 2 پاورشل در حال استفاده هستید که باید به روز شود به نگارش 3 .

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۵/۰۸ ۲۰:۲۹

از توجهتون سپاسگزارم.
مهندس جان من اینکار را انجام دادم و پاورشل را هم ارتقاء دادم، ولی مشکل برطرف نشد.
یه پروژهٔ جدید ایجاد کردم و همهٔ فایل‌های کد پروژهٔ قبلی (بغیر از app.config) را بهش اضافه کردم و بعد با استفاده از NuGet نسخهٔ آخر EF را به پروژه اضافه کردم. کانکشن استرینگ را به پروژه اضافه نکردم (کلاً به app.config دست نزدم) و دستور **enable-migrations** را اجرا کردم. دستور با موفقیت اجرا شد و اون چیزهایی که فرمودین به پروژه اضافه شد.
دیتابیس را به صورت دستی Drop کردم و یه بار برنامه را اجرا کردم و بعد از اون هم تغییراتی که به مدل دادم به دیتابیس اعمال شد و رکوردهای مربوط بهش داخل جدول dbo.__MigrationHistoryثبت شد.

رفتم به درس پنجم، و دوباره به محض بازکردن کنسول همون پیغامی را که توی کامنت اولم نوشتم می‌نویسه و وقتی هم می‌خوام که دستور Add-Migration را اجرا کنم، همون چیزی را می‌نویسه که قبلاً برای enable-migrations می‌نوشت. جالب اینکه برای خود دستور enable-migrations هم همینو می‌نویسه!برای دستور Update-Database هم همینطور، در حالی که برای پروژهٔ شما می‌نوشت که هم اکنون فعال هست.

فکرمی‌کنم مشکل از فایل **init.psd1** باشه که هنگام اضافه کردن EF در فولدر tools ایجاد میشه.
من فایل init.psd1 را که در پوشهٔ tools مربوط به پروژهٔ ایجاد شده توسط شما قرار داشت جایگزین فایل با همین نام که در پروژهٔ خودم بود کردم و مشکل برطرف شد. نمی‌دونم ایراد کار چیه؟ من قدم به قدم همونطور که شما نوشتین پیش رفتم و چیزی را هم تغییر نداده‌م. آیا ممکنه توی پروژه‌هایی که می‌خوام از EF استفاده کنم این مسئله دردسرساز بشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۸ ۲۰:۴۲

این مورد احتمالا یک باگ هست که اگر بهشون گزارش کنید بهتره تا برای همه اعمال شود. عنوان کنید بسته نیوگت دریافتی دات نت 4.5 فایل init.ps1 مشکل داری داره و اگر اون رو با یک نمونه از بسته نیوگت دات نت 4 اندکی قدیمی‌تر جایگزین کنم مشکلی نیست. تمام خطاها رو هم دقیقا گزارش بدید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۸ ۲۰:۵۸

ضمنا یک سری تجربه با این خطا در اینجا هم هست:
الف) عنوان شده آخرین نسخه بتا این مشکل رو نداره (Install-Package EntityFramework -IncludePrerelease)
ب) VS.NET را با دسترسی Admin اجرا کنید.
ج) یا یک نفر دیگر در اینجا عنوان کرده که پروژه حتما باید در VS.NET باز شده باشد.
د) یا شخص دیگری عنوان کرده که آدرس فایل‌های پروژه اگر مثلا یک [داشته باشد، کار نمی‌کند.

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۵/۰۹ ۵:۰

نمی‌دونم چطور تشکر کنم و از وقتی که برای حل شدن مشکل من گذاشته‌اید بی‌اندازه ممنونم.
مشکل حل شد! من همهٔ این چهار مورد را بررسی کردم و اشکال همون وجود کروشه در مسیر پروژه بود. من پروژه‌هام را در فولدری به نام [Projects] نگهداری میکنم و تغییر نام دادن اون حذف کروشه‌ها خطا را برطرف کرد.

ضمناً موارد دیگه را هم امتحان کردم:
مورد الف هم خطا را برطرف می‌کنه. یعنی به نظر میرسه که EF 6.0 این مشکل را نخواهد داشت.
مورد ب هیچ تأثیری نداشت و مورد ج هم که اصلاً مطرح نبود (چون پروژه داخل VS باز می‌شد.)

پاینده و پیروز باشید.

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۶/۰۳ ۱۶:۳۷

سلام آقای نصیری . من یه گیر اساسی کردم تو این مسئله مهاجرت دیتابیس. ببینین من دارم یه ساختاری به شکل ماژولار تو یه وب برای خودم درست می‌کنم . تمام این ماژول هام در نهایت میشن یک دی ال ال .و تمام مسائل مربوط به اونا به صورت خودکار توسط خود ماژول انجام و مدیریت میشه و ...
یکی از این مسائل، ایجاد و بروز رسانی دیتابیس هستش که من با همین Code First Migration ایجاد کردم. مشکل اینجاست که وقتی این ماژول هام اجرا میشن هر کدومشون میخوان دیتا بیسو آپدیت کنند و یا جداولشونو ایجاد کنند اما هر ماژولی میاد اول همه جداول دیتابیسو پاک میکنه بعد ماله خودشو ایجاد میکنه.
این Contexها و migrationهای من هر کدوم جداگانه تو یه دی ال ال هستند. من چطور به اینا حالی کنم که فقط یه سری جدول خاصو چک کنند اگه نبود ایجاد و اگر بود آپدیت کنند و در نهایت متد Seed شون رو فراخوانی کتنند و به بقیه جداولم کار نگیرن... .
من حتی از MigrateDatabaseToLatestVersion هم استفاده میکنم اما فایده نداره . چون گفتم Contexهای من از هم خبر ندارند و هر کدومشون فکر میکنه فقط دیتابیس ماله خودشه . میخوام برای migration تعریف کنم و فقط این مهاجرت رو روی یه سری تیبل خاص بررسی کنه . ممنون . منتظر جوابم ...

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۰۳ ۱۷:۳۴

یک کلاس Context و یک کلاس مهاجرت مرکزی درست کنید که با استفاده از Reflection تمام ماژول‌ها را بارگذاری کرده و تعاریف DbSetها را از روی آن‌ها بر اساس مثلا کلاس پایه‌ای که موجودیت‌های آن‌ها از آن ارث بری می‌کنند، ایجاد کند. در این مورد مطلب داریم در سایت:
خودکار کردن تعاریف DbSetها در EF Code first
افزودن خودکار کلاس‌های تنظیمات نگاشت‌ها در EF Code first

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۸/۰۸ ۱۱:۱۷

بر اساس دیتابیس موجود که دارای اطلاعات است مدل را می‌سازم می‌خواستم migration را فعال کنم. بر اساس دستورات پاور شل قبلا آموزش داده اید می‌خواستم ببینم چگونه می‌توان این کار را بصورت code base انجام داد. ظاهرا دستورات پاورش معادل code base هم دارند.
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۸ ۱۱:۵۰

از کدهای کلاس SimpleDbMigrations ذکر شده در انتهای مطلب استفاده کنید یا ایده بگیرید.
ضمنا سورس کامل ابزارهای migration نیز در دسترس است.

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۸/۱۱ ۱۰:۲۸

یعنی راهکار ساده‌تری وجود دارد که معادل سه دستور پاور شل زیر باشد

```
enable-migrations
add-migration Initial -IgnoreChanges
update-database
```

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۱ ۱۱:۱۶

- این‌ها ساده هستند و قابل اجرا بر روی دیتابیس ریموت (فقط باید ConnectionString را صریحا ذکر کنید):

```
Update-Database -StartUpProjectName "...name..." -ConnectionString "...data..." -ConnectionProviderName "System.Data.SqlClient"
```

- و بله. «ویژگی مهاجرت خودکار» را در برنامه فعال کنید و لذت ببرید. نیازی به هیچ دستور پاور شلی ندارد. هر زمان که مدل‌های شما تغییر کرد، به صورت خودکار ساختار بانک اطلاعاتی را در اولین اجرای بعدی برنامه، به روز می‌کند.
- « بازسازی جدول MigrationHistory با کد نویسی در EF Code first »

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۴:۳۹

سلام و وقتتون بخیر
ممنون بابت مطلبتون.من دقیقا طبق مطلب شما Migrations رو پیاده کردم.اما یه مشکل دارم.اونم اینه که وقتی دیتابیسم برا اولین بار میخواد ساخته بشه ،خطای لاگین میده:

```
Cannot open database "Test" requested by the login. The login failed.
Login failed for user 'sa'.
```

البته این خطا فقط در صورتی داده میشه که مهاجرت خودکار فعال شده باشه،اگر این خط رو کامنت کنم دیتابیس بدون مشکل ساخته میشه:

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<CommonContext, Configuration>());
```

با توجه به مطلب شما حدس می‌زدم به خاط نکته ای باشه که در مورد جدول dbo.__MigrationHistory گفتید.اما با استفاده از

این لینک جدول رو از اول به صورت غیر سیستمی میسازم اما بازم مشکل دارم.البته دیتا بیس ایجاد میشه،فقط جدول dbo.__MigrationHistory رو میسازه و بعدش خطایی که گفتم رو میده.
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۴:۵۶

قسمت connectionStrings در فایل کانفیگ (name و connectionString اضافه شده) را چطور تعریف کردید؟ (Cannot open database مشکل اتصال و مشکل سطح دسترسی اکانت مورد استفاده است). ضمنا محتوای inner exception داده شده را هم بررسی کنید.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۵:۷

connectionStrings رو به این دو صورت میگذارم:

```
<add name="TestContext"
          connectionString="Data Source=localhost;Initial Catalog=Test;User
ID=sa;Password=123;MultipleActiveResultSets=True;"
          providerName="System.Data.EntityClient" />
```

و

```
<add name="TestContext"
        connectionString="Data Source=localhost;Initial Catalog=Test;Integrated Security=True"
        providerName="System.Data.EntityClient" />
```

هر دو یه مشکل رو دارند.فکر نمی‌کنم چون اگر مشکل دسترسی اکانت بود منطقا بدون Migration هم باید خطا می‌داد؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۵:۱۶

- برای انجام اعمال مختلف در SQL Server، سطوح دسترسی مختلفی وجود دارند. یک کاربر می‌تواند دسترسی درج رکوردها را داشته باشد، اما دسترسی ایجاد یا تغییر ساختار بانک اطلاعاتی را نداشته باشد.
- در EF 6 این جدول MigrationHistory دیگر سیستمی نیست.
- یوزر sa دسترسی مدیریتی دارد (حالت اول). احتمالا در حالت دوم که یکپارچه با ویندوز است، اکانت وارد شده به سیستم نیز admin است؛ وگرنه دسترسی لازم را نخواهد داشت که دیتابیس ایجاد کند.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۵:۴۰

-الان یکبار دیگه یه پروژه کوچیک با EF6 ایجاد کردم و مشکلی نداشت.آیا امکانش هست مربوط به ورژن EFم باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۷:۱۵

ممکنه در نگارش‌های اولیه EF Code first از این نوع خطاها وجود داشته و بعدا برطرف شده. در مورد ارتقاء به EF 6 به این مطالب مراجعه کنید: « ارتقاء به Entity framework 6 و استفاده از بانک‌های اطلاعاتی غیر از SQL Server » و همچنین « بروز رسانی استفاده از SqlServer Compact در Entityframework 6.0 »

نویسنده: Behnam
تاریخ: ۱۳۹۲/۱۰/۰۳ ۱۷:۳

با سلام و عرض خسته نباشید
من از کد قسمت **ساده سازی پروسه مهاجرت خودکار** استفاده کردم، با EF6، مشکلی که هست اینه که وقتی یک فیلد رو کم یا زیاد میکنم پیغام زیر رو میده:

```
The model backing the 'Sample2Context' context has changed since the database was created. Consider
using Code First Migrations to update the database (http://go.microsoft.com/fwlink/?LinkId=238269).
```

ولی مثال قبلتون با استفاده از

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample2Context,
Migrations.Configuration>());
```

هیچ مشکلی نداره و اجرا میشه، علت خطای قبل از چه چیزی میتونه باشه؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۳ ۲۲:۴۳

با EF6 هم مشکلی مشاهده نشد:
Sample27.cs
مثال فوق از کلاس DbMigrator استفاده می‌کند. متد Test.RunTests آن‌را اجرا کنید؛ البته بعد از اضافه کردن Connection1 که در سورس ذکر شده. بانک اطلاعاتی جدیدی ساخته می‌شود. سپس به کلاس منو یک فیلد جدید اضافه کنید و مجددا برنامه را اجرا کنید. مشاهده خواهید کرد که این فیلد اضافه شده و خطایی صادر نمی‌شود.

نویسنده: اس ام
تاریخ: ۱۳۹۲/۱۲/۰۶ ۱۵:۵۴

سلام
آیا امکان این وجود داره که Connection string و تنظیمات مربوط به اون مثل نام کاربری دیتابیس و رمز عبور و نام دیتابیس ،رو موقع نصب اولین بار برنامه از کاربر دریافت کنیم؟ مثل دات نت نیوک،که همه عملیات به صورت داینامیک انجام بشه؟
و اینکه این عملیات فقط یک بار در هنگام اولین نصب برنامه انجام بگیره و در ادامه دیگه این کار انجام نشه؟
اگر بله! مکان قرار دادن کد ایجاد دیتابیس از روی مدل رو کجای برنامه قرار بدیم بهتره؟ مثلا تو پروژه MVC

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۶ ۱۶:۲۳

روش‌های زیادی برای تعیین رشته اتصالی در EF وجود دارند. این موارد به همراه نظرات و مطلب « نحوه‌ی وادار کردن EF Code first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه » بحث شدند. کدهای آن اگر قرار است در حین نصب اولیه اجرا شوند، می‌توانند در همان روال مثلا دکمه‌ی نصب یا آغاز به نصب قرار گیرند. ابتدا مثلا ctx.Database.Connection.ConnectionString مقدار دهی می‌شود و بعد نکته‌ی وادار سازی EF به ساخت بانک اطلاعاتی.
پس از انجام اینکار می‌توان این اطلاعات را در فایل کانفیگ برنامه برای استفاده‌های بعدی ذخیره کرد. کلاس WebConfigurationManager امکان ویرایش قسمت‌های مختلف فایل کانفیگ برنامه را می‌دهد.

نویسنده: اس ام
تاریخ: ۱۳۹۳/۰۱/۰۶ ۱۳:۴۶

سلام؛ من این مقاله رو خوندم و این پروژه رو تو MVC5 و vs2013 update1 و EF6 انجام دادم. همه چی درسته ولی کلاس

InitialCreate رو ایجاد نمیکنه. آیا باید این کلاس رو دستی ایجاد کنم؟ من مدل‌ها و context رو تو اسمبلی‌های جدا گانه گذاشتم.

Solution 'SMCodeFirst' (3 projects)
SM.DataLayer
Properties
References
Context
SMDbContext.cs
Migrations
Configuration.cs
App.config
packages.config
SM.DomainClasses
Properties
References
Entities
Project.cs
User.cs
App.config
packages.config
SM.Web
Properties
References
App_Data
App_Start
Controllers
Models
Views
Global.asax
packages.config
Web.config

در ضمن وقتی میخوام Enable-migrations رو انجام بدم اگه تو DataLayer به اسمبلی Web ارجاعی نداشته باشم، میگه Connectionstring درست نیست ولی کلاس مربوط به migration رو میسازه. ممنون میشم راهنمایی کنید. چون در نهایت باید در اسمبلی web به DataLayer ارجاعی داشته باشیم دیگه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۶ ۱۶:۴۳

- InitialCreate زمانی ایجاد خواهد شد که دیتابیسی موجود است و اکنون در مدل‌های شما تغییری حاصل شده‌است. اگر بار اول است، نیازی به آن نیست (ایجاد نخواهد شد) و در حین ایجاد اولیه بانک اطلاعاتی، تمام مراحل لازم طی می‌شوند.
- رشته اتصالی را در فایل کانفیگ پروژه‌ای که مهاجرت روی آن فعال می‌شود نیز قرار دهید.
+ تمام این دستورات پارامتر رشته اتصالی هم دارند:

```
Update-Database -Verbose
 -ConnectionString "CONNECTIONSTRING"
 -ConnectionProviderName "System.Data.SqlClient"
 -StartupProjectName WEBSITE_PROJECT -ProjectName MIGRATION_PROJECT
```

نویسنده: مصطفی
تاریخ: ۱۳۹۳/۰۷/۱۸ ۱۵:۵۷

من مراحل بالا رو رفتم اما مشکلی که هست اینه که دفعه اول که دستور update-databse رو اجرا می‌کنم دیتابیس ایجاد میشه اما دفعه دوم بااجرای این دستور پیغام خطای زیر میاد

```
.Cannot open database "*****" requested by the login. The login failed.
```

```
'*****'  Login failed for user
```

در واقع برنامه به دیتابیسی که خودش ساخته دسترسی نداره .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۸ ۱۸:۵۳

در خطای نهایی، نام کاربر مشخص شده. بررسی کنید آیا دسترسی کافی دارد یا خیر.
همچنین این اطلاعات را صریحا هم می‌شود مشخص کرد:

```
Update-Database -StartUpProjectName "...name..." -ConnectionString "...data..." -ConnectionProviderName
"System.Data.SqlClient"
```

نویسنده: علیرضا م
تاریخ: ۱۳۹۳/۰۷/۲۴ ۱۱:۵۲

سلام
در صورت نیاز به بررسی تطابق مدل با پایگاه داده در نرم افزار :

```
bool isCompatible = Context.Database.CompatibleWithModel(true);
```

نویسنده: زینب
تاریخ: ۱۳۹۳/۰۷/۲۶ ۱۴:۰

سلام و باتشکر

من migration را فعال کردم ولی کلاس دوم که حاوی متدهای up ,dwon هست را برام نساخته و زمانی که نوع خاصیت جدولی را

تغییر می‌دم یا جدولی را دستی حذف می‌کنم پیغام خطای زیر را می‌بینم برای اینکه این پیغام را نبینم چه کار کنم

.There is already an object named 'tablename' in the database

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۶ ۱۴:۵۹

مورد پنجم « بررسی خطاهای متداول عملیات Migration در حین به روز رسانی پروژه‌های EF Code First »

در قسمت قبل خاصیت AutomaticMigrationsEnabled را در کلاس Configuration به true تنظیم کردیم. به این ترتیب، عملیات ساده شده، اما یک سری از قابلیت‌های ردیابی تغییرات را از دست خواهیم داد و این عملیات، صرفا یک عملیات رو به جلو خواهد بود.

اگر AutomaticMigrationsEnabled را مجددا به false تنظیم کنیم و هربار به کمک دستوارت Add-Migration و Update-Database تغییرات مدل‌ها را به بانک اطلاعاتی اعمال نمائیم، علاوه بر تشکیل تاریخچه این تغییرات در برنامه، امکان بازگشت به عقب و لغو تغییرات صورت گرفته نیز مهیا می‌گردد.

هدف قرار دادن مرحله‌ای خاص یا لغو آن

به همان پروژه قسمت قبل مراجعه نمائید. در کلاس Configuration آن، خاصیت AutomaticMigrationsEnabled را به false تنظیم کنید. سپس یک خاصیت جدید را به کلاس Project اضافه نموده و برنامه را اجرا نمائید. بلافاصله خطای زیر را دریافت خواهیم کرد:

```
Unable to update database to match the current model because there are pending changes and
automatic migration is disabled. Either write the pending model changes to a code-based migration
or enable automatic migration. Set DbMigrationsConfiguration.AutomaticMigrationsEnabled to true
to enable automatic migration.
```

EF تشخیص داده است که کلاس مدل برنامه، با بانک اطلاعاتی تطابق ندارد و همچنین ویژگی مهاجرت خودکار نیز فعال نیست. بنابراین اعمال code-based migration را توصیه کرده است.

برای این منظور به کنسول پاورشل NuGet مراجعه نمائید (منوی Tools در ویژوال استودیو، گزینه Library package manager آن و سپس انتخاب گزینه package manager console). در ادامه فرمان add-m را نوشته و دکمه tab را فشار دهید. یک منوی Auto Complete ظاهر خواهد شد که از آن می‌توان فرمان add-migration را انتخاب نمود. در اینجا یک نام را هم نیاز است وارد کرد؛ برای مثال:

```
Add-Migration AddSomeProp2ToProject
```

به این ترتیب کلاس زیر را به صورت خودکار تولید خواهد کرد:

```
namespace EF_Sample02.Migrations
{
    using System.Data.Entity.Migrations;

    public partial class AddSomeProp2ToProject : DbMigration
    {
        public override void Up()
        {
            AddColumn("Projects", "SomeProp", c => c.String());
            AddColumn("Projects", "SomeProp2", c => c.String());
        }

        public override void Down()
```

```
        {
            DropColumn("Projects", "SomeProp2");
            DropColumn("Projects", "SomeProp");
        }
    }
}
```

مدل‌های برنامه را با بانک اطلاعاتی تطابق داده و دریافته است که هنوز دو خاصیت در اینجا به بانک اطلاعاتی اضافه نشده‌اند.
از متد Up برای اعمال تغییرات و از متد Down برای بازگشت به قبل استفاده می‌گردد. نام فایل این کلاس هم طبق معمول چیزی است شبیه به timeStamp_AddSomeProp2ToProject.cs .

در ادامه نیاز است این تغییرات به بانک اطلاعاتی اعمال شوند. به همین منظور دستور زیر را در کنسول پاورشل وارد نمائید:

```
Update-Database -Verbose
```

پارامتر Verbose آن سبب خواهد شد تا جزئیات عملیات به صورت مفصل گزارش داده شود که شامل دستورات ALTER TABLE نیز هست:

```
Using NuGet project 'EF_Sample02'.
Using StartUp project 'EF_Sample02'.
Target database is: 'testdb2012' (DataSource: (local), Provider: System.Data.SqlClient, Origin:
Configuration).
Applying explicit migrations: [201205061835024_AddSomeProp2ToProject].
Applying explicit migration: 201205061835024_AddSomeProp2ToProject.
ALTER TABLE [Projects] ADD [SomeProp] [nvarchar](max)
ALTER TABLE [Projects] ADD [SomeProp2] [nvarchar](max)
[Inserting migration history record]
```

اکنون مجددا یک خاصیت دیگر را مثلا به نام public string SomeProp3، به کلاس Project اضافه نمائید.
سپس همین روال باید مجددا تکرار شود. دستورات زیر را در کنسول پاورشل NuGet اجرا نمائید:

```
Add-Migration AddSomeProp3ToProject
Update-Database -Verbose
```

اینبار نیز یک کلاس جدید به نام AddSomeProp3ToProject به پروژه اضافه خواهد شد و سپس بر اساس آن، امکان به روز رسانی بانک اطلاعاتی میسر می‌گردد.

در ادامه برای مثال به این نتیجه رسیده‌ایم که نیازی به خاصیت public string SomeProp3 اضافه شده، نبوده است. روش متداول، باز هم مانند سابق است. ابتدا خاصیت را از کلاس Project حذف خواهیم کرد و سپس دو دستور Add-Migration و Update-Database را اجرا خواهیم نمود.
اما با توجه به اینکه مهاجرت خودکار را غیرفعال کرده‌ایم و هربار با فراخوانی دستور Add-Migration یک کلاس جدید، با متدهای Up و Down به پروژه، جهت نگهداری سوابق عملیات اضافه می‌شوند، می‌توان دستور Update-Database را جهت فراخوانی متد Down صرفا یک مرحله موجود نیز فراخوانی نمود.

نکته:

اگر علاقمند باشید که راهنمای مفصل پارامترهای دستور Update-Database را مشاهده کنید، تنها کافی است دستور زیر را در کنسول پاورشل اجرا نمائید:

```
get-help update-database -detailed
```

به عنوان نمونه اگر در حین فراخوانی دستور Update-Database احتمال از دست رفتن اطلاعات باشد، عملیات متوقف می‌شود. برای وادار کردن پروسه به انجام تغییرات بر روی بانک اطلاعاتی می‌توان از پارامتر Force در اینجا استفاده کرد.

در ادامه برای اینکه دستور Update-Database تنها یک مرحله مشخص را که سابقه آن در برنامه موجود است، هدف قرار دهد، باید از پارامتر TargetMigration به همراه نام کلاس مرتبط استفاده کرد:

```
Update-Database -TargetMigration:"AddSomeProp2ToProject" -Verbose
```

اگر دقت کرده باشید در اینجا AddSomeProp **2** ToProject بجای AddSomeProp **3** ToProject بکارگرفته شده است. اگر یک مرحله قبل را هدف قرار دهیم، متد Down را اجرا خواهد کرد:

```
Using NuGet project 'EF_Sample02'.
Using StartUp project 'EF_Sample02'.
Target database is: 'testdb2012' (DataSource: (local), Provider: System.Data.SqlClient, Origin:
Configuration).
Reverting migrations: [201205061845485_AddSomeProp3ToProject].
Reverting explicit migration: 201205061845485_AddSomeProp3ToProject.
DECLARE @var0 nvarchar(128)
SELECT @var0 = name
FROM sys.default_constraints
WHERE parent_object_id = object_id(N'Projects')
AND col_name(parent_object_id, parent_column_id) = 'SomeProp3';
IF @var0 IS NOT NULL
    EXECUTE('ALTER TABLE [Projects] DROP CONSTRAINT ' + @var0)
ALTER TABLE [Projects] DROP COLUMN [SomeProp3]
[Deleting migration history record]
```

همانطور که ملاحظه می‌کنید در اینجا عملیات حذف ستون SomeProp3 انجام شده است. البته این خاصیت به صورت خودکار از کدهای برنامه (کلاس Project در این مثال) حذف نمی‌شود و فرض بر این است که پیشتر اینکار را انجام داده‌اید.

سفارشی سازی کلاس‌های مهاجرت

تمام کلاس‌های خودکار مهاجرت تولید شده توسط پاورشل، از کلاس DbMigration ارث بری می‌کنند. در این کلاس امکانات قابل توجهی مانند AddColumn، AddForeignKey، AddPrimaryKey، AlterColumn، CreateIndex و امثال آن وجود دارند که در تمام کلاس‌های مشتق شده از آن، قابل استفاده هستند. حتی متد Sql نیز در آن پیش بینی شده است که در صورت نیاز به اجرای دستوارت خام SQL، می‌توان از آن استفاده کرد.
برای مثال فرض کنید مجددا همان خاصیت public string SomeProp3 را به کلاس Project اضافه کرده‌ایم. اما اینبار نیاز است حین تشکیل این فیلد در بانک اطلاعاتی، یک مقدار پیش فرض نیز برای آن درنظر گرفته شود که در صورت نال بودن مقدار خاصیت آن در برنامه، به صورت خودکار توسط بانک اطلاعاتی مقدار دهی گردد:

```
namespace EF_Sample02.Migrations
```

```
{
    using System.Data.Entity.Migrations;

    public partial class AddSomeProp3ToProject : DbMigration
    {
        public override void Up()
        {
            AddColumn("Projects", "SomeProp3", c => c.String(defaultValue: "some data"));
            Sql("Update Projects set SomeProp3=N'some data'");
        }

        public override void Down()
        {
            DropColumn("Projects", "SomeProp3");
        }
    }
}
```

متد String در اینجا چنین امضایی دارد:

```
public ColumnModel String(bool? nullable = null, int? maxLength = null, bool? fixedLength = null,
bool? isMaxLength = null, bool? unicode = null, string defaultValue = null, string defaultValueSql =
null,
string name = null, string storeType = null)
```

که برای نمونه در اینجا پارامتر defaultValue آن‌را در کلاس AddSomeProp3ToProject مقدار دهی کرده‌ایم.
برای اعمال این تغییرات تنها کافی است دستور Update-Database -Verbose اجرا گردد. اینبار خروجی SQL اجرا شده آن به نحو زیر است که شامل مقدار پیش فرض نیز شده است:

```
ALTER TABLE [Projects] ADD [SomeProp3] [nvarchar](max) DEFAULT 'some data'
```

تعیین مقدار پیش فرض، زمانیکه یک فیلد not null تعریف شده‌است نیز می‌تواند مفید باشد. همچنین در اینجا امکان اجرای دستورات مستقیم SQL نیز وجود دارد که نمونه‌ای از آن‌را در متد Up فوق مشاهده می‌کنید.

افزودن رکوردهای پیش فرض در حین به روز رسانی بانک اطلاعاتی

در قسمت‌های قبل با متد Seed که به همراه آغاز کننده‌های بانک اطلاعاتی EF ارائه شده‌اند، جهت افزودن رکوردهای اولیه و پیش فرض به بانک اطلاعاتی آشنا شدید. در اینجا نیز با تحریف متد Seed در کلاس Configuration، چنین امری میسر است:

```
namespace EF_Sample02.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    internal sealed class Configuration : DbMigrationsConfiguration<EF_Sample02.Sample2Context>
    {
        public Configuration()
        {
            this.AutomaticMigrationsEnabled = false;
            this.AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(EF_Sample02.Sample2Context context)
        {
            context.Users.AddOrUpdate(
                 a => a.Name,
```

```
                new Models.User { Name = "Vahid", AddDate = DateTime.Now },
                new Models.User { Name = "Test", AddDate = DateTime.Now });
        }
    }
}
```

متد AddOrUpdate در EF 4.3 اضافه شده است. این متد ابتدا بررسی می‌کند که آیا رکورد مورد نظر در بانک اطلاعاتی وجود دارد یا خیر. اگر خیر، آن‌را اضافه خواهد کرد در غیراینصورت، نمونه موجود را به روز رسانی می‌کند. اولین پارامتر آن، identifierExpression نام دارد. توسط آن مشخص می‌شود که بر اساس چه خاصیتی باید در مورد update یا add تصمیم‌گیری شود. دراینجا اگر نیاز به ذکر بیش از یک خاصیت وجود داشت، از anonymously type object می‌توان کمک گرفت new { p.Name, p.LastName } .

تولید اسکریپت به روز رسانی بانک اطلاعاتی

بهترین کار و امن‌ترین روش حین انجام این نوع به روز رسانی‌ها، تهیه اسکریپت SQL فرامینی است که باید بر روی بانک اطلاعاتی اجرا شوند. سپس می‌توان این دستورات و اسکریپت نهایی را دستی هم اجرا کرد (که روش متداول‌تری است در محیط کاری).
برای اینکار تنها کافی است دستور زیر را در کنسول پاورشل اجرا نمائیم:

```
Update-Database -Verbose -Script
```

پس از اجرای این دستور، یک فایل اسکریپت با پسوند sql تولید شده و بلافاصله در ویژوال استودیو جهت مرور نیز گشوده خواهد شد. برای نمونه محتوای آن برای افزودن خاصیت جدید SomeProp5 به صورت زیر است:

```
ALTER TABLE [Projects] ADD [SomeProp5] [nvarchar](max)
INSERT INTO [__MigrationHistory] ([MigrationId], [CreatedOn], [Model], [ProductVersion]) VALUES
('201205060852004_AutomaticMigration', '2012-05-06T08:52:00.937Z', 0x1F8B0800000........... '4.3.1')
```

همانطور که ملاحظه می‌کنید، در یک مرحله، جدول پروژه‌ها را به روز خواهد کرد و در مرحله بعد، سابقه آن‌را در جدول __MigrationHistory ثبت می‌کند.

یک نکته:
اگر دستور فوق را بر روی برنامه‌ای که با بانک اطلاعاتی هماهنگ است اجرا کنیم، خروجی را مشاهده نخواهیم کرد. برای این منظور می‌توان مرحله خاصی را توسط پارامتر SourceMigration هدف گیری کرد:

```
Update-Database -Verbose -Script -SourceMigration:"stepName"
```

استفاده از DB Migrations در عمل

البته این یک روش پیشنهادی و امن است:
الف) در ابتدای اجرا برنامه، پارامتر ورودی متد System.Data.Entity.Database.SetInitializer را به نال تنظیم کنید تا برنامه تغییری را بر روی بانک اطلاعاتی اعمال نکند.
ب) توسط دستور enable-migrations، فایل‌های اولیه DB Migration را ایجاد کنید. پیش فرض‌های آن را نیز تغییر ندهید.

ج) هر بار که کلاس‌های مدل برنامه تغییر کردند و پس از آن نیاز به به روز رسانی ساختار بانک اطلاعاتی وجود داشت دو دستور زیر را اجرا کنید:

```
Add-Migration AddSomePropToProject
Update-Database -Verbose -Script
```

به این ترتیب سابقه تغییرات در برنامه نگهداری شده و همچنین بدون اجرای دستورات بر روی بانک اطلاعاتی، اسکریپت نهایی اعمال تغییرات تولید می‌گردد.

د) اسکریپت تولید شده را بررسی کرده و پس از تائید و افزودن به سورس کنترل، به صورت دستی بر روی بانک اطلاعاتی اجرا کنید (مثلا توسط management studio).

نظرات خوانندگان

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۷:۳۸

آیا در نسخه نهایی باید تنظیمات مربوط به Migrations رو حذف کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۷:۴۱

فقط قسمت Database.SetInitializer را نال کنید تا برنامه مستقیما کار به روز رسانی ساختار بانک اطلاعاتی را انجام ندهد. بعد، از اسکریپت تولیدی مطابق روشی که در انتهای بحث توضیح دادم پس از بررسی‌های لازم استفاده کنید. بنابراین تنظیمی را لازم نیست حذف کنید. فقط باید با احتیاط جلو رفت و بررسی کامل اسکریپت تولیدی و سپس اجرای دستی آن روی بانک اطلاعاتی.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۲۳:۲۶

من همین کاری که گفتید کردم سایتو آپلود کردم
ولی این error میده
Exception Details: System.Data.SqlClient.SqlException: Cannot open database "DataLayer.Context.MedicallexiconContext" requested by the login. The login failed. Login failed for user 'ServerName\medicallexicon_web'.
تو stackoverflow هم مطرح کرم جوای نگرفتم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۲۳:۳۴

چندتا بحث هست در مورد این خطا:
- EF Code first زمانیکه مشاهده کنه دیتابیس تعریف شده در رشته اتصالی وجود خارجی ندارد، سعی در ایجاد آن خواهد کرد. شما در هاست‌ها عموما دسترسی dbo ندارید. یعنی دسترسی ساخت دیتابیس ندارید. به عبارتی باید یک دیتابیس خالی از پیش تعیین شده داشته باشید. اگر پنل خاصی دارید از آن استفاده کنید برای ساخت دیتابیس. اگر ندارید باید تماس بگیرید تا دیتابیس برای شما ایجاد شود.
- زمانیکه خطای یافت نشدن بانک اطلاعاتی DataLayer.Context.MedicallexiconContex را دریافت می‌کنید یعنی پیش فرض‌های EF Code first رو رعایت نکردید. در اینجا name ایی که در رشته اتصالی تعریف می‌کنید مهم است. به صورت پیش فرض این name باید همان نام کلاس Context شما باشد (صرفنظر از اینکه رشته اتصالی شما به چه بانک اطلاعاتی اشاره می‌کند).

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۲۳:۴۴

تو سرور خودم آپلود کردما.در ضمن میدونم اینجا مناسب سوال پرسیدن نیس شرمنده

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۲۳:۴۸

مهم این است که یوزری که قصد اتصال دارد چه دسترسی برای آن تعریف شده. آیا دسترسی ایجاد بانک اطلاعاتی را دارد. همچنین بحث name رشته اتصالی هم هست.
این مباحث رو من در قسمت‌های قبل، لابلای مطالب و کامنت‌ها توضیح دادم. باید وقت بگذارید مطالعه کنید.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۸ ۰:۵

چیزی که برای من جالبه اینکه من تو کانکشن استرینگم User مشخص کردم ولی error که به من میده مربوطه به ServerName\medicallexicon_web میگه با این user نمیتونم login کنم اصلا نمیدون این user باسه چی هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۸ ۰:۱۶

نه. این اصطلاحا instance نام داره. در sql server شما چندین instance می‌تونید تعریف کنید. اگر نام سرور به تنهایی ذکر شود یعنی وهله پیش فرض. در غیر اینصورت به این معنا است که یک سری تنظیمات امنیتی خاص بر روی وهله‌ای خاص اعمال شده و شما باید از آن استفاده کنید. (البته می‌تونه در خطای ذکر شده نام یوزر هم باشه .... باید رشته اتصالی رو ببینم که از چه حالت اعتبار سنجی استفاده می‌کنه)
در کل این بحث در اینجا ادامه پیدا نکند بهتر است. چون نه مشخص است تنظیمات سرور شما چی هست. نه رشته اتصالی شما رو من دیدم. نه تنظیمات برنامه و Context شما مشخص است و نه تعداد وهله‌های سرور. نه سطح دسترسی یوزر اتصالی شما و نه نوع اعتبار سنجی لاگین یوزر متصل به سرور (ویندوزی است یا مخصوص sql server است).

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۶:۴۳

آقای نصیری با سلام. من هر وقت برای بروزرسانی دیتابیس دستور Add-migration را در Package manager console وارد می‌کنیم ، همیشه در اسکریپت ایجاد شده من جدول reportparameter هم وجود دارد ، در حالیکه اصلاً تغییری در آن ایجاد نکردم.

```
namespace Dal.Ef.Migrations
{
using System;
using System.Data.Entity.Migrations;
public partial class AddActiveColumnToClassesTable : DbMigration
{
public override void Up()
{
CreateTable(
"dbo.ReportParameters",
c => new
{
Id = c.Int(nullable: false, identity: true),
CenterCode = c.String(maxLength: 10),
CenterTitle = c.String(maxLength: 100),
TermCode = c.String(maxLength: 10),
TermTitle = c.String(maxLength: 100),
MasulBarnamerizi = c.String(maxLength: 100),
ModirAmuzesh = c.String(maxLength: 100),
Term_Id = c.Int(nullable: false),
Center_Id = c.Int(nullable: false),
})
.PrimaryKey(t => t.Id)
.ForeignKey("dbo.Terms", t => t.Term_Id, cascadeDelete: true)
.ForeignKey("dbo.Centers", t => t.Center_Id, cascadeDelete: true)
.Index(t => t.Term_Id)
.Index(t => t.Center_Id);
AddColumn("dbo.Classes", "IsActive", c => c.Boolean(nullable: false));
}
public override void Down()
{
DropIndex("dbo.ReportParameters", new[] { "Center_Id" });
DropIndex("dbo.ReportParameters", new[] { "Term_Id" });
DropForeignKey("dbo.ReportParameters", "Center_Id", "dbo.Centers");
DropForeignKey("dbo.ReportParameters", "Term_Id", "dbo.Terms");
```

;("DropColumn("dbo.Classes", "IsActive

```
DropTable("dbo.ReportParameters");
        }
```

```
    }
}
```

در بالا فقط یک ستون به جدول classes ایجاد کردم ولی همیشه برای جدول reportParameter که در جدول نیز وجود دارد هم اسکریپت ایجاد میکند.
البته من یکبار بعد از آنکه جدول reportParameter را ایجاد کرده بودم ، دستی آن را از دیتابیس حذف کرده بودم.
متشکرم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۷:۲۳

یک روش کلی زمانیکه همه چیز را دستی به هم ریخته‌اید وجود دارد:
- جدول سیستمی MigrationHistory را از بانک اطلاعاتی حذف کنید.
- کلاس‌های قبلی migrations موجود را هم کلا حذف کنید (هرچیزی که وجود دارد).
حالا مراجعه کنید به قسمت چهارم «استفاده از Code first migrations بر روی یک بانک اطلاعاتی موجود» تا مجددا بتوانید جدول سیستمی MigrationHistory تازه‌ای را تولید کنید:

```
Add-Migration InitialMigration -IgnoreChanges
Update-Database
```

دستورات فوق به این معنا هستند که فرض می‌شود تطابق کاملی بین بانک اطلاعاتی و کلاس‌های مدل وجود دارند. اکنون جدول سیستمی MigrationHistory متناظری را تولید کن.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۸:۱۳

مهندس جان سلام.
مشکلم کاملاً برطرف شد.
بسیار ممنون از لطف شما. یاعلی.

نویسنده: سارا زرمهر
تاریخ: ۱۳۹۱/۱۱/۰۱ ۱۵:۳

سلام؛

با توجه به این قسمت مطالب **استفاده از DB Migrations در عمل**
من اگر پارامترSystem.Data.Entity.Database.SetInitializer رو null بدم با خطا مواجه میشم!

میشه یه نمونه کد برام مثال بزنید که به چه صورت باید این کد رو بنویسم؟
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۱ ۱۶:۲۶

چه خطایی دریافت می‌کنید؟
ضمنا تمام مثال‌های این سری را از اینجا می‌توانید دریافت کنید.

نویسنده: سارا زرمهر
تاریخ: ۱۳۹۱/۱۱/۰۲ ۱۱:۱۴

فکر کنم من صحیح کدمو ننو شتم.

این لینکی که دادید من چطوری کدهاشو دریافت کنم؟
کلیک که میکنم، کدشده به من نشون داده میشه.

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۱۱/۰۲ ۱۱:۲۴

کلیک راست کن، گزینه Save Link As رو انتخاب کن (Firefox) یا لینکش رو توی IDM کپی و دانلود کن.

نویسنده: سارا زرمهر
تاریخ: ۱۳۹۱/۱۱/۰۲ ۱۱:۳۸

تشکر استاد راد

نویسنده: میثم خوشقدم
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۸:۳۳

سلام

من هم مشکل ایشون رو دارم

ولی با کارهایی که شما فرمودین مشکل حل نشد

چندین بار جدول و کلاس‌های میگریشن رو حذف کردم ولی گاهی درست کار می‌کند و بعضی وقت‌ها هم خیر.

مشکل اینجاست که در Add_Migration متدهای Up و Down خالی است یعنی تغییری را تشخیص نداده اما در Update-database می‌خواهد مجددا جدول را ایجاد کند!

نویسنده: سید مهدی فاطمی
تاریخ: ۱۳۹۲/۰۵/۱۱ ۰:۲۲

ضمن تشکر از مطالبتون
آیا میشه این مراحلی رو که گفتید رو خود برنامه انجام بده و کاربر نهایی از اون اطلاعی نداشته باشه ؟
مثلا من یه برنامه دادم تحویل کاربر و دیتابیس اون هم حاوی اطلاعات هست حالا من به این نتیجه رسیدم که تغییری در دیتابیس بدم. طبق گفته شما من باید یه اسکریپت از تغییرات خودم درست کنم و تحویل کاربر بدم تا اونو ایجاد کنه اما من دنبال راهی می‌گردم که در برنامه وقتی کلاس‌ها رو تغییر دادم برنامه‌ی جدید رو تحویل کاربر بدم و کاربر بیاد برنامه قدیمشو حذف و برنامه‌ی جدید و نصب کنه و وقتی که برنامه‌ی جدید برای اولین بار اجرا شد تغییرات رو در دیتابیس قدیمی اعمال کنه بدون اینکه کاربر از پشت پرده اطلاعی داشته باشته

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۱ ۰:۵۳

- بله. مراجعه کنید به قسمت چهارم مباحث «فعال سازی گزینه‌های مهاجرت خودکار، آزمودن ویژگی مهاجرت خودکار و عکس العمل ویژگی مهاجرت خودکار در مقابل از دست رفتن اطلاعات» آن.
- امکان ردیابی تغییرات آن (منظور از ردیابی در اینجا، ثبت وقایع و ذخیره سازی خروجی SQL به روز رسانی ساختار بانک اطلاعاتی است) هم در همان قسمت چهارم ذیل مبحث «ساده سازی پروسه مهاجرت خودکار» مطرح شده.

نویسنده: حامد رشنو
تاریخ: ۱۳۹۳/۰۹/۰۳ ۱۶:۲۱

من از چند کانتکست استفاده میکنم و موقع استفاده از دستورات:

Add-Migration

Update-Database

به مشکل بر میخورم.

آیا برای استفاده از چند کانتکست باید برای هر کدوم یک DbMigration جداگانه ساخت؟ با قرار دادن T به جای اسم کانتکست در DbMigration هم مشکلم حل نشد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۰۳ ۱۸:۳۱

« استفاده از چندین Context در EF 6 Code first »

عنوان: **وادار کردن EF Code first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۰ ۹:۴۹
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

یکی از مواردی که در EF Code First سبب سردرگمی تازه‌کاران می‌شود (بارها در کامنت‌های سایت مطرح شده)، فراخوانی متد Database.SetInitializer و ... عدم تشکیل بانک اطلاعاتی در این لحظه است. تنها کاری که توسط متد Database.SetInitializer صورت می‌گیرد، مشخص سازی استراتژی نحوه آغاز بانک اطلاعاتی است و نه اجرای آن استراتژی. این اجرا تا زمانیکه اولین کوئری به بانک اطلاعاتی ارسال نشود مثلا فراخوانی context.Entity.Find، به تعویق خواهد افتاد. به همین جهت برای وادار کردن EF به ساخت بانک اطلاعاتی و یا اعمال تغییرات جدید به آن، می‌توان از نکته زیر استفاده کرد:

```
protected void Application_Start() {
     //...
     Database.SetInitializer(...همانند سابق...);
     using (var context = new MyContext()) {
          context.Database.Initialize(force: true);
     }
     //...
}
```

در اینجا در روال آغاز برنامه و پیش از اینکه رابط کاربری نمایان شود، توسط متد context.Database.Initialize، سبب اجرای اجباری استراتژی آغاز بانک اطلاعاتی خواهیم شد.

نظرات خوانندگان

نویسنده: ایلیا
تاریخ: ۱۳۹۱/۰۷/۱۰ ۱۰:۱۸

نکته مفیدی بود . سپاس

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۳۹۱/۰۷/۱۰ ۱۲:۹

خیلی مفیده. ممنون.
در ضمن میشه توی سازنده استاتیک کلاس هم این موارد رو قرار داد تا فورا بعد از شروع برنامه همه کارها انجام شود.

نویسنده: فرشید علی اکبری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۷:۵۷

با سلام
موقعی که برنامه شروع شده و سراغ کانکست میره که اونو چک کنه و دیتابیس رو بسازه این پیغام میده :
The target context 'Common_Infrastructure.ContextCentralSystem' is not constructible. Add a default constructor or provide an implementation of IDbContextFactory.

با وجودیکه من از این دوحالت استفاده کردم :

```
public ContextCentralSystem(DbConnection db)
            : base(db, true)
        {

        }
        public ContextCentralSystem(string connectionnamestring)
            : base(nameOrConnectionString: connectionnamestring)
        {

        }
```

و میخوام طبق مشخصاتی که سیستم ازکاربرگرفته کانکت کرده و دیتابیسم رو با نامه مثلا XX بسازه ولی این پیغام میگه حتما باید یک ctor پیش فرض پیاده کنی ... که با این وجود رشته‌ی منو قبول نمی‌کنه... اگه لطف کنین ممنون میشم ..... درضمن از فایل app.config هم نمی‌خوام استفاده کنم و برنامه دسکتاپ هستش.
تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۸:۵۶

یکی از روش‌های تعریف رشته اتصالی است:

```
public class CustomContext : DbContext
{
     public CustomContext() :  base("AppConfigConnectionStringName") { }

// or
public class CustomContext : DbContext
{
    public CustomContext() :
            base(@"Data Source=(local);Initial Catalog=MyDBName;Integrated
Security=True;Pooling=False") { }
```

روش دیگر :

```
var ctx = new MyContext();
ctx.Database.Connection.ConnectionString = "...";
```

و یا

```
Database.DefaultConnectionFactory =
new SqlConnectionFactory(@"Data Source=(local);Initial Catalog=MyDBName;Integrated
Security=True;Pooling=False");
```

و ...

نویسنده: سانای رحیمی
تاریخ: ۱۳۹۳/۰۵/۲۲ ۹:۲۸

سلام
من از الگوی UoW استفاده می‌کنم. حالا اومدم داخل Context خودم یک متد InitializeDatabase نوشتم که دستورات بالا رو داخلش قرار دادم. ولی زمانی که فراخوانی می‌کنم هیچ اتفاقی نمی‌افته. برنامه خطا نمیده ولی اجرا که می‌کنم خود به خود بسته میشه، بدون اینکه خطایی بده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۲۲ ۹:۴۷

- این کدها دقیقا به همین شکلی که در اینجا مشخص شده باید اجرا شوند. ارتباطی با مباحث UoW ندارند. راسا و مستقلا توسط EF مدیریت می‌شوند و در اصل از یک Context درونی مخصوص این کار استفاده می‌کنند که با Context اصلی برنامه یکی نیست. نامش HistoryContext است.
- اگر برنامه‌ای خودبخود بسته می‌شود، ابتدا به Event viewer توکار ویندوز مراجعه کنید. به احتمال زیاد لاگ خطای آن در آنجا قابل مشاهده است. همچنین خطاهای مدیریت نشده را هم باید بررسی کنید.

عنوان: **بررسی خطاهای متداول عملیات Migration در حین به روز رسانی پروژه‌های EF Code First**
نویسنده: مرتضی دلیل
تاریخ: ۱۳۹۳/۰۴/۰۳ ۱۷:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

1. شاید یکی از آزاردهنده‌ترین مشکلات، برخورد با پیغام‌های خطا، هنگام عملیات migration باشد. یکی از ده‌ها نوع خطا، زمانی رخ می‌دهد که متد seed در حال اجراست. در این حالت هیچ نوع break-point ایی به کمک ما نخواهد آمد.

سوال ایجاست که آیا می‌توان این بخش را دیباگ نمود؟ بهترین راه حل، اجرای آپدیت از طریق متدها(یا اکشن ها) است.

فراخوانی migration بسیار ساده است. باید یک نمونه از کلاس Configuration را ساخته و در جایی از پروژه قرار دهیم و صد البته مطمئن باشیم که migration فعال است.

```
var configuration = new Configuration();
var migrator = new DbMigrator(configuration);
migrator.Update();
```

اگر بخواهید این تغییرات بر روی دیتابیسی با اسم و رسم انجام شود، از کد زیر بهره بگیرید:

```
var configuration = new Configuration();
configuration.TargetDatabase = new DbConnectionInfo(
    "Server=MyServer;Database=MyDatabase;Trusted_Connection=True;",
    "System.Data.SqlClient");
var migrator = new DbMigrator(configuration);
migrator.Update();
```

می‌توانید این کد را در ابتدای اکشن index در کنترلر Home قرار دهید و با قرار دادن Break-Point در بخش‌های مختلف متد Seed ، آن را بررسی کنید.

2. خطای :

```
Duplicate type name within an assembly.
```

معمولا بخاطر وجود break-point این مشکل رخ میدهد. یا break-pointهای درون seed را حذف کنید یا جای آنها را تغییر دهید. [اینجا]

3. خطای :

```
Validation failed for one or more entities. See 'EntityValidationErrors' property for more details.
```

این مشکل میتواند دلایل مختلفی داشته باشد. کد درون متد seed را داخلtry/catch قرا ر دهید و علت آن را بررسی کنید:

```
try
                {
                    var user = new ApplicationUser { UserName = "Admin", Phone = "09120000000", Email =
"m@gmail.com" };
                    usermanager.Create(user, "09120000000");
                    usermanager.AddToRole(user.Id, "admin");
                }
                catch (DbEntityValidationException dbEx)
                {
                    foreach (var validationErrors in dbEx.EntityValidationErrors)
                    {
```

```
                        foreach (var validationError in validationErrors.ValidationErrors)
                        {
                            Trace.TraceInformation("Property: {0} Error: {1}",
validationError.PropertyName, validationError.ErrorMessage);
                        }
                    }
                }
```

فراموش نکنید، seed را به کمک حالتی که در شماره 1 گفته شد اجرا کنید تا بتوانید از BreakPoint استفاده کنید.

4.خطای :

```
More than one context type was found in the assembly 'Ex1_CodeFirst'
```

این خطا وقتی ظاهر میشود که چندین Context برای پروژه جاری داشته باشیم و ویژوال استودیو نتواند Context مورد نظر ما را تشخیص دهد. بهتر است هنگام اجرای migration نام context مورد نظر را بنویسیم (مثلا MyConfiguration نام کانتکست شماست) :

```
Update-Database -ConfigurationTypeName MyConfiguration
```

5.خطای :

```
There is already an object named 'UserProfile' in the database.
```

یعنی مدل شما پس از اینکه جدولی با همین نام (در این جا به عنوان مثال UserProfile) از پیش موجود بوده، تغییر کرده است؛ پس migration آپدیت شدنی نیست. ابتدا این دستور را می‌نویسیم (پیش از آنکه تغییراتمان را روی کلاس مربوط به جدول UserProfile اعمال کنیم):

```
Add-Migration Initial –IgnoreChanges
```

اینکار باعث می‌شود یک فایل خالی به نام InitialMigration ایجاد شود. حالا تنظیمات مورد نظر، اعم از تغییر نام یا اضافه کردن ستون خاص را به کلاس UserProfile اعمال کرده و دیتابیس را آپدیت میکنیم :

```
update-database –verbose
```

گاهی این مشکل زمانی پیش می‌آید که واقعا تغییری در جدول نامبرده انجام نداده‌ایم. در این حالت روش پله‌ای زیر را به کار می‌بریم:
پاک کردن یا ریست کردن migration (در شماره 9 همین مقاله این کار در چند مرحله توضیح داده شده است. در مرحله سوم حتما از Add-Migration Initial –IgnoreChanges استفاده کنید)
بعد از آپدیت دیتابیس باید فایل زیر دارای محتویات باشد

محتویات این فایل دقیقا شرایط فعلی جدول شماست.
اگر این فایل ایجاد نشده است مراحل را تکرار کنید تا محتویات درون آن را ببینید.
حالا تغییری را در یکی از مدل‌های خود انجام دهید. احتمالا با مشکل آپدیت شدن مواجه میشوید و پیغام زیر را دوباره خواهید دید:

```
There is already an object named 'UserProfile' in the database
```

جدولی را که پیغام خطا به آن اشاره کرده، در فایل فوق بیابید و محدوده create آن را کامنت کنید تا ساخته نشود.
دیتابیس را آپدیت کنید، احتمالا پیغام خطای فوق برای جدول دیگری نمایش داده می‌شود. آن را هم کامنت کنید و دیتابیس را آپدیت کنید و اگر باز هم خطا بود مکانیزم بالا را تا جایی تکرار کنید که خطایی نبینید .
حالا جدولی را که تغییراتی در آن داده بودید، در دیتابیس چک کنید که تغییرات اعمال شده باشد.
هر آنچه را در فایل initial کامنت کرده بودید، از کامنت خارج کنید و دیتابیس را آپدیت کنید .
برای آزمایش، یک آیتم به یکی از مدل‌ها اضافه کنید و ببینید که migration درست کار می‌کند یا خیر.

6. خطای :

```
Unable to update database to match the current model because there are pending changes
and automatic migration is disabled. Either write the pending model changes to a code-based
migration or enable automatic migration.
Set DbMigrationsConfiguration.AutomaticMigrationsEnabled to true to enable automatic migration.
You can use the Add-Migration command to write the pending model changes to a code-based migration.
```

همانطور که از متن این پیغام پیداست، یعنی شما AutomaticMigration را true نکرده‌اید و اینکار را باید در فایل Configuration.cs در فولدر Migration انجام داد.
7.خطای :

```
Automatic migration was not applied because it would result in data loss.
```

این خط را در سازنده‌ی کلاس Configuration اضافه میکنیم :

```
AutomaticMigrationDataLossAllowed = true;
```

8.خطای :

```
Introducing FOREIGN KEY constraint 'FK_dbo.ProductProductGroups_dbo.ProductGroups_ProductGroupId' on
table
'ProductProductGroups' may cause cycles or multiple cascade paths.
Specify ON DELETE NO ACTION or ON UPDATE NO ACTION, or modify other FOREIGN KEY constraints.
```

```
Could not create constraint or index. See previous errors.
```

خطای فوق وقتی رخ میدهد که دو جدول از یک طرف به هم وصل باشند و از طرف دیگر مشخصات ذکر نشده باشد.
9. راه حل برای خطاهای عجیبی که شاید نیاز به صرف زمان بیشتر برای کشف و برطرف کردن داشته باشند :
بهتر نیست مایگریشن خود را از نو بسازید؟
روش به روز رسانی و بازسازی migration [اینجا]:
- پاک کردن فولدر Migrations در پروژه - پاک کردن جدولی به نام _MigrationHistory در دیتابیس (ممکن است زیر مجموعه جدول‌های system باشد) - اجرای دستور زیر کنسول پکیج منیجر ویژوال استودیو :

```
Enable-Migrations -EnableAutomaticMigrations -Force
```

- اجرای دستور زیر :

```
Add-Migration Initial
```

نظرات خوانندگان

نویسنده: شایان
تاریخ: ۱۳۹۳/۰۵/۲۱ ۱۳:۰

سلام موقع اجرای دستور update-database به من این خطا را میده :

```
Could not load file or assembly 'Microsoft.SqlServer.BatchParser, Version=11.0.0.0, Culture=neutral, PublicKeyToken=89845dcd8080cc91' or one of its dependencies. The system cannot find the file specified
```

شما برخورد نداشتین باهاش ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۵/۲۱ ۱۳:۳۴

چنین DLL ایی (SMO - Shared Management Objects) در هیچ کجای سورس‌های EF استفاده نشده. احتمالا یک کتابخانه‌ی ثالث در برنامه‌ی شما این مشکل رو درست کرده. اگر ارجاعی در برنامه به آن دارید حذفش کنید. خصوصا فایل‌های کانفیگ رو هم بررسی کنید.

در کل این DLL رو از اینجا می‌تونید دریافت کنید: http://www.microsoft.com/en-us/download/details.aspx?id=35580

نویسنده: محمد صاحب
تاریخ: ۱۳۹۳/۰۶/۱۹ ۱۵:۵

با اجازه صاحب مطلب

10.خطای :

```
User canceled out of save dialog
```

وقتی میخواهیم اسکریپت برای ما ساخته بشه

```
update-database -verbose -script
```

در واقع مشکل از SQL Server Data Tools هست و با آپدیت کردن مشکل حل میشه.

عنوان: **بازسازی جدول MigrationHistory با کد نویسی در EF Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۳ ۸:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

فرض کنید با استفاده از ابزار EF Power tools معادل Code first یک بانک اطلاعاتی موجود را تهیه کرده‌اید. اکنون برای استفاده از آن با گردش کاری متداول EF Code first نیاز است تا جدولی را به نام MigrationHistory نیز به این بانک اطلاعاتی اضافه کنیم. از این جدول برای نگهداری سوابق به روز رسانی ساختار بانک اطلاعاتی بر اساس مدل‌های برنامه و سپس مقایسه آن‌ها استفاده می‌شود. یا حتی ممکن است به اشتباه در حین کار با بانک اطلاعاتی این جدول حذف شده باشد. روش باز تولید آن توسط دستورهای پاور شل به سادگی اجرای سه دستور ذیل است:

```
enable-migrations
add-migration Initial -IgnoreChanges
update-database
```

IgnoreChanges سبب می‌شود تا EF فرض کند، تطابق یک به یکی بین مدل‌های برنامه و ساختار جداول بانک اطلاعاتی وجود دارد. سپس بر این اساس، جدول MigrationHistory جدیدی را آغاز می‌کند.

سؤال: چگونه می‌توان همین عملیات را با کدنویسی انجام داد؟

متد UpdateDatabase کلاس ذیل، دقیقا معادل است با اجرای سه دستور فوق :

```
using System.Data.Entity.Migrations;
using System.Data.Entity.Migrations.Design;

namespace EFTests
{
    /// <summary>
    /// Using Entity Framework Code First with an existing database.
    /// </summary>
    public static class CreateMigrationHistory
    {
        /// <summary>
        /// Creates a new '__MigrationHistory' table.
        /// Enables migrations using Entity Framework Code First on an existing database.
        /// </summary>
        public static void UpdateDatabase(DbMigrationsConfiguration configuration)
        {
            var scaffolder = new MigrationScaffolder(configuration);
            // Creates an empty migration, so that the future migrations will start from the current
state of your database.
            var scaffoldedMigration = scaffolder.Scaffold("IgnoreChanges", ignoreChanges: true);

            // enable-migrations
            // add-migration Initial -IgnoreChanges
            configuration.MigrationsAssembly = new
MigrationCompiler(ProgrammingLanguage.CSharp.ToString())
                                                .Compile(configuration.MigrationsNamespace,
scaffoldedMigration);

            // update-database
            var dbMigrator = new DbMigrator(configuration);
            dbMigrator.Update();
        }
    }
}
```

توضیحات

MigrationScaffolder کار تولید خودکار کلاس‌های cs مهاجرت‌های EF را انجام می‌دهد. زمانیکه به متد Scaffold آن پارامتر ignoreChanges: true ارسال شود، کلاس مهاجرتی را ایجاد می‌کند که خالی است (متدهای up و down آن خالی تشکیل می‌شوند). سپس این کلاس‌ها کامپایل شده و در حین اجرای برنامه مورد استفاده قرار می‌گیرند.

برای استفاده از آن، نیاز به کلاس MigrationCompiler خواهید داشت. این کلاس در مجموعه آزمون‌های واحد EF به عنوان یک

کلاس کمکی وجود دارد: MigrationCompiler.cs

صرفا جهت تکمیل بحث و همچنین سهولت ارجاعات آتی، کدهای آن در ذیل نیز ذکر خواهد شد:

```
using System;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Data.Entity.Migrations.Design;
using System.Data.Entity.Spatial;
using System.IO;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using System.Resources;
using System.Text;

namespace EF_General.Models.Ex22
{
    public enum ProgrammingLanguage
    {
        CSharp,
        VB
    }

    public class MigrationCompiler
    {
        private readonly CodeDomProvider _codeProvider;

        public MigrationCompiler(string language)
        {
            _codeProvider = CodeDomProvider.CreateProvider(language);
        }

        public Assembly Compile(string @namespace, params ScaffoldedMigration[] scaffoldedMigrations)
        {
            var options = new CompilerParameters
            {
                GenerateExecutable = false,
                GenerateInMemory = true
            };

            options.ReferencedAssemblies.Add(typeof(string).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(Expression).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbMigrator).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbContext).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbConnection).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(Component).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(MigrationCompiler).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbGeography).Assembly.Location);

            var embededResources = GenerateEmbeddedResources(scaffoldedMigrations, @namespace);
            foreach (var resource in embededResources)
                options.EmbeddedResources.Add(resource);

            var sources = scaffoldedMigrations.SelectMany(g => new[] { g.UserCode, g.DesignerCode });

            var compilerResults = _codeProvider.CompileAssemblyFromSource(options, sources.ToArray());
            foreach (var resource in embededResources)
                File.Delete(resource);

            if (compilerResults.Errors.Count > 0)
            {
                throw new InvalidOperationException(BuildCompileErrorMessage(compilerResults.Errors));
            }

            return compilerResults.CompiledAssembly;
        }

        private static string BuildCompileErrorMessage(CompilerErrorCollection errors)
        {
            var stringBuilder = new StringBuilder();

            foreach (CompilerError error in errors)
            {
                stringBuilder.AppendLine(error.ToString());
            }
```

```
            return stringBuilder.ToString();
        }

        private static IEnumerable<string> GenerateEmbeddedResources(IEnumerable<ScaffoldedMigration>
scaffoldedMigrations, string @namespace)
        {
            foreach (var scaffoldedMigration in scaffoldedMigrations)
            {
                var className = GetClassName(scaffoldedMigration.MigrationId);
                var embededResource = Path.Combine(
                    Path.GetTempPath(),
                    @namespace + "." + className + ".resources");

                using (var writer = new ResourceWriter(embededResource))
                {
                    foreach (var resource in scaffoldedMigration.Resources)
                        writer.AddResource(resource.Key, resource.Value);
                }

                yield return embededResource;
            }
        }

        private static string GetClassName(string migrationId)
        {
            return migrationId
                .Split(new[] { '_' }, 2)
                .Last()
                .Replace(" ", string.Empty);
        }
    }
}
```

جهت مطالعه توضیحات بیشتری در مورد CodeDom می‌توان به مطلب « کامپایل پویای کد در دات نت » مراجعه کرد.

استفاده از این کلاس‌ها نیز بسیار ساده است. یکبار دستور ذیل را در ابتدای کار برنامه فراخوانی کنید تا جدول MigrationHistory دوباره ساخته شود:

```
CreateMigrationHistory.UpdateDatabase(new Configuration());
```

با این فرض که کلاس Configuration شما چنین شکلی را دارد:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
    }
```

نظرات خوانندگان

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۸/۱۴ ۱۶:۵۸

با تشکر از اینکه به این مطلب مستقلا پرداختید
فقط قسمتهای زیر شناخته نمی‌شد آیا به اسمبلی خاصی نیاز دارد؟

```
using System.Data.Entity.Spatial;
 options.ReferencedAssemblies.Add(typeof(DbGeography).Assembly.Location);
scaffoldedMigration.Resources
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۴ ۱۷:۱۹

DbGeography از EF **5** به بعد اضافه شده. اگر پروژه EF 4 دارید، راحت قابل ارتقاء به نگارش 6 هست. البته EF نگارش 5 مخصوص دات نت 4 این قابلیت رو نداشت (فقط نگارش مخصوص دات نت 4.5 آن شامل این پیشرفت‌ها می‌شد). اما EF 6 مخصوص دات نت 4 هم این فضاهای نام رو داره و این محدودیت‌ها برطرف شده.

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۴:۴۹

در خط

```
var scaffolder = new MigrationScaffolder(configuration);
```

با خطای زیر مواجه شد:

The Entity Framework provider type 'System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' for the 'System.Data.SqlClient' ADO.NET provider could not be loaded. Make sure the provider assembly is available to the running application. See http://go.microsoft.com/fwlink/?LinkId=260882 for more information.

ضمنا آیا اسمبلی مربوط به EF6 را نمی‌توان بجز نیوگت از روش دیگری بدست آورد.؟
همچنین EF Power Tools مربوط به EF6 را از کجا می‌توان تهیه کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۵:۱۴

- مطلب جاری فقط برای حالتی است که جدول migration حذف شده یا وجود ندارد.
+ مطلب « ارتقاء به Entity framework 6 و استفاده از بانک‌های اطلاعاتی غیر از SQL Server » را باید دقیق مطالعه کنید. یک سری اسمبلی باید حذف شوند. تعدادی اضافه شوند. فایل کانفیگ حتما باید ویرایش شود و تعریف پروایدر را داشته باشد. این کارها را نیوگت به صورت خودکار انجام می‌دهد. ضمنا اینکار باید برای تمام زیر پروژه‌های شما نیز تکرار شود و طوری نباشد که دو کتابخانه از 4 استفاده کنند، دوتای دیگر از 5 و اصلی هم از 6. همه باید یک دست شوند و اسمبلی منسوخ شده قدیمی نیز حذف.
- اسمبلی EF به تنهایی کافی نیست ولی از اینجا به صورت جداگانه قابل دریافت است. باید دقت داشت که ارتقاء به نگارش 6 سه مرحله‌ای است که عنوان شد.
- از اینجا

نویسنده: reza110

تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۰:۳۸

با تشکر فراوان از زحمات شما
با دانلود پکیجی که اشاره کردید و افزودن اسمبلی‌های مربوطه (Entityframework .sqlserver, Entityframework6,) و اصلاح آدرس فضای نام در قسمتهای مختلف برنامه قابلیت ایجاد جدول مهاجرت ایجاد گردید.
سوالی که داشتم این بود که در سایت نیوگت پکیج هایی که وجود دارد را چگونه می‌توان مستقیما بدون استفاده از vs دریافت کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۰:۵۰

اینجا توضیح دادم

عنوان: **استفاده از چندین Context در EF 6 Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۳ ۱۴:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در نگارش قبلی EF Code first به ازای یک پروژه تنها یک سیستم Migration قابل تعریف بود و این سیستم مهاجرت، تنها با یک DbContext کار می‌کرد. در نگارش ششم این کتابخانه، سیستم مهاجرت Code first آن از چندین DbContext، به ازای یک پروژه که به یک بانک اطلاعاتی اشاره می‌کنند، پشتیبانی می‌کند. مزیت اینکار اندکی بهبود در نگهداری تنها کلاس DbContext تعریف شده است. برای مثال می‌توان یک کلاس DbContext مخصوص قسمت ثبت نام را ایجاد کرد. یک کلاس DbContext مخصوص کلیه جداول مرتبط با مقالات را و همینطور الی آخر. نهایتا تمام این Contextها سبب ایجاد یک بانک اطلاعاتی واحد خواهند شد.
اگر در یک پروژه EF Code first چندین Context وجود داشته باشد و دستور enable-migrations را بدون پارامتری فراخوانی کنیم، پیغام خطای More than one context type was found in the assmbly xyz را دریافت خواهیم کرد.
الف) اما در EF 6 می‌توان با بکار بردن سوئیچ جدید ContextTypeName، به ازای هر Context، مهاجرت مرتبط با آن‌را تنظیم نمود:

```
enable-migrations -ContextTypeName dbContextName1 -MigrationDirectory DataContexts\Name1Migrations
```

همچنین در اینجا نیز می‌توان با استفاده از سوئیچ MigrationDirectory، فایل‌های تولید شده را در پوشه‌های مجزایی ذخیره کرد.

ب) در مرحله بعد، نیاز به فراخوانی دستور add-migration است:

```
add-migration -ConfigurationTypeName FullNameSpaceCtx1.Configuration "InitialCreate"
```

با اجرای دستور enable-migrations یک کلاس Configuration جهت DbContext مشخص شده، ایجاد می‌شود. سپس آدرس کامل این کلاس را به همراه ذکر دقیق فضای نام آن در اختیار دستور add-migration قرار می‌دهیم.
ذکر کامل فضای نام، از این جهت مهم است که کلاس Configuration به ازای Contextهای مختلف ایجاد شده، یک نام را خواهد داشت؛ اما در فضاهای نام متفاوتی قرار می‌گیرد.
با اجرای دستور add-migration، کدهای سی شارپ مورد نیاز جهت اعمال تغییرات بر روی ساختار بانک اطلاعاتی تولید می‌شوند. در مرحله بعد، این کدها تبدیل به دستورات SQL متناظری شده و بر روی بانک اطلاعاتی اجرا خواهند شد.
بدیهی است اگر دو Context در برنامه تعریف کرده باشید، دوبار باید دستور enable-migrations و دوبار دستور add-migration را با پارامترهای اشاره کننده به Conetxtهای مدنظر اجرا کرد.

ج) سپس برای اعمال این تغییرات، باید دستور update-database را اجرا کرد.

```
update-database  -ConfigurationTypeName FullNameSpaceCtx1.Configuration
```

اینبار دستور update-database نیز بر اساس نام کامل کلاس Configuration مدنظر باید اجرا گردد و به ازای هر Context موجود، یکبار نیاز است اجرا گردد.
نهایتا اگر به بانک اطلاعاتی مراجعه کنید، تمام جداول و تعاریف را یکجا در همان بانک اطلاعاتی می‌توانید مشاهده نمائید.

داشتن چندین Context در برنامه و مدیریت تراکنش‌ها

در EF، هر DbContext معرف یک واحد کار است. یعنی تراکنش‌ها و چندین عمل متوالی مرتبط انجام شده، درون یک DbContext معنا پیدا می‌کنند. متد SaveChanges نیز بر همین اساس است که کلیه اعمال ردیابی شده در طی یک واحد کار را در طی یک تراکنش به بانک اطلاعاتی اعمال می‌کند. همچنین مباحثی مانند lazy loading نیز در طی یک Context مفهوم دارند. به علاوه دیگر امکان join نویسی بین دو Context وجود نخواهد داشت. باید اطلاعات را از یکی واکشی و سپس این اطلاعات درون حافظه‌ای را به دیگری ارسال کنید.

یک نکته

می‌توان یک DbSet را در چندین Context تعریف کرد. یعنی اگر بحث join نویسی مطرح است، با تکرار تعریف DbSetها اینکار قابل انجام است اما این مساله اساس جداسازی Contextها را نیز زیر سؤال می‌برد.

داشتن چندین Context در برنامه و مدیریت رشته‌های اتصالی

در EF Code first روش‌های مختلفی برای تعریف رشته اتصالی به بانک اطلاعاتی وجود دارند. اگر تغییر خاصی در کلاس مشتق شده از DbContext ایجاد نکنیم، نام کلید رشته اتصالی تعریف شده در فایل کانفیگ باید به نام کامل کلاس Context برنامه اشاره کند. اما با داشتن چندین Context به ازای یک دیتابیس می‌توان از روش ذیل استفاده کرد:

```
public class Ctx1 : DbContext
    {
        public Ctx1()
            : base("DefaultConnection")
        {
            //Database.Log = sql => Debug.Write(sql);
        }
    }

  public class Ctx2 : DbContext
    {
        public Ctx2()
            : base("DefaultConnection")
        {
            //Database.Log = sql => Debug.Write(sql);
        }
    }
```

```
<connectionStrings>
    <add name="DefaultConnection" connectionString="…." providerName="System.Data.SqlClient" />
  </connectionStrings>
```

در اینجا در سازنده کلاس‌های Context تعریف شده، نام کلید رشته اتصالی موجود در فایل کانفیگ برنامه ذکر شده است. به این ترتیب این دو Context به یک بانک اطلاعاتی اشاره خواهند کرد.

چه زمانی بهتر است از چندین Context در برنامه استفاده کرد؟

عموما در طراحی‌های سازمانی SQL Server، تمام جداول از schema مدیریتی به نام dbo استفاده نمی‌کنند. جداول فروش از schema خاص خود و جداول کاربران از schema دیگری استفاده خواهند کرد. با استفاده از چندین Context می‌توان به ازای هر کدام از schemaهای متفاوت موجود، «یک ناحیه ایزوله» را ایجاد و مدیریت کرد.

```
public class Ctx2 : DbContext
    {
        public Ctx2()
            : base("DefaultConnection")
        {
            //Database.Log = sql => Debug.Write(sql);
        }

         protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.HasDefaultSchema("sales");
            base.OnModelCreating(modelBuilder);
        }
    }
```

در این حالت در EF 6 می‌توان DefaultSchema کلی یک Context را در متد بازنویسی شده OnModelCreating به نحو فوق تعریف و مدیریت کرد. در این مثال به صورت خودکار کلیه DbSetهای Ctx2 از schema ایی به نام sales استفاده می‌کنند.

نظرات خوانندگان

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۹/۰۳ ۲۲:۴۹

می خواستم ببینم می‌توان از این امکان برای پیاده سازی دیتابیس‌های توزیع شده استفاده کرد؟
مثلا جدول کاربران در یک پایگاه داده‌ی مستقل و جدول دیدگاه‌های کاربران در یک پایگاه داده مستقل دیگر باشد و به ازای هر کدام یک Context جداگانه تعریف کرد و در برنامه با آن‌ها تعامل کرد به گونه ای که به نظر آید با یک پایگاه داده سر و کار داریم.
آیا اگر از این شیوه‌ی طراحی استفاده شود دیگر مسائل رابطه‌ی بین جداول منتفی است؟ و اگر بله شبیه سازی رابطه‌ها باید به این صورت پیاده سازی شود که اطلاعات جداول به صورت جداگانه از دیتابیس‌ها خوانده شود و سپس با استفاده از Linq To Object رابطه‌ی بین آن‌ها برقرار شود؟
با این شیوه‌ی طراحی تراکنش‌ها چگونه پیاده سازی می‌شود؟ آیا هر Context دارای یک تراکنش جداگانه است و یا امکان پیاده سازی آن به صورت یک تراکنش هم وجود دارد؟ الگوی Unit Of Work را باید به ازای هر Context جداگانه تعریف کرد؟
البته من اطلاعات خیلی ناقصی از پایگاه‌های داده توزیع شده دارم و ممکنه حرفام کاملا اشتباه باشد. متاسفانه من جایی را پیدا نکردم که در مورد پیاده سازی عملی آن بحث کرده باشد و بیشتر با یک سری مفاهیم تئوری برخورد کردم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱:۲۳

- در SQL Server قابلیتی به نام Linked servers وجود دارد که توسط آن در یک شبکه داخلی می‌شود چندین بانک اطلاعاتی SQL Server را در نودهای مختلف شبکه به هم متصل کرد و با آن‌ها یکپارچه و مانند یک دیتابیس واحد کار کرد. این مورد در برنامه‌های سازمانی خیلی مرسوم است. قرار نیست به ازای هر برنامه‌ای که برای یک سازمان نوشته می‌شود، هر کدام جداگانه بانک اطلاعاتی کاربران و لاگین خاص خودشان را داشته باشند. عموما یک دیتابیس مرکزی مثلا مدیریت منابع انسانی وجود دارد که مابقی برنامه‌ها را تغذیه می‌کند. حتی می‌شود به یک بانک اطلاعاتی MySQL هم متصل شد (^).
- ORMها صرفا کارهایی را قادر به انجام هستند که بانک اطلاعاتی مورد استفاده پشتیبانی می‌کند. برای نمونه در SQL Server امکان تهیه رابطه بین دو جدول از دو بانک اطلاعاتی مختلف وجود ندارد . منظور مثلا ایجاد کلید خارجی بین جداول دو بانک اطلاعاتی مختلف است و نه نوشتن کوئری بین آن‌ها که از این لحاظ مشکلی نیست.
A FOREIGN KEY constraint can reference columns in tables in the same database or within the same table
- هدف از پشتیبانی چند Context در EF 6، تلفیق این‌ها با هم در قالب یک بانک اطلاعاتی است (مطلب فوق).
- همچنین در EF 6 می‌توان چندین Context را به چندین بانک اطلاعاتی مختلف با رشته‌های اتصالی متفاوت، انتساب داد. مباحث آغاز بانک‌های اطلاعاتی هر کدام جداگانه عمل کرده و مستقل از هم عمل می‌کنند.
یک مثال جهت اجرا و آزمایش:
Sample25.cs
- اگر می‌خواهید به انعطاف پذیری Linked servers برسید (مثلا در طی یک کوئری و نه چند کوئری از جداول دو بانک اطلاعاتی مختلف در دو سرور متفاوت کوئری بگیرید)، نیاز خواهید داشت مثلا View یا SP تهیه کنید و سپس این‌ها را در برنامه مورد استفاده قرار دهید. Viewها با استفاده از T-SQL می‌توانند کوئری واحدی از چند بانک اطلاعاتی تهیه کنند. اینبار برنامه هم نهایتا به یک بانک اطلاعاتی متصل می‌شود و برای آن اهمیتی ندارد که View یا SP به چه نحوی تهیه شده و با چند بانک اطلاعاتی کار می‌کند.
- تراکنش‌های توزیع شده هم نیاز به تنظیمات خاصی در SQL Server دارند و باید MSDTC راه اندازی و تنظیم شود: (^). در غیراینصورت پیام خطای The underlying provider failed on Open. MSDTC on server is unavailable را دریافت خواهید کرد.
بعد از آن باید از TransactionScope برای کار همزمان با چند Context استفاده کنید.

نویسنده: محمد دبیرسیاقی
تاریخ: ۱۳۹۲/۰۹/۱۳ ۲۱:۱۷

من از دو context در برنامم برای schemaهای مختلف استفاده میکنم مشکلی که دارم اینه که entity یک schema وابستگی به entity شمای دیگر دارد و من entityهای هر schema را در یک assembly قرار دادم .الان نمیدانم چطور روابط را برقرار کنم مثلا در یک schema جدول کاربران را دارم که در یک Assembly است و در شمای دیگر جدول مقالات را در assembly دیگری دارم

.ارتباط این دو به این صورت است که وقتی مقاله ای درج می‌شود باید کاربری که مقاله را درج کرده نیز در دیتابیس درج شود.
الان باید یک پروپرتی از جنس مقاله در کلاس کاربر و یک پروپرتی از جنس کاربر در کلاس مقاله در نظر بگیرم.با وجود اینکه هر کدام در یک Assembly هستند باید Refrence از یک اسمبلی در دیگری داشته باشم که این موضوع امکان پذیر نیست لطفا راهنمایی نمائید ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۳ ۲۳:۵۴

- عموما circular reference بین اسمبلی‌ها نشانه‌ی طراحی بد است.
- استفاده از چند Context برای اینکه هر کدام قرار است در یک دیتابیس جدا ذخیره شوند؟ نمی‌شود FK بین این‌ها (جداول دو دیتابیس مختلف) تعریف کرد (SQL Server چنین کاری را پشتیبانی نمی‌کند).
- اگر برنامه ماژولار است، در EF می‌توان به صورت خودکار تمام ماژول‌ها را در طی یک Context یکپارچه بارگذاری کرد (^ و ^).
- هدف از ایجاد Schema در SQL Server، ایجاد ظروفی برای گروه بندی منطقی اشیاء است. مثلا عده‌ای به سه SP خاص دسترسی داشته باشند. عده‌ای فقط بتوانند با Viewها کار کنند. یا حتی عده‌ای به تمام موارد دسترسی داشته باشند. بنابراین یک نوع ایزوله سازی قسمت‌های مختلف بانک اطلاعاتی مدنظر هست، در اصل. حالا اگر عده‌ای فقط به سه جدول خاص دسترسی دارند، آیا می‌توانند ارجاعی را به جدول چهارمی که در یک schema دیگر تعریف شده داشته باشند؟ بله. البته فقط به این شرط که کاربران schema سه جدول فعلی به schema جدول چهارم، دسترسی و مجوز لازم را داشته باشد و برای این دسترسی دادن‌ها هم باید مستقلا T-SQL بنویسید.
و ... ضمنا گاهی از اوقات از Schema برای مدیریت نام‌های هم نام استفاده می‌شود. چیزی شبیه به namespace در سی‌شارپ مثلا. نمونه‌اش طراحی چند مستاجری است.
نتیجه گیری؟ برای سرگرمی Schema ایجاد نکنید؛ مگر اینکه واقعا قصد ایزوله سازی قسمت‌های مختلف یک بانک اطلاعاتی را از کاربرانی خاص داشته باشید. به Schema به شکل یک Sandbox امنیتی (یک قرنطینه) نگاه کنید.

برای مطالعه بیشتر
Understanding the Difference between Owners and Schemas in SQL Server
Implementation of Database Object Schemas

نویسنده: میثم فغفوری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۰:۱۹

خسته نباشید. سناریوی بنده این است که میخوام سایتی طراحی کنم که کاملا ماژولار باشد یعنی هر بخش رو به صورت user controller طراحی و در هسته اصلی لود کنم و هر کدوم از این کنترلرها جداول مخصوص به خودشون رو دارن توی بانک اطلاعاتی که خوب طبیعتا کلاس POCO و DBContext مخصوص هر ماژول باید توی سورس کد خود ماژول نوشته بشه. با فرض اینکه بعد از کامپایل پروژه دیگه دسترسی به migration نداریم میتونین بنده رو راهنمایی کنین که چطور میتونم با این روشی که فرمودید جداول جداگانه هر کنترلر یا ماژول رو به بانک اضافه کنم بدون دسترسی به migration ؟ خودم هر راهی به ذهنم رسید انجام دادم ولی همچنان ارور تغییر در کلاس‌های POCO را میدهد سایت.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۰:۴۷

از یک Context مرکزی استفاده کنید که موجودیت‌ها را به صورت خودکار خوانده و اضافه می‌کند . همچنین فعال سازی گزینه‌های مهاجرت خودکار را هم مطالعه کنید.

نویسنده: میثم فغفوری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۵:۳

ممنون مشکلم حل شد فقط برای عملیات Seed برای جداول هر ماژول هم راهی هست؟ با استفاده از همون Reflection.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۶:۱۵

ایده کار، این بود که قسمتی را مثلا توسط یک کلاس پایه، یا یک اینترفیس خالی علامتگذاری کنید و سپس اطلاعات آن‌را تک تک به متد Entity مربوط به DbModelBuilder ارسال کنید. همین ایده را در متد Seed هم می‌شود پیاده سازی کرد. یک اینترفیس خالی را مثلا به نام IِMySeed تعریف کنید و به کلاس دلخواهی انتساب دهید (یا از MEF استفاده کنید). سپس اینطرف با Reflection این نوع کلاس‌ها را بارگذاری کرده و Context متد Seed را به طراحی انجام شده، برای عملیات نهایی و دلخواه ارسال کنید.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۳/۰۳/۰۳ ۱۹:۲۵

با سلام
من از EF 5 dbfirst به صورت Context های جداگانه در پروژه‌های وب جدا استفاده کردم و در نهایت تمامی این assembly ها را در یک وب سایت publish می‌کنم . در صورتی که از یک Entity به صورت مشترک در context 2 استفاده کرده باشم با خطای زیر

```
System.Data.MetadataException: Schema specified is not valid. Errors:
 Multiple types with the name &#39;Customer&#39; exist in the EdmItemCollection in different
namespaces .  Convention based mapping requires unique names without regard to namespace in the
EdmItemCollection
```

مواجه میشم . با اینکه Assembly های مربوط به Contextها متفاوت هست اما با این خطا روبرو میشم . آیا قابلیت گفته شده در EF 6 این مشکل برطرف شده است ؟ و یا در ef 5 راهکاری برای این مشکل وجود ندارد ؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۰۳ ۲۰:۱۴

جستجوی عین خطا در گوگل، پاسخ اول .

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۳/۰۳/۲۶ ۱۲:۷

در خط

```
enable-migrations -ContextTypeName dbContextName1 -MigrationDirectory DataContexts\Name1Migrations
```

فکر میکنم MigrationDirectory باید به MigrationsDirectory تغییر کند.

عنوان: EF Code First #2
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۵ ۰۹:۳۶:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در قسمت قبل با تنظیمات و قراردادهای ابتدایی EF Code first آشنا شدیم، هرچند این تنظیمات حجم کدنویسی ابتدایی راه اندازی سیستم را به شدت کاهش می‌دهند، اما کافی نیستند. در این قسمت نگاهی سطحی و مقدماتی خواهیم داشت بر امکانات مهیا جهت تنظیم ویژگی‌های مدل‌های برنامه در EF Code first.

تنظیمات EF Code first توسط اعمال متادیتای خواص

اغلب متادیتای مورد نیاز جهت اعمال تنظیمات EF Code first در اسمبلی System.ComponentModel.DataAnnotations.dll قرار دارند. بنابراین اگر مدل‌های خود را در اسمبلی و پروژه class library جداگانه‌ای تعریف و نگهداری می‌کنید (مثلا به نام DomainClasses)، نیاز است ابتدا ارجاعی را به این اسمبلی به پروژه جاری اضافه نمائیم. همچنین تعدادی دیگر از متادیتای قابل استفاده در خود اسمبلی EntityFramework.dll قرار دارند. بنابراین در صورت نیاز باید ارجاعی را به این اسمبلی نیز اضافه نمود.
همان مثال قبل را در اینجا ادامه می‌دهیم. دو کلاس Blog و Post در آن تعریف شده (به این نوع کلاس‌ها POCO – the Plain Old CLR Objects نیز گفته می‌شود)، به همراه کلاس Context که از کلاس DbContext مشتق شده است. ابتدا دیتابیس قبلی را دستی drop کنید. سپس در کلاس Blog، خاصیت public int Id را مثلا به public int MyTableKey تغییر دهید و پروژه را اجرا کنید.
برنامه بلافاصله با خطای زیر متوقف می‌شود:

```
One or more validation errors were detected during model generation:
\tSystem.Data.Entity.Edm.EdmEntityType: : EntityType 'Blog' has no key defined.
```

زیرا EF Code first در این کلاس خاصیتی به نام Id یا BlogId را نیافته‌است و امکان تشکیل Primary key جدول را ندارد. برای رفع این مشکل تنها کافی است ویژگی Key را به این خاصیت اعمال کنیم:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace EF_Sample01.Models
{
    public class Blog
    {
        [Key]
        public int MyTableKey { set; get; }
```

همچنین تعدادی ویژگی دیگر مانند MaxLength و Required را نیز می‌توان بر روی خواص کلاس اعمال کرد:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace EF_Sample01.Models
{
    public class Blog
    {
        [Key]
        public int MyTableKey { set; get; }

        [MaxLength(100)]
```

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

        [Required]
        public string AuthorName { set; get; }

        public IList<Post> Posts { set; get; }
    }
}
```

این ویژگی‌ها دو مقصود مهم را برآورده می‌سازند:
الف) بر روی ساختار بانک اطلاعاتی تشکیل شده تاثیر دارند:

```
CREATE TABLE [dbo].[Blogs](
 [MyTableKey] [int] IDENTITY(1,1) NOT NULL,
 [Title] [nvarchar](100) NULL,
 [AuthorName] [nvarchar](max) NOT NULL,
 CONSTRAINT [PK_Blogs] PRIMARY KEY CLUSTERED
(
 [MyTableKey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

همانطور که ملاحظه می‌کنید در اینجا طول فیلد Title به 100 تنظیم شده است و همچنین فیلد AuthorName اینبار NOT NULL است. به علاوه primary key نیز بر اساس ویژگی Key اعمالی تعیین شده است.
البته برای اجرای کدهای تغییر کرده مدل، فعلا بانک اطلاعاتی قبلی را دستی می‌توان حذف کرد تا بتوان به ساختار جدید رسید. در مورد جزئیات مبحث DB Migration در قسمت‌های بعدی مفصلا بحث خواهد شد.

ب) اعتبار سنجی اطلاعات پیش از ارسال کوئری به بانک اطلاعاتی
برای مثال اگر در حین تعریف وهله‌ای از کلاس Blog، خاصیت AuthorName مقدار دهی نگردد، پیش از اینکه رفت و برگشتی به بانک اطلاعاتی صورت گیرد، یک validation error را دریافت خواهیم کرد. یا برای مثال اگر طول اطلاعات خاصیت Title بیش از 100 حرف باشد نیز مجددا در حین ثبت اطلاعات، یک استثنای اعتبار سنجی را مشاهده خواهیم کرد. البته امکان تعریف پیغام‌های خطای سفارشی نیز وجود دارد. برای این حالت تنها کافی است پارامتر ErrorMessage این ویژگی‌ها را مقدار دهی کرد. برای مثال:

```
[Required(ErrorMessage = "لطفا نام نویسنده را مشخص نمائید")]
public string AuthorName { set; get; }
```

نکته‌ی مهمی که در اینجا وجود دارد، وجود یک اکوسیستم هماهنگ و سازگار است. این نوع اعتبار سنجی هم با EF Code first هماهنگ است و هم برای مثال در ASP.NET MVC به صورت خودکار جهت اعتبار سنجی سمت سرور و کلاینت یک مدل می‌تواند مورد استفاده قرار گیرد و مفاهیم و روش‌های مورد استفاده در آن نیز یکی است.

تنظیمات EF Code first به کمک Fluent API

اگر علاقمند به استفاده از متادیتا، جهت تعریف قیود و ویژگی‌های خواص کلاس‌های مدل خود نیستید، روش دیگری نیز در EF Code first به نام Fluent API تدارک دیده شده است. در اینجا امکان تعریف همان ویژگی‌ها توسط کدنویسی نیز وجود دارد، به علاوه اعمال قیود دیگری که توسط متادیتای مهیا قابل تعریف نیستند.
محل تعریف این قیود، کلاس Context که از کلاس DbContext مشتق شده است، می‌باشد و در اینجا، کار با تحریف متد OnModelCreating شروع می‌شود:

```
using System.Data.Entity;
using EF_Sample01.Models;

namespace EF_Sample01
{
    public class Context : DbContext
    {
        public DbSet<Blog> Blogs { set; get; }
        public DbSet<Post> Posts { set; get; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Blog>().HasKey(x => x.MyTableKey);
            modelBuilder.Entity<Blog>().Property(x => x.Title).HasMaxLength(100);
            modelBuilder.Entity<Blog>().Property(x => x.AuthorName).IsRequired();

            base.OnModelCreating(modelBuilder);
        }
    }
}
```

به کمک پارامتر modelBuilder، امکان دسترسی به متدهای تنظیم کننده ویژگی‌های خواص یک مدل یا موجودیت وجود دارد. در اینجا چون می‌توان متدها را به صورت یک زنجیره به هم متصل کرد و همچنین حاصل نهایی شبیه به جمله بندی انگلیسی است، به آن Fluent API یا API روان نیز گفته می‌شود.

البته در این حالت امکان تعریف ErrorMessage وجود ندارد و برای این منظور باید از همان data annotations استفاده کرد.

نحوه مدیریت صحیح تعاریف نگاشت‌ها به کمک Fluent API

OnModelCreating محل مناسبی جهت تعریف حجم انبوهی از تنظیمات کلاس‌های مختلف مدل‌های برنامه نیست. در حد سه چهار سطر مشکلی ندارد اما اگر بیشتر شد بهتر است از روش زیر استفاده شود:

```
using System.Data.Entity;
using EF_Sample01.Models;
using System.Data.Entity.ModelConfiguration;

namespace EF_Sample01
{
    public class BlogConfig : EntityTypeConfiguration<Blog>
    {
        public BlogConfig()
        {
            this.Property(x => x.Id).HasColumnName("MyTableKey");
            this.Property(x => x.RowVersion).HasColumnType("Timestamp");
        }
    }
```

با ارث بری از کلاس EntityTypeConfiguration، می‌توان به ازای هر کلاس مدل، تنظیمات را جداگانه انجام داد. به این ترتیب اصل SRP یا Single responsibility principle نقض نخواهد شد. سپس برای استفاده از این کلاس‌های Config تک مسئولیتی به نحو زیر می‌توان اقدام کرد:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
     modelBuilder.Configurations.Add(new BlogConfig());
```

نحوه تنظیمات ابتدایی نگاشت کلاس‌ها به بانک اطلاعاتی در EF Code first

الزامی ندارد که EF Code first حتما با یک بانک اطلاعاتی از نو تهیه شده بر اساس پیش فرض‌های آن کار کند. در اینجا می‌توان از بانک‌های اطلاعاتی موجود نیز استفاده کرد. اما در این حالت نیاز خواهد بود تا مثلا نام جدولی خاص با کلاسی مفروض در برنامه، یا نام فیلدی خاص که مطابق استانداردهای نامگذاری خواص در سی شارپ تعریف نشده، با خاصیتی در یک کلاس تطابق داده شوند. برای مثال اینبار تعاریف کلاس Blog را به نحو زیر تغییر دهید:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace EF_Sample01.Models
{
    [Table("tblBlogs")]
    public class Blog
    {
        [Column("MyTableKey")]
        public int Id { set; get; }

        [MaxLength(100)]
        public string Title { set; get; }

        [Required(ErrorMessage = "لطفا نام نویسنده را مشخص نمائید")]
        public string AuthorName { set; get; }

        public IList<Post> Posts { set; get; }

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

در اینجا فرض بر این است که نام جدول متناظر با کلاس Blog در بانک اطلاعاتی مثلا tblBlogs است و نام خاصیت Id در بانک اطلاعاتی مساوی فیلدی است به نام MyTableKey. چون نام خاصیت را مجددا به Id تغییر داده‌ایم، دیگر ضرورتی به ذکر ویژگی Key وجود نداشته است. برای تعریف این دو از ویژگی‌های Table و Column جهت سفارشی سازی نام‌های خواص و کلاس استفاده شده است.

یا اگر در کلاس خود خاصیتی محاسبه شده بر اساس سایر خواص، تعریف شده است و قصد نداریم آن‌را به فیلدی در بانک اطلاعاتی نگاشت کنیم، می‌توان از ویژگی NotMapped برای مزین سازی و تعریف آن کمک گرفت.

به علاوه اگر از نام پیش فرض کلید خارجی تشکیل شده خرسند نیستید می‌توان به کمک ویژگی ForeignKey، نسبت به تعریف مقداری جدید مطابق تعاریف یک بانک اطلاعاتی موجود، اقدام کرد.

همچنین خاصیت دیگری به نام RowVersion در اینجا اضافه شده که با ویژگی TimeStamp مزین گردیده است. از این خاصیت ویژه برای بررسی مسایل همزمانی ثبت اطلاعات در EF استفاده می‌شود. به علاوه بانک اطلاعاتی می‌تواند به صورت خودکار آن‌را در حین ثبت مقدار دهی کند.

تمام این تغییرات را به کمک Fluent API نیز می‌توان انجام داد:

```
modelBuilder.Entity<Blog>().ToTable("tblBlogs");
modelBuilder.Entity<Blog>().Property(x => x.Id).HasColumnName("MyTableKey");
modelBuilder.Entity<Blog>().Property(x => x.RowVersion).HasColumnType("Timestamp");
```

تبدیل پروژه‌های قدیمی EF به کلاس‌های EF Code first به صورت خودکار

روش متداول کار با EF از روز اول آن، مهندسی معکوس خودکار اطلاعات یک بانک اطلاعاتی و تبدیل آن به یک فایل EDMX بوده است. هنوز هم می‌توان از این روش در اینجا نیز بهره جست. برای مثال اگر قصد دارید یک پروژه قدیمی را تبدیل به نمونه جدید

Code first کنید، یا یک بانک اطلاعاتی موجود را مهندسی معکوس کنید، بر روی پروژه در Solution explorer کلیک راست کرده و گزینه Add|New Item را انتخاب کنید. سپس از صفحه ظاهر شده، ADO.NET Entity data model را انتخاب کرده و در ادامه گزینه «Generate from database» را انتخاب کنید. این روال مرسوم کار با EF Database first است.
پس از اتمام کار به entity data model designer مراجعه کرده و بر روی صفحه کلیک راست نمائید. از منوی ظاهر شده گزینه «Add code generation item» را انتخاب کنید. سپس در صفحه باز شده از لیست قالب‌های موجود، گزینه «ADO.NET DbContext Generator» را انتخاب نمائید. این گزینه به صورت خودکار اطلاعات فایل EDMX قدیمی یا موجود شما را تبدیل به کلاس‌های مدل Code first معادل به همراه کلاس DbContext معرف آن‌ها خواهد کرد.

روش دیگری نیز برای انجام اینکار وجود دارد. نیاز است افزونه‌ی به نام Entity Framework Power Tools را دریافت کنید. پس از نصب، از منوی Entity Framework آن گزینه‌ی «Reverse Engineer Code First» را انتخاب نمائید. در اینجا می‌توان مشخصات اتصال به بانک اطلاعاتی را تعریف و سپس نسبت به تولید خودکار کدهای مدل‌ها و DbContext مرتبط اقدام کرد.

استراتژی‌های مقدماتی تشکیل بانک اطلاعاتی در EF Code first

اگر مثال این سری را دنبال کرده باشید، مشاهده کرده‌اید که با اولین بار اجرای برنامه، یک بانک اطلاعاتی پیش فرض نیز تولید خواهد شد. یا اگر تعاریف ویژگی‌های یک فیلد را تغییر دادیم، نیاز است تا بانک اطلاعاتی را دستی drop کرده و اجازه دهیم تا بانک اطلاعاتی جدیدی بر اساس تعاریف جدید مدل‌ها تشکیل شود که ... هیچکدام از این‌ها بهینه نیستند.
در اینجا دو استراتژی مقدماتی را در حین آغاز یک برنامه می‌توان تعریف کرد:

```
System.Data.Entity.Database.SetInitializer(new DropCreateDatabaseIfModelChanges<Context>());
// or
System.Data.Entity.Database.SetInitializer(new DropCreateDatabaseAlways<Context>());
```

می‌توان بانک اطلاعاتی را در صورت تغییر اطلاعات یک مدل به صورت خودکار drop کرده و نسبت به ایجاد نمونه‌ای جدید اقدام کرد (DropCreateDatabaseIfModelChanges)؛ یا در حین آزمایش برنامه همیشه (DropCreateDatabaseAlways) با شروع برنامه، ابتدا باید بانک اطلاعاتی drop شده و سپس نمونه جدیدی تولید گردد.
محل فراخوانی این دستور هم باید در نقطه آغازین برنامه، پیش از وهله سازی اولین DbContext باشد. مثلا در برنامه‌های وب در متد Application_Start فایل global.asax.cs یا در برنامه‌های WPF در متد سازنده کلاس App می‌توان بانک اطلاعاتی را آغاز نمود.
البته الزامی به استفاده از کلاس‌های DropCreateDatabaseIfModelChanges یا DropCreateDatabaseAlways وجود ندارد.
می‌توان با پیاده سازی اینترفیس IDatabaseInitializer از نوع کلاس Context تعریف شده در برنامه، همان عملیات را شبیه سازی کرد یا سفارشی نمود:

```
public class MyInitializer : IDatabaseInitializer<Context>
{
        public void InitializeDatabase(Context context)
        {
            if (context.Database.Exists() ||
                context.Database.CompatibleWithModel(throwIfNoMetadata: false))
                context.Database.Delete();

            context.Database.Create();
        }
}
```

سپس برای استفاده از این کلاس در ابتدای برنامه، خواهیم داشت:

```
System.Data.Entity.Database.SetInitializer(new MyInitializer());
```

نکته:
اگر از یک بانک اطلاعاتی موجود استفاده می‌کنید (محیط کاری) و نیازی به پیش فرض‌های EF Code first ندارید و همچنین این بانک اطلاعاتی نیز نباید drop شود یا تغییر کند، می‌توانید تمام این پیش فرض‌ها را با دستور زیر غیرفعال کنید:

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

بدیهی است این دستور نیز باید پیش از ایجاد اولین وهله از شیء DbContext فراخوانی شود.

همچنین باید درنظر داشت که در آخرین نگارش‌های پایدار EF Code first، این موارد بهبود یافته‌اند و مبحثی تحت عنوان DB Migration ایجاد شده است تا نیازی نباشد هربار بانک اطلاعاتی drop شود و تمام اطلاعات از دست برود. می‌توان صرفا تغییرات کلاس‌ها را به بانک اطلاعاتی اعمال کرد که به صورت جداگانه، در قسمتی مجزا بررسی خواهد شد. به این ترتیب دیگر نیازی به drop بانک اطلاعاتی نخواهد بود. به صورت پیش فرض در صورت از دست رفتن اطلاعات یک استثناء را سبب خواهد شد (که توسط برنامه نویس قابل تنظیم است) و در حالت خودکار یا دستی با تنظیمات ویژه قابل اعمال است.

تنظیم استراتژی‌های آغاز بانک اطلاعاتی در فایل کانفیگ برنامه

الزامی ندارد که حتما متد Database.SetInitializer را دستی فراخوانی کنیم. با اندکی تنظیم فایل‌های app.config و یا web.config نیز می‌توان نوع استراتژی مورد استفاده را تعیین کرد:

```
<appSettings>
 <add key="DatabaseInitializerForType MyNamespace.MyDbContextClass, MyAssembly"
     value="MyNamespace.MyInitializerClass, MyAssembly"  />
</appSettings>

<appSettings>
 <add key="DatabaseInitializerForType MyNamespace.MyDbContextClass, MyAssembly"
     value="Disabled" />
</appSettings>
```

یکی از دو حالت فوق باید در قسمت appSettings فایل کانفیگ برنامه تنظیم شود. حالت دوم برای غیرفعال کردن پروسه آغاز بانک اطلاعاتی و اعمال تغییرات به آن، بکار می‌رود.
برای نمونه در مثال جاری، جهت استفاده از کلاس MyInitializer فوق، می‌توان از تنظیم زیر نیز استفاده کرد:

```
<appSettings>
    <add key="DatabaseInitializerForType EF_Sample01.Context, EF_Sample01"
       value="EF_Sample01.MyInitializer, EF_Sample01" />
  </appSettings>
```

اجرای کدهای ویژه در حین تشکیل یک بانک اطلاعاتی جدید

امکان سفارشی سازی این آغاز کننده‌های پیش فرض نیز وجود دارد. برای مثال:

```
public class MyCustomInitializer : DropCreateDatabaseIfModelChanges<Context>
{
    protected override void Seed(Context context)
    {
        context.Blogs.Add(new Blog { AuthorName = "Vahid", Title = ".NET Tips" });
        context.Database.ExecuteSqlCommand("CREATE INDEX IX_title ON tblBlogs (title)");
        base.Seed(context);
    }
}
```

در اینجا با ارث بری از کلاس DropCreateDatabaseIfModelChanges یک آغاز کننده سفارشی را تعریف کرده‌ایم. سپس با تحریف متد Seed آن می‌توان در حین آغاز یک بانک اطلاعاتی، تعدادی رکورد پیش فرض را به آن افزود. کار ذخیره سازی نهایی در متد base.Seed انجام می‌شود.

برای استفاده از آن اینبار در حین فراخوانی متد System.Data.Entity.Database.SetInitializer، از کلاس MyCustomInitializer استفاده خواهیم کرد.

و یا توسط متد context.Database.ExecuteSqlCommand می‌توان دستورات SQL را مستقیما در اینجا اجرا کرد. عموما دستوراتی در اینجا مدنظر هستند که توسط ORMها پشتیبانی نمی‌شوند. برای مثال تغییر collation یک ستون یا افزودن یک ایندکس و مواردی از این دست.

سطح دسترسی مورد نیاز جهت فراخوانی متد Database.SetInitializer

استفاده از متدهای آغاز کننده بانک اطلاعاتی نیاز به سطح دسترسی بر روی بانک اطلاعاتی master را در SQL Server دارند (زیرا با انجام کوئری بر روی این بانک اطلاعاتی مشخص می‌شود، آیا بانک اطلاعاتی مورد نظر پیشتر تعریف شده است یا خیر). البته این مورد حین کار با SQL Server CE شاید اهمیتی نداشته باشد. بنابراین اگر کاربری که با آن به بانک اطلاعاتی متصل می‌شویم سطح دسترسی پایینی دارد نیاز است Persist Security Info=True را به رشته اتصالی اضافه کرد. البته این مورد را پس از انجام تغییرات بر روی بانک اطلاعاتی جهت امنیت بیشتر حذف کنید (یا به عبارتی در محیط کاری Persist Security Info=False باید باشد).

```
Server=(local);Database=yourDatabase;User
ID=yourDBUser;Password=yourDBPassword;Trusted_Connection=False;Persist Security Info=True
```

تعیین Schema و کاربر فراخوان دستورات SQL

در EF Code first به صورت پیش فرض همه چیز بر مبنای کاربری با دسترسی مدیریتی یا dbo schema در اس کیوال سرور تنظیم شده است. اما اگر کاربر خاصی برای کار با دیتابیس تعریف گردد که در هاست‌های اشتراکی بسیار مرسوم است، دیگر از دسترسی مدیریتی dbo خبری نخواهد بود. اینبار نام جداول ما بجای dbo.tableName مثلا someUser.tableName می‌باشند و عدم دقت به این نکته، اجرای برنامه را غیرممکن می‌سازد.

برای تغییر و تعیین صریح کاربر متصل شده به بانک اطلاعاتی اگر از متادیتا استفاده می‌کنید، روش زیر باید بکارگرفته شود:

```
[Table("tblBlogs", Schema="someUser")]
public class Blog
```

و یا در حالت بکارگیری Fluent API به نحو زیر قابل تنظیم است:

```
modelBuilder.Entity<Blog>().ToTable("tblBlogs", schemaName:"someUser");
```

نظرات خوانندگان

نویسنده: Mohammad
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۰:۲۰:۲۴

سلام آقای نصیری.
code first اجازه درست کردن trigger رو می‌ده؟
کلا من یه مکانیزمی می‌خوام که اگه کسی (غیر خودم) تو یه جدول خاص insert کرد با یه چیزی مثل event تو برنامه متوجه بشم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۰:۳۹:۱۵

لطف کنید سؤالی رو که مطرح می‌کنید در حیطه مطلب جاری عنوان شده باشد و خارج از آن نباشد.
سؤال شما هم بحث کلاینت سروری است و نه بحث کلاینت تنها که EF روی آن مشغول به کار است.
- می‌شود در متد Seed ایی که در بالا توضیح دادم در SQL Server تریگر درست کرد. (که مثلا اگر کاربر دیگری به شرط اینکه این کاربر جزو کاربران تعریف شده در خود SQL Server باشد نه در برنامه شما، اتفاق خاصی رخ دهد. برنامه شما هم بدیهی است باید سرور را مدام چک کند تا از این مساله مطلع شود)- SQL Server مبحثی دارد به نام Service Broker : (^) . توسط آن می‌توان از طریق سرور به کلاینت اطلاع رسانی کرد. بازهم خارج است از بحث یک ORM. یا تمام ORMهای موجود. - EF مبحثی دارد به نام Concurrency check که اگر شخصی در شبکه بر روی رکوردی که همین الان شما مشغول به کار هستید، تغییری را ایجاد کرد، به شما اطلاع رسانی کند. (در قسمت‌های بعدی بحث خواهد شد). البته این هم خودکار نیست. لازم است یک رفت و برگشت به سرور انجام شود.- entity framework auditing هم میسر است. خودکار نیست. در همان کلاس Context فوق که از DbContext مشتق می‌شود می‌توان متد تحریف شده public override int SaveChanges را تعریف کرد. در اینجا می‌توان به تمام تغییراتی که قرار است اعمال شوند دسترسی داشت. مثلا آن‌ها را در یک جدول مجزا ثبت کرد. بدیهی است برنامه بعدا نیاز خواهد داشت از این جدول گزارشگیری کند.

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۸:۰۳:۵۰

با سلام و تشکر

سوال اول:
این قسمت آخری را که فرمودید:
"در EF Code first به صورت پیش فرض همه چیز بر مبنای کاربری با دسترسی
مدیریتی یا dbo schema در اس کیوال سرور تنظیم شده است. اما اگر کاربر خاصی
برای کار با دیتابیس تعریف گردد که در هاست‌های اشتراکی بسیار مرسوم است،
دیگر از دسترسی مدیریتی dbo خبری نخواهد بود."

من متوجه نشدم! ما در هاستهای اشتراکی مثلا از طریق پنل پلسک یک بانک به همراه یک کاربر به نام فرضی user1 ایجاد می کنم و در کانشکن استرینگ هم با همین نام کاربری متصل می شویم. حال منظور شما از کاربر خاص یعنی چه کسی؟
این scheme که نام آنرا someUser گذاشتید، مربوط به چه کسی است و از کجا آمده است؟

سوال دوم:
آیا در مورد بانک Membership پیش فرض مایکروسافت و تلفیق آن با بانک اصلی برنامه در EF راه کاری اندیشیده شده؟
بنده هیچ وقت از این امکان به جهت دو دسته شدن جداول و ساختار بانکم استفاده نکردم ولی با توجه به یکپارچه شدن آن با ASP.NET MVC کم کم دارم متقاعد می شوم که به جای منطق Membership خودم از این امکان استفاده کنم، نظر شما در مورد منطق Membership برنامه ای که با EF و MVC نوشته می شود چیست؟

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۸:۱۶:۵۱

آیا EF امکاناتی دارد که به کمک آن بتوان پس از ران شدن برنامه و ساخت بانک به صورت پویا به آن جدول یا فیلد و چیزهایی مثل Data Annotation اضافه کرد و بعد از هم از POCO آنها در حالت Runtime استفاده کرد؟
ترکیب EF Code First با پروژه Roslyn برای رسیدن به این منظور مناسب است؟ (گرچه فکر کنم راسلین هنوز کلاس های پیچیده زبان را پشتیبانی نمی کند)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۸:۴۹:۱۸

بله؛ این امکان وجود دارد که سیستم را افزونه پذیر کرد و مدل‌ها رو در زمان اجرا به DbContext اضافه کرد.
جزئیات آن خارج است از بحث «قسمت دوم» ما.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۸:۵۵:۵۶

- این‌ها مباحث پایه‌ای SQL Server است. در مورد schema ایی که از آن صحبت شد می‌تونید اینجا بیشتر مطالعه کنید: (^)
- شما در ASP.NET MVC می‌تونید این موارد رو سفارشی کنید و از پیاده سازی خودتون استفاده کنید. در یک قسمت مجزا به این مورد پرداختم که در سایت هست (^) . به علاوه پروژه‌هایی مانند این هم وجود دارند: (^)

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۹:۰۷:۰۷

سوال بنده این بود که این someUser که شما به عنوان schema تعریف کرده اید از کجا آمده است؟ متعلق به کدام کاربر است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۵ ۱۹:۲۶:۰۷

در SQL Server برای کار با بانک اطلاعاتی یک سری سطوح امنیتی وجود دارد. عنوان کردید که من یک نام کاربری دارم و پسود و از آن در رشته اتصالی استفاده می‌کنیم. بله. این درسته. اما این فقط ابتدای کار است. زمانیکه کاربری در SQL Server تعریف می‌شود یک سری سطح دسترسی را می‌شود به آن داد یا از آن گرفت. مثلا دسترسی اجرای SPها را نداشته باشد؛ دسترسی drop بانک اطلاعاتی را نداشته باشد؛ یا امکان فراخوانی delete را نداشته باشد. برای اینکه این موارد را بهتر مدیریت کنند یک schema تعریف می‌شود که در حقیقت قالبی است جهت مشخص سازی این سطوح دسترسی. dbo یکی از این قالب‌ها است که جزو مجموعه بالاترین سطوح دسترسی است. در هاست‌های اشتراکی که به مسایل امنیتی اهمیت می‌دهند امکان نداره به شما دسترسی dbo بدن. بله شما نام کاربری و کلمه عبور دارید اما schema سفارشی شما ممکن است دسترسی drop یا create بانک اطلاعاتی رو نداشته باشه (که در هاست‌های خوب ندارید).
این مساله چه مشکلی رو ایجاد می‌کنه؟
اگر کوئری پیش فرض شما select * from dbo.table1 باشد، در یک هاست اشتراکی با سطح دسترسی بالا که به شما دسترسی drop و create یک بانک اطلاعاتی رو نداده، دیگر اجرا نخواهد شد چون dbo نیستید. ضمنا روش صحیح و توصیه شده کار با SQL Server نیز ذکر schema در کوئری‌ها است زیرا سرعت اجرا را بالا می‌برد از این لحاظ که اگر آن‌را ذکر نکنید، SQL Server مجبور خواهد شد دست به سعی و خطا بزند. اما با ذکر آن یک راست از سطح دسترسی صحیح استفاده می‌شود. EF هم از همین روش استفاده می‌کنه. بنابراین لازم است schema رو اینجا در صورت مساوی نبودن با dbo حتما ذکر کرد و گرنه کوئری‌های شما دیگر اجرا نخواهند شد، چون دسترسی dbo رو ندارید.

نویسنده: مرادی
تاریخ: ۱۳۹۱/۰۲/۱۵ ۲۳:۳۹:۲۰

امکانش هست به خاطر عوض کردن اسم یه Property، کل دیتابیس رو Drop نکنه ؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۵ ۲۳:۴۳:۵۷

- می‌تونید SetInitializer رو به null مقدار دهی کنید تا کاری به بانک اطلاعاتی نداشته باشد.
و در حالت کلی، بله. مبحثی هست که اخیرا اضافه شده به نام DB Migration که در چند قسمت بعد به آن خواهیم رسید.

نویسنده: Navid
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۳:۳۹:۰۰

با سلام خدمت جناب نصیری
بنده از روی مثاله شما جلو رفتم و الگوی خودم رو در آوردم به این ترتیب که :
یک کلاس پایه ابسترکت درست کردم که یکسری پراپرتی در آن تعریف نمودم(فرضا کلاس جانداران)
چند کلاس دیگر ایجاد کردم که از کلاس پایه ارث برده بودند و پراپرتی های خود را هم داشتند
زمانیکه برنامه را اجرا کردم و دیتابیس از روی برنامه ایجاد شد ، تنها به ازای کلاس های فرزند جدول درست شده بود ولی تمامی پراپرتی های کلاس پایه در جداول ایجاد شده بود اما جدولی به اسم جاندار (کلاس پایه) ایجاد نشده بود .
سواله بنده اینه که آیا بایستی کاره خاصی کرد که یک جدول جداگانه برای کلاس پایه هم ایجاد شود ؟
من این کار رو توی مدل فرست بدون مشکل انجام میدادم و نتیجه کار هم خوب بود .
با تشکر از زحمات جنابعالی

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۴:۲۶:۳۱

من در قسمت «««««««««««««««««««««دوم»»»»»»»»»»»»»»»»»»»» از «ارث بری» و مباحث Table-per-Type Inheritance و Table-per-Hierarchy strategy صحبتی نکردم. این‌ها در قسمت‌های
«««««««««««««««««««««««««««««بعد»»»»»»»»»»»»»»»»»»»»»»»»»»»»»» بررسی می‌شوند.

نویسنده: مرادی
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۴:۳۱:۳۴

ممنون از پاسخ دهی شما، ولی فکر کنم منظورم رو بد رسوندم، من دقیقا می خوام برام Alter Table بزنه، نه این که کاری به دیتابیس نداشته باشه، (مثل NHibernate)

آیا با Migration ها چنین چیزی به سادگی (در حد چند خط کد قابل استفاده برای کل برنامه)، امکانپذیر هست یا خیر ؟
متاسفانه من همچین چیزی ندیدم تو امکاناتش

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۶ ۱۴:۳۶:۱۴

بله. بحث DB Migration روی همین مورد تمرکز دارد. جزئیات آن مفصل است. بسیار کاملتر است از نمونه NHibernate و طراحی مهندسی قابل توجهی دارد. شاید توضیحش نیاز به دو جلسه داشته باشد.

نویسنده: Salehi
تاریخ: ۱۳۹۱/۰۲/۱۹ ۰۱:۴۱:۴۸

اگه از خاصیت RowVersion استفاده نشه، همزمانی کنترل نمی شه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۹ ۰۹:۱۴:۴۰

نه. همانطوری که کوئری را صادر کردید اجرا می‌شود. یعنی فقط بر اساس primary key کار آیدیت یا delete را انجام می‌دهد.

نویسنده: Salehi
تاریخ: ۱۳۹۱/۰۲/۱۹ ۱۳:۱۹:۵۸

-1آیا منظور از کنترل همزمانی اینه : "من یه رکورد رو گرفتم و ویرایش کردم، و قبل از ثبت ، شخص دیگه ای اون رو ویرایش و ثبت کرد، اجازه ثبت رکورد ویرایش شده به من داده نمی شه"
-2 در حالت database first که این فیلد وجود نداره چطور؟ اونجا که کنترل همزمانی انجام میشه. اونجا از چه روشی استفاده می کنه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۹ ۱۴:۳۲:۰۵

- بله. استثنای زیر را دریافت خواهید کرد:
Entities may have been modified or deleted since entities were loaded
- اونجا هم وجود داره. در طراح EF در VS.NET یک فیلد رو می‌تونید انتخاب کنید. بعد در برگه خواص، ConcurrencyMode رو تعیین کنید.

نویسنده: peyman
تاریخ: ۱۳۹۱/۰۴/۰۴ ۱۷:۲۵

سلام مهندس . EF 4.3 دسترسی به EntityTypeConfiguration ندارم . چون میخوام بصورت Fluent کار مپینگ رو انجام بدم و یک نفر اشاره کرده بود که از 4.1 به اینور حذف شده آیا درسته ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۴ ۱۷:۳۱

خیر. در قسمت ششم توضیح دادم.

نویسنده: torisoft
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۰:۱۶

سلام جناب نصیری
ببخشید سوالات من در سطح پائینیه و وقت شمارو هم میگیره ولی خوب....
پروژه من بصورت زیر تعریف شده :
1- MVVMLight SL5 بدون هیچ هاستی
2- Wcf service که تو این پروژه اومدم هاست رو تعریف کردم و همچنین پروژه SL رو در Properties این قسمت Add کردم
3- دو پروژه مجزا مطابق با درس شما DataLayer و DomainClasses
پروژه بعد از Run شدن دیتابیس رو تشکیل نمیده ضمن اینکه هیچ خطا یا هشداری هم ندارم.
لطفا در صورت فرصت راهنمائی بفرمائید.
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۰:۵۵

در برنامه‌های وب، قسمت Database.SetInitializer باید در روال Application_Start فایل Global.asax.cs اضافه شود.
ضمن اینکه مکان تعریف رشته اتصالی به بانک اطلاعاتی اینبار فایل web.config خواهد بود.

نویسنده: torisoft
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۹:۲۵

سلام جناب نصیری
ممنون از پاسخگوئیتون
مواردی رو که شما فرمودید انجام دادم(رشته اتصالی به بانک اطلاعاتی در web.config از قبل درست بود) با سه حالت مختلف
1- فقط از متد Database.SetInitializer در روال Application_Start استفاده کردم که با خطای زیر مواجه شدم

```
GenericArguments[0], 'DataLayer.TestContext', on
'System.Data.Entity.IDatabaseInitializer`1[TContext]' violates the
constraint of type parameter 'TContext'.
```

2- از متد Database.SetInitializer صرف نظر کردم و موارد زیر رو به web.config اضافه کردم

```
<contexts>
    <context type=" DataLayer.TestContext, DataLayer">
      <databaseInitializer type="System.Data.Entity.DropCreateDatabaseAlways`1[
                            [DataLayer.TestContext, DataLayer]], EntityFramework" />
    </context>
  </contexts>
```

که با خطای زیر مواجه شدم

```
An error occurred during the processing of a configuration file required to service this request.
Please review the specific error details below and modify your configuration file appropriately
```

3- فقط از <appSettings> استفاده کردم که بدون هیچ خطا و هشداری بود ولی باز هم دیتابیس تشکیل نشد.
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۹:۳۳

به نظر می‌رسه یک آغاز کننده سفارشی رو تهیه کردید و نوع جنریک ارسالی به آن از DbContext مشتق نشده. این مساله بیشتر زمانی رخ می‌ده که در پروژه جاری از چند نگارش مختلف EF در حال استفاده هستید. مثلا لایه سرویس EF A.1 است و لایه دیگر EF A.2 و فایل کانفیگ برنامه به نگارش A.3 اشاره می‌کند. همه رو باید یک دست کنید.

نویسنده: torisoft
تاریخ: ۱۳۹۱/۰۴/۱۵ ۱۷:۵۷

سلام جناب نصیری
یه سوال
نحوه ارتباط wcf با model در mvvm با فرض استفاده از تکنولوژی code first به چه صورت است ؟
آیا از DomainClasses می‌توان به عنوان model در mvvm استفاده کرد ؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۵ ۱۸:۵۶

بهتر است که model یک view با domain model یکی نباشد. هر view ممکن است در عمل فقط به تعدادی فیلد محدود نیاز داشته باشد. این‌ها با entityهای تعریف شده یکی نیستند و ضرورتی هم ندارد یکی باشند. حتی ممکن است جهت ارائه یک View تعدادی خاصیت جدید را هم تعریف کنید، اما از حاصل محاسباتی خاص بر روی آن‌ها، یک نتیجه را در بانک اطلاعاتی ثبت کنید.
ضمن اینکه تا این سری 15 قسمتی که به عمد با برنامه کنسول جلو رفته رو تموم نکنید، درک صحیحی از اجزای مختلف آن پیدا نخواهید کرد.
هر زمانی هم خواستید مطلبی را در این سطح آموزش دهید با برنامه‌ی کنسول کار کنید چون هدف در اینجا نحوه نمایش آن با

سیلورلایت یا asp.net یا winforms و غیره نیست. هدف آشنایی با زیرساخت‌های اصلی یک فناوری است؛ صرفنظر از نحوه نمایش آن به کاربر.
چگونه باید کلاس‌ها را به بانک اطلاعاتی نگاشت کرد. چگونه باید پس از تغییر کلاس‌ها، بانک اطلاعاتی را با برنامه هماهنگ کرد. چطور باید حالت‌های یک به چند و امثال آن‌را تعریف کرد. چطور باید یک Context را صحیح مدیریت کرد و غیره. هدف این سری، این نوع مباحث پایه‌ای بوده نه فناوری نمایش نهایی آن.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۶ ۱۳:۵۵

یک مورد را هم اضافه کنم. تا زمانیکه اولین کوئری، به بانک اطلاعاتی ارسال نشود، کار آغاز دیتابیس انجام نشده و تا آن زمان به تاخیر خواهد افتاد. بنابراین اجرای برنامه به معنای ساخت همزمان بانک اطلاعاتی نخواهد بود.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۱/۰۴/۲۳ ۱۶:۴

با سلام
در صورتی که بخواهم از یک Initializer استفاده کنم و کلاس رو از DropCreateDatabaseAlways به ارث ببرم . و در متد Seed مانند مثال همین جلسه دیتابیس را مقدار دهی اولیه کنم با خطای Cannot drop database "testdb2012" because it is currently in use روبرو می‌شوم و اصلا متد Seed فراخوانی نمی‌شود . در صورتی که در مثال نمونه که (^) در بخش ارسال نظر گذاشته اید این مورد به درستی اجرا میشود . آیا به نوع پروژه بستگی دارد ؟ پروژه ای که من ایجاد کردم از نوع Consol می‌باشد.

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۳ ۱۶:۳۳

پیغام خطای «it is currently in use» جزو پیغام‌های معروف SQL Server است. زمانیکه در SQL Server بانک اطلاعاتی مورد نظر در حال استفاده باشد، امکان drop آن نیست.
اگر تنها در محیط توسعه خودتان دارید با SQL Server لوکال سیستم کار می‌کنید، این خطا به معنای باز بودن یک کانکشن از management studio به SQL Server است که با بستن management studio مشکل حل می‌شود.
اگر دیتابیس کاری است یا دیتابیس راه دور است، باید بانک اطلاعاتی را به حالت single user دربیارید و بعد می‌تونید اون رو دراپ کنید.

نویسنده: احمد احمدی
تاریخ: ۱۳۹۱/۰۶/۲۶ ۲۳:۴

با سلام و تشکر بخاطر مقالات مفید EF
بنده طبق مثال مقاله پیش رفتم و متادیتاهای Key و Required را اضافه کردم اما با متادیتای MaxLength به مشکل خوردم .
ویژوال همچین پیغامی میده :

```
/*
The type 'System.ComponentModel.DataAnnotations.MaxLengthAttribute' exists in both

'c:\Program Files\Microsoft ADO.NET Entity Framework 4.1\Binaries\EntityFramework.dll'

and

'c:\Program Files\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5\System.ComponentModel.DataAnnotations.dll'
*/
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۶ ۲۳:۱۷

شما در حال استفاده از EF 4.1 با دات نت 4 و نیم هستید. این دو با هم سازگاری ندارند. از EF 5 با دات نت 4 و نیم استفاده کنید تا مشکل تداخل فضاهای نامی که ذکر شده، برطرف شود.

نویسنده: احمد احمدی
تاریخ: ۱۳۹۱/۰۶/۲۷ ۰:۱

بسیار ممنون - مشکل با نصب این پک برطرف شد :

```
/*
Install-Package EntityFramework -Version 5.0.0-beta2 -Pre
*/
```

فقط :

امکان نصب EF5 بدون Nuget وجود نداره؟

اگر خیر ، پس باید برای هر پروژه یکبار یک دستور رو اجرا کنیم؟

اگر بخوایم پروژه رو روی یک سیستم دیگه که EF5 نداره اجرا کنیم تکلیف چیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۷ ۰:۱۰

- EF 5 نسخه نهایی دارد. نیازی به نسخه بتا نیست.
- بهترین روش استفاده از NuGet است چون دفعه‌ی بعد که به روز رسانی شد به شما اطلاع خواهد داد. مانند به روز رسانی افزونه‌های فایرفاکس. به علاوه سورس آن هم در CodePlex موجود است .
- زمانیکه یکبار دریافت شد در پوشه packages شما قرار می‌گیرد. به این صورت در پروژه‌های دیگر هم قابل ارجاع است.
- توزیع فایل dll آن به همراه برنامه. نیازی به نصب ندارد.

نویسنده: MehRad
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۳:۵۹

با سلام

اگر برنامه ما به شکلی باشد که ماژول پذیر باشد و در DLL جداگانه همانند این مثال یک ماژول با نام Blog داشته باشیم برای اضافه کردن به Context باید چگونه عمل نماییم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۴:۳

« خودکار کردن تعاریف DbSetها در EF Code first »

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۲/۲۴ ۱۰:۲

با سلام (چند سوال)

۱- زمانی که ef کار میکردم اگه میومدیم تیبل درست میکردیم وداخل مثلا۰۰۳ رکورد بود اگه وسط کار این جدوال تغییرات لازم داشت دوباره اسکریپت میگرفتیم تمامی رکوردها حذف و دوبار جدوال ساخته میشد این مشکل در ef حل میشد.ایادر efcf هست که میدونم نیست چطوری حل کرده.مخصوصا در ef

۲- من در efcf امدم جدوال دستی از بانک حذف کردم الان که تغییران رو میدم میخام دوباره درست کنم این خطا رو میده

.An error occurred while updating the entries. See the inner exception for details
البته قبلا ش یه خطا دیگه میداد

The model backing the 'MyDBContext' context has changed since the database was created. Either manually delete/update the database, or call Database.SetInitializer with an IDatabaseInitializer instance. For example, the DropCreateDatabaseIfModelChanges strategy will automatically delete and recreate the database, and optionally seed it with new data

که با این دستور حلش کردم

```
System.Data.Entity.Database.SetInitializer<Context>(null);
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۴ ۱۰:۱۰

- این هم EF هست. یکی database first، یکی code first و یکی model first. ولی زیر ساخت همشون یکی هست.
- اکثر خطاهای EF به صورت inner exception است. یعنی صفحه نمایش استثناء رو باید باز کنید و کمی درخت نمایش داده شده را پیمایش کنید تا به inner exception برسید.
- ریز مسایل به روز رسانی بانک اطلاعاتی، در قسمت‌های 4 و 5 این سری بررسی شده. عجله نکنید. قدم به قدم ...

نویسنده: میثم خوشقدم
تاریخ: ۱۳۹۲/۰۲/۱۳ ۱:۱۰

سلام

من از ابزار Power tools جهت استخراج Contex از دیتابیس موجود استفاده نکردم بلکه از Ado.net data model خود ویژوال استدیو اما مدلی که به من میده خیلی شلوغ و پیچیده است و قابل استفاده نیست و این درحالی است که من یک جدول ساده بدون هیچ گونه ریلیشن و یا روابط پیچیده را استفاده کردم !

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۳ ۹:۱۲

یکی از مزایای Code first همین مسایل است. چون کد رو ابزار تولید نمی‌کنه تمیزتر هست. ضمنا روش Database first که شما رفتید، به ObjectContext ختم میشه اما روش Code first به DbContext. به علاوه DbContext هم می‌تونه سفارشی سازی داشته باشه مثل اضافه شدن تعاریف Fluent API .

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۳/۲۸ ۰:۱

با سلام؛ الان که داشتم این بخش رو می‌خوندم به این قسمت رسیدم

البته در این حالت امکان تعریف ErrorMessage وجود ندارد و برای این منظور باید از همان data annotations استفاده کرد.

حال با توجه به این مطلب آیا بهتره که در MVC از annotations به جای Fluent API استفاده کنیم ؟ (چون با استفاده از Fluent API نمی توانیم متن خطا را ایجاد کنیم)

ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۲۸ ۰:۳۳

عموما در یک پروژه واقعی، از ترکیبی از هر دو روش استفاده می‌شود.
در قسمت‌های بعد مواردی عنوان شده‌اند که با ویژگی‌ها قابل تنظیم نیست؛ مثلا تعیین نام سفارشی جدول واسط چند به چند یا تنظیم روابط خود ارجاع دهنده و موارد پیشرفته دیگری.

نویسنده: پوریا یک کدنویس
تاریخ: ۱۳۹۲/۱۰/۰۴ ۱۰:۳۳

بنده از Nuget نسخه EF6 رو نصب کردم...
در بخشی گفتید که

```
[Column("MyTableKey")]
public int Id { set; get; }
```

اما فکر کنم در این نسخه خاصیت Column وجود نداره... معادلش وجود داره؟ یا حذف شده این خاصیت؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۴ ۱۰:۴۲

- حذف نشده. منتقل شده به فضای نام System.ComponentModel.DataAnnotations.Schema .
- عموما در VS.NET اگر اشاره‌گر را در محل یک کلاس نامشخص قرار دهید، یک منوی drop down ظاهر می‌شود که امکان انتخاب فضای نام ممکن و یافت شده‌ای را برای آن میسر می‌کند.

نویسنده: vici
تاریخ: ۱۳۹۲/۱۱/۰۶ ۹:۱۰

سلام؛ اسم schemaName هر چیزی می‌تونه باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۶ ۱۰:۱۲

بحث استفاده از چند Context در یک برنامه و نظرات آن‌را مطالعه کنید. بررسی معماری چند مستاجری هم مفید است.

نویسنده: امیر حسین
تاریخ: ۱۳۹۲/۱۱/۱۱ ۲:۲۹

با توجه به اینکه در یک فیلد RowVersion جهت همزمانی ایجاد نمودین ، آیا این فیلد رو باید برای همه جداول در نظر گرفت ؟ به همه جداول این فیلد رو اضافه کنم ؟

```
modelBuilder.Entity<Blog>().ToTable("tblBlogs", schemaName:"someUser");
```

در مورد کد بالا : آیا اگر رشته اتصال با نام user1 بود این قسمت رو به این صورت پر کنم ؟

```
modelBuilder.Entity<Blog>().ToTable("tblBlogs", schemaName:"user1");
```

چطور متوجه بشم در هاست اشتراکی از چه schema استفاده میکنم ؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۳:۲۵

- به هر قسمتی که فکر می‌کنید این مورد همزمانی ورود یا ویرایش اطلاعات امکان رخ دادن دارد، بله. بهتر است اضافه شود.
- تا زمانیکه خطایی نگرفتید، هیچ تنظیمی را تغییر ندهید.
برای یافتن schema، با استفاده از management studio به سرور متصل شده و یک جدول ساده را درست کنید. بعد مشاهده کنید که ابتدای نام جدول، به چه صورتی شروع شده. مثلا user1.table1 است؟ یا مورد دیگری.

نویسنده: جوادنبی
تاریخ: ۱۳۹۳/۰۱/۱۲ ۱۲:۵۱

با سلام من کد زیر را در application_Start برنامه ام گذاشته ام

```
System.Data.Entity.Database.SetInitializer(new DropCreateDatabaseAlways<ProductContext>());
```

اما خطای به این صورت می‌دهد.

.Cannot drop database "testdb" because it is currently in use

باید چه کار انجام دهم تا هر دفعه که پروژه‌ام را اجرا می‌کنم دیتابیس از نو ساخته بشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۲ ۱۳:۲

- در SQL Server اگر تنها یک کانکشن باز به دیتابیس مفروضی وجود داشته باشد، امکان drop آن‌را نمی‌دهد. برای مثال اگر همزمان management studio هم باز است، این مورد یعنی یک کانکشن باز. آن‌را ببندید تا SQL Server به این نتیجه برسد که کسی از بانک اطلاعاتی درخواستی در حال استفاده نیست.
- در کل رویه ذخیره شده‌ی سیستمی به نام SP_WHO وجود دارد که مصرف کنندگان را لیست می‌کند. شماره آن‌ها را یافته و سپس توسط رویه ذخیره شده دیگری به نام Kill، حذفشان کنید.
- روش دیگر drop آنی یک بانک اطلاعاتی، تک کاربره کردن و سپس حذف آن است:

```
alter database [MyDatbase] set single_user with rollback immediate
 drop database [MyDatabase]
```

نویسنده: علی
تاریخ: ۱۳۹۳/۰۸/۱۰ ۱۹:۴۸

با سلام

من تمام مراحل توضیح داده شده رو انجام دادم ولی متاسفانه دیتابیس رو نمیسازه که ناچار شدم برم از خود منو ado.net entity data modal بسازم ولی باز مشکل اینه که تیبل‌ها رو اضافه نمی‌کنه ممنون میشم راهنمایی بفرمایید

```
protected void Application_Start()
        {

            System.Data.Entity.Database.SetInitializer(new DropCreateDatabaseAlways<SchoolContext>());

            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
   public partial class SchoolContext : DbContext
    {
        public SchoolContext(): base("name=SchoolDBContext")
        {
           // Database.SetInitializer<SchoolContext>(new CreateDatabaseIfNotExists<SchoolContext>());
        }
        public DbSet<student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Course> Courses { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
```

```
        {
        }

        public class Sample2DbInitializer : DropCreateDatabaseAlways<SchoolContext>
        {
            protected override void Seed(SchoolContext context)
            {
                context.Courses.Add(new Course
                {
                    Credits = 20,
                    Title = "Vahid"

                });

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

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۰ ۲۰:۷

« وادار کردن EF Code first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه »

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۸/۱۴ ۲۲:۲۵

ممنون بابت متن و توضیحاتتون
نکته ای که الان در مورد هاست‌های اشتراکی زدید این بود که اومدید schema رو برای یک کلاس تعریف کردید
آیا این امکان هست که بتونیم schema رو به تمامی کلاس‌ها ست کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۴ ۲۲:۳۱

« بازنویسی ساده‌تر پیش فرض‌های EF Code first در نگارش 6 آن »

عنوان: **تعیین شمای جداول بانکی در EF Code First**
نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۴/۱۰ ۱۵:۴۸
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

کلاس‌های زیر را در نظر بگیرید:

```
public class Customer
{
   public virtual string Id { get; set; }
   public virtual string Name { get; set; }
   public virtual DateTime HireDate { get; set; }
}
```

و Context ی که کلاس فوق به آن اضافه میشود:

```
public class MyContext : DbContext
{
    public DbSet<Customer> Customers{ set; get; }
}
```

حالا زمان ایجاد جدول، با توجه به دسترسی کاربر بانک داده، جداول ایجاد میشوند:

```
CREATE TABLE Customers (
    [Id] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    [HireDate] datetime,
    CONSTRAINT [PK_Product] PRIMARY KEY ([Id])
)
```

ولی به محض ارسال درخواست و عدم دسترسی و نیز اقدام به ایجاد دوباره جداول در صورت نبود آن، به مشکل زیر بر میخوریم:

```
Invalid object name 'dbo.Customers'.
```

1شکال در شمای پیش فرض Entityframework است که در زمان درخواست بصورت پیش فرض از dbo استفاده میکند، برای حل مشکل در هنگام نگاشت دو راه موجود است:

راه حل اول(مزین کردن کلاس):

```
[Table("Customers", Schema = "dbSchemaUser")]
public class Customer
{
    ...
```

راه حل دوم(استفاده از نگاشت کننده در context):

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
   modelBuilder.Entity<Customer>().ToTable(tableName: "Customers", schemaName: "dbSchemaUser");
   ...
```

که نتیجه کار بصورت عبارات sql زیر می‌باشد:

```
CREATE TABLE dbSchemaUser.Customers (
    [Id] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    [HireDate] datetime,
    CONSTRAINT [PK_Product] PRIMARY KEY ([Id])
```

```
)
```

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۷:۱

با توجه به نوع تعریف شما برای Migration میتونید آپدیت شمای بانک رو بصورت اتوماتیک به ef بسپارید:

```
// DAL Layer
    public class Configuration : DbMigrationsConfiguration<MyContext> {
      public Configuration() {
   AutomaticMigrationsEnabled = true;
       AutomaticMigrationDataLossAllowed = true;

 // Global.asax
    void Application_Start(object sender, EventArgs e) {
       Database.SetInitializer(
         new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
// Program.cs
    class Program
    {
        static void Main(string[] args) {
           Database.SetInitializer(
             new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
```

ولی برای درخواست شما از سمت بانک به مدل باید فعلا بطور دستی این کار رو انجام بدید

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: فربد 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.

نویسنده: سوین

تاریخ: ۱۳۹۳/۰۷/۱۸ ۱۰:۱۱

با سلام؛ در یه برنامه حسابداری برای ذخیره عناوین اسناد یه جدول به صورت زیر طراحی کردم

```
public partial class Sanad
    {
        public int Sanad_FixCode { get; set; }
        public int Sanad_Code { get; set; }
        public string Sanad_Date { get; set; }
        .....
    }
```

که پراپرتی Sanad_FixCode به صورت identity و PK هست و شماره ثابت اسناد رو نگهداری می‌کنه و پراپرتی Sanad_Code شماره جاری اسناد رو نگه می‌داره و امکان sort اسناد بر اساس این پراپرتی وجود داره و در موقع افزودن سند جدید به max این پراپرتی یکی افزوده میشه و شماره سند جدید بدست میاد در حالت Single User مشکلی نیست اما در Multi User ممکنه مشکل پیش بیاد و دوتا Sanad_Code یکی ایجاد بشه برای رفع این مشکل ممنون میشم راهنماییم کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۸ ۱۰:۲۰

روی فیلد، ایندکس منحصربفرد ایجاد کنید. در این حالت توسط خود بانک اطلاعاتی تضمین می‌شود که فقط یک کاربر موفق به ثبت شماره سندی مشخص شود.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۰۲ ۱:۲

با سلام و خسته نباشید
الان گفتید که خوب هست کلاس مدل‌ها در یک جای جداگانه نوشته بشن ولی اگه بخوایم در پروژه اصلی از ریسورس‌ها روی متاتگ‌ها استفاده کنیم به چه صورت هست؟
چون ریسورس‌ها در پروژه اصلی قرار دارن و مدل‌ها در یک پروژه دیگه که به این پروژه اصلی رفرنس شدند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۰۲ ۱:۱۴

مراجعه کنید به مطلب و نظرات « ASP.NET MVC #22 ». روش کار یکی هست. در نظرات آن حداقل دو مثال را در این‌باره می‌توانید دریافت کنید.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۲۴ ۱۹:۲۲

یک موردی که وجود داره این هست که انگار [compare] و [notmapped] بهم نمی‌سازن
این مورد رو برای فیلد تکرار کلمه عبور قرار دادم که هم به جدول نگاشت نشه و هم بتونم توی فرم استفاده کنم ولی مثل اینکه این دو بهم نمیسازن
مورد بعدی اینکه وقتی از یک ویو مدل استفاده میشه که از یک مدل ارث بری می‌کنه، ef موقع ساخت دیتابیس ساختار جدول رو از روی ویومدل بر میداره

عنوان: EF Code First #6
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۹ ۰۰:۱۳:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

ادامه بررسی Fluent API جهت تعریف نگاشت کلاس‌ها به بانک اطلاعاتی

در قسمت‌های قبل با استفاده از متادیتا و data annotations جهت بررسی نحوه نگاشت اطلاعات کلاس‌ها به جداول بانک اطلاعاتی آشنا شدیم. اما این موارد تنها قسمتی از توانایی‌های Fluent API مهیا در EF Code first را ارائه می‌دهند. یکی از دلایل آن هم به محدود بودن توانایی‌های ذاتی Attributes بر می‌گردد. برای مثال حین کار با Attributes امکان استفاده از متغیرها یا lambda expressions و امثال آن وجود ندارد. به علاوه شاید عده‌ای علاقمند نباشند تا کلاس‌های خود را با data annotations شلوغ کنند.

در قسمت دوم این سری، مروری مقدماتی داشتیم بر Fluent API. در آنجا ذکر شد که امکان تعریف نگاشت‌ها به کمک توانایی‌های Fluent API به دو روش زیر میسر است:
الف) می‌توان از متد protected override void OnModelCreating در کلاس مشتق شده از DbContext کار را شروع کرد.
ب) و یا اگر بخواهیم کلاس Context برنامه را شلوغ نکنیم بهتر است به ازای هر کلاس مدل برنامه، یک کلاس mapping مشتق شده از EntityTypeConfiguration را تعریف نمائیم. سپس می‌توان این کلاس‌ها را در متد OnModelCreating یاد شده، توسط متد modelBuilder.Configurations.Add جهت استفاده و اعمال، معرفی کرد.

کلاس‌های مدلی را که در این قسمت بررسی خواهیم کرد، همان کلاس‌های User و Project قسمت سوم هستند و هدف این قسمت بیشتر تطابق Fluent API با اطلاعات ارائه شده در قسمت سوم است؛ برای مثال در اینجا چگونه باید از خاصیتی صرفنظر کرد، مسایل همزمانی را اعمال نمود و امثال آن.
بنابراین یک پروژه جدید کنسول را آغاز نمائید. سپس با کمک NuGet ارجاعات لازم را به اسمبلی‌های EF اضافه نمائید.
در پوشه Models این پروژه، سه کلاس تکمیل شده زیر، از قسمت سوم وجود دارند:

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

namespace EF_Sample03.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 FullName
        {
            get { return Name + " " + LastName; }
        }

        public string Email { set; get; }
        public string Description { set; get; }
        public byte[] Photo { set; get; }
        public IList<Project> Projects { set; get; }
        public byte[] RowVersion { set; get; }
        public InterestComponent Interests { set; get; }

        public User()
        {
            Interests = new InterestComponent();
        }
    }
}
```

```
using System;

namespace EF_Sample03.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; }
        public byte[] RowVesrion { set; get; }
    }
}
```

```
namespace EF_Sample03.Models
{
    public class InterestComponent
    {
        public string Interest1 { get; set; }
        public string Interest2 { get; set; }
    }
}
```

سپس یک پوشه جدید به نام Mappings را به پروژه اضافه نمائید. به ازای هر کلاس فوق، یک کلاس جدید را جهت تعاریف اطلاعات نگاشت‌ها به کمک Fluent API اضافه خواهیم کرد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample03.Models;

namespace EF_Sample03.Mappings
{
    public class InterestComponentConfig : ComplexTypeConfiguration<InterestComponent>
    {
        public InterestComponentConfig()
        {
            this.Property(x => x.Interest1).HasMaxLength(450);
            this.Property(x => x.Interest2).HasMaxLength(450);
        }
    }
}
```

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample03.Models;

namespace EF_Sample03.Mappings
{
    public class ProjectConfig : EntityTypeConfiguration<Project>
    {
        public ProjectConfig()
        {
            this.Property(x => x.Description).IsMaxLength();
            this.Property(x => x.RowVesrion).IsRowVersion();
        }
    }
}
```

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample03.Models;
using System.ComponentModel.DataAnnotations;

namespace EF_Sample03.Mappings
{
```

```
    public class UserConfig : EntityTypeConfiguration<User>
    {
        public UserConfig()
        {
            this.HasKey(x => x.Id);
            this.Property(x => x.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
            this.ToTable("tblUser", schemaName: "guest");
            this.Property(p =>
p.AddDate).HasColumnName("CreateDate").HasColumnType("date").IsRequired();
            this.Property(x => x.Name).HasMaxLength(450);
            this.Property(x => x.LastName).IsMaxLength().IsConcurrencyToken();
            this.Property(x => x.Email).IsFixedLength().HasMaxLength(255); //nchar(128)
            this.Property(x => x.Photo).IsOptional();
            this.Property(x => x.RowVersion).IsRowVersion();
            this.Ignore(x => x.FullName);
        }
    }
}
```

توضیحاتی در مورد کلاس‌های تنظیمات نگاشت‌های خواص به جداول و فیلدهای بانک اطلاعاتی

نظم بخشیدن به تعاریف نگاشت‌ها

همانطور که ملاحظه می‌کنید، جهت نظم بیشتر پروژه و شلوغ نشدن متد OnModelCreating کلاس Context برنامه، که در ادامه کدهای آن معرفی خواهد شد، به ازای هر کلاس مدل، یک کلاس تنظیمات نگاشت‌ها را اضافه کرده‌ایم.
کلاس‌های معمولی نگاشت‌ها ازکلاس EntityTypeConfiguration مشتق خواهند شد و جهت تعریف کلاس InterestComponent به عنوان Complex Type، اینبار از کلاس ComplexTypeConfiguration ارث بری شده است.

تعیین طول فیلدها

در کلاس InterestComponentConfig، به کمک متد HasMaxLength، همان کار ویژگی MaxLength را می‌توان شبیه سازی کرد که در نهایت، طول فیلد nvarchar تشکیل شده در بانک اطلاعاتی را مشخص می‌کند. اگر نیاز است این فیلد nvarchar از نوع max باشد، نیازی به تنظیم خاصی نداشته و حالت پیش فرض است یا اینکه می‌توان صریحا از متد IsMaxLength نیز برای معرفی nvarchar max استفاده کرد.

تعیین مسایل همزمانی

در قسمت سوم با ویژگی‌های ConcurrencyCheck و Timestamp آشنا شدیم. در اینجا اگر نوع خاصیت byte array بود و نیاز به تعریف آن به صورت timestamp وجود داشت، می‌توان از متد IsRowVersion استفاده کرد. معادل ویژگی ConcurrencyCheck در اینجا، متد IsConcurrencyToken است.

تعیین کلید اصلی جدول

اگر پیش فرض‌های EF Code first مانند وجود خاصیتی به نام Id یا ClassName+Id رعایت شود، نیازی به کار خاصی نخواهد بود. اما اگر این قراردادها رعایت نشوند، می‌توان از متد HasKey (که نمونه‌ای از آن‌را در کلاس UserConfig فوق مشاهده می‌کنید)، استفاده کرد.

تعیین فیلدهای تولید شده توسط بانک اطلاعاتی

به کمک متد HasDatabaseGeneratedOption، می‌توان مشخص کرد که آیا یک فیلد Identity است و یا یک فیلد محاسباتی ویژه و یا هیچکدام.

تعیین نام جدول و schema آن

اگر نیاز است از قراردادهای نامگذاری خاصی پیروی شود، می‌توان از متد ToTable جهت تعریف نام جدول متناظر با کلاس جاری استفاده کرد. همچنین در اینجا امکان تعریف schema نیز وجود دارد.

تعیین نام و نوع سفارشی فیلدها

همچنین اگر نام فیلدها نیز باید از قراردادهای دیگری پیروی کنند، می‌توان آن‌ها را به صورت صریح توسط متد HasColumnName معرفی کرد. اگر نیاز است این خاصیت به نوع خاصی در بانک اطلاعاتی نگاشت شود، باید از متد HasColumnType کمک گرفت. برای مثال در اینجا بجای نوع datetime، از نوع ویژه date استفاده شده است.

معرفی فیلدها به صورت nchar بجای nvarchar

برای نمونه اگر قرار است هش کلمه عبور در بانک اطلاعاتی ذخیره شود، چون طول آن ثابت می‌باشد، توصیه شده‌است که بجای nvarchar از nchar برای تعریف آن استفاده شود. برای این منظور تنها کافی است از متد IsFixedLength استفاده شود. در این حالت طول پیش فرض 128 برای فیلد درنظر گرفته خواهد شد. بنابراین اگر نیاز است از طول دیگری استفاده شود، می‌توان همانند سابق از متد HasMaxLength کمک گرفت.

ضمنا این فیلدها همگی یونیکد هستند و با n شروع شده‌اند. اگر می‌خواهید از varchar یا char استفاده کنید، می‌توان از متد IsUnicode با پارامتر false استفاده کرد.

معرفی یک فیلد به صورت null پذیر در سمت بانک اطلاعاتی

استفاده از متد IsOptional، فیلد را در سمت بانک اطلاعاتی به صورت فیلدی با امکان پذیرش مقادیر null معرفی می‌کند.

البته در اینجا به صورت پیش فرض byte arrayها به همین نحو معرفی می‌شوند و تنظیم فوق صرفا جهت ارائه توضیحات بیشتر در نظر گرفته شد.

صرفنظر کردن از خواص محاسباتی در تعاریف نگاشت‌ها

با توجه به اینکه خاصیت FullName به صورت یک خاصیت محاسباتی فقط خواندنی، در کدهای برنامه تعریف شده است، با استفاده از متد Ignore، از نگاشت آن به بانک اطلاعاتی جلوگیری خواهیم کرد.

معرفی کلاس‌های تعاریف نگاشت‌ها به برنامه

استفاده از کلاس‌های Config فوق خودکار نیست و نیاز است توسط متد modelBuilder.Configurations.Add معرفی شوند:

```
using System.Data.Entity;
using System.Data.Entity.Migrations;
using EF_Sample03.Mappings;
using EF_Sample03.Models;

namespace EF_Sample03.DataLayer
{
    public class Sample03Context : DbContext
    {
        public DbSet<User> Users { set; get; }
        public DbSet<Project> Projects { set; get; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new InterestComponentConfig());
            modelBuilder.Configurations.Add(new ProjectConfig());
            modelBuilder.Configurations.Add(new UserConfig());

            //modelBuilder.ComplexType<InterestComponent>();
            //modelBuilder.Ignore<InterestComponent>();

            base.OnModelCreating(modelBuilder);
        }
    }

    public class Configuration : DbMigrationsConfiguration<Sample03Context>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(Sample03Context context)
        {
            base.Seed(context);
```

```
        }
    }
}
```

در اینجا کلاس Context برنامه مثال جاری را ملاحظه می‌کنید؛ به همراه کلاس Configuration مهاجرت خودکار که در قسمت‌های قبل بررسی شد.
در متد OnModelCreating نیز می‌توان یک کلاس را از نوع Complex معرفی کرد تا برای آن در بانک اطلاعاتی جدول جداگانه‌ای تعریف نشود. اما باید دقت داشت که اینکار را فقط یکبار می‌توان انجام داد؛ یا توسط کلاس InterestComponentConfig و یا توسط متد modelBuilder.ComplexType. اگر هر دو با هم فراخوانی شوند، EF یک استثناء را صادر خواهد کرد.

و در نهایت، قسمت آغازین برنامه اینبار به شکل زیر خواهد بود که از آغاز کننده MigrateDatabaseToLatestVersion (قسمت چهارم این سری) نیز استفاده کرده است:

```
using System;
using System.Data.Entity;
using EF_Sample03.DataLayer;

namespace EF_Sample03
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample03Context,
Configuration>());

            using (var db = new Sample03Context())
            {
                var project1 = db.Projects.Find(1);
                if (project1 != null)
                {
                    Console.WriteLine(project1.Title);
                }
            }
        }
    }
}
```

ضمنا رشته اتصالی مورد استفاده تعریف شده در فایل کانفیگ برنامه نیز به صورت زیر تعریف شده است:

```
<connectionStrings>
    <clear/>
    <add
       name="Sample03Context"
       connectionString="Data Source=(local);Initial Catalog=testdb2012;Integrated Security = true"
       providerName="System.Data.SqlClient"
      />
/connectionStrings>
```

در قسمت‌های بعد مباحث پیشرفته‌تری از تنظیمات نگاشت‌ها را به کمک Fluent API، بررسی خواهیم کرد. برای مثال روابط ارث بری، many-to-many و ... چگونه تعریف می‌شوند.

نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۷:۰۶:۱۹ ۱۳۹۱/۰۲/۱۹

سلام. بسیار ممنون بابت مقالات مفید.
چند تاسوال. من یک پروژه class library جدا برای DomainClasses و DataLayer ایجاد کردم ولی نمی توانم آنها را به پروژه سیلورلایت خود reference دهم .
دوم اینکه validation در codeFirst را نمی توان بطور کامل در fluent api پیاده سازی کرد و حتما باید annotation بکار برد. نمیشود بطورکامل از fluent api استفاده کرد .

نویسنده: وحید نصیری
تاریخ: ۱۹:۳۲:۴۸ ۱۳۹۱/۰۲/۱۹

- بحث سیلورلایت جدا است. سیلورلایت یک فناوری سمت کاربر است. مثل جاوا اسکریپت. برای دسترسی به سرور نیاز دارد با وب سرویس کار کند. متداول‌ترین آن WCF RIA Services است که نگارش‌های جدید آن امکان استفاده از EF Code first را هم دارد. بنابراین مستقیما نمی‌تونید از «هیچ» ORM ایی در سیلورلایت «مستقیما» استفاده کنید؛ اما ... لایه سرویس سمت سرور شما این امکان را دارد. در مورد WCF RIA Services قبلا مطلب نوشتم (البته مربوط به database first است؛ در آن زمان که نوشته شده): (^) (قسمت 26)
- این رو به نظر در قسمت‌های قبل ذکر کردم که فقط پیغام‌های خطا رو نمی‌تونید اینجا ذکر کنید و گرنه حداکثر طول و فیلداجباری و غیره همان اثر را دارد.

نویسنده: Hassan
تاریخ: ۱۰:۱۷:۴۷ ۱۳۹۱/۰۲/۲۰

Code generatorها تمامی لایه ها را می سازند. Entity Framework Code First می تواند لایه های DAL و BLL را بسازد یا Add lonای برای این کار وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۰:۵۰:۵۵ ۱۳۹۱/۰۲/۲۰

یکی از مهم‌ترین اهداف EF Code first این است که با زیرساخت‌های یک ORM آشنا شوید. نباید سؤال بپرسید database first که مسایل همزمانی رو اعمال می‌کنه؛ ولی اطلاع نداشته باشید که پشت صحنه آن در تنظیمات خواص یک فیلد یا جدول، چه امکاناتی وجود دارد و چه مسایلی از چشم شما دور مانده است. این فرق کسی است که اول کد می‌نویسد و طراحی می‌کند (code first)، با کسی که فقط وابسته است به یک سری ابزار که سازوکار درونی آن‌ها را نمی‌داند (database first).
بنابراین سؤال اینجا است که آیا وظیفه‌ی یک ORMاست که برای شما کدنویسی لایه‌های مختلف را انجام دهد؟ یا اینکه اومدید اینجا یک سطح بالاتر رو تجربه کنید؟
البته ابزار هم وجود دارد مانند MVC Scaffolding که بر مبنای EF code first کار می‌کند و یک Code generator است برای ASP.NET MVC . ولی هدف از این مباحث چیز دیگری است.

نویسنده: Mohammadreza Shakeri
تاریخ: ۱۲:۵۵:۲۳ ۱۳۹۱/۰۲/۲۰

سلام.ممنون به خاطر مطالبی که قرار میدید.
من در قسمت دوم به خطایی برخورد کردم که تو این قسمت هم دوباره با این خطا مواجه شدم.

```
Inconsistent accessibility: property type 'System.Data.Entity.DbSet' is less accessible than property
''EF_Sample03.DataLayer.Sample03Context.Projects
```

اشکال کار چی می تونه باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۰ ۱۳:۲۱:۳۱

پاسخ دقیق نیاز به بررسی کدهای شما دارد ولی ... اشکال کار فقط در سطح دسترسی کلاس‌ها و خواص تعریف شده می‌تواند باشد. برای مثال تعدادی رو مثلا internal class تعریف کردید تعداد دیگر رو public class و از این نوع موارد.

نویسنده: peyman
تاریخ: ۱۳۹۱/۰۴/۰۶ ۱۹:۴۶

سلام آقای نصیری . وقتی EF code First رو در این وب سایت سرچ میکنم تمامی سری آموزشی EF code First بجز شماره 5 نمایش داده میشه ! و در واقع نتونستم شماره 5 رو ببینم .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۶ ۲۱:۲۰

از تگ مربوطه استفاده کنید.

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۵/۲۸ ۱۲:۳۰

سلام آقای نصیری - میخواستم بدونم نحوه ایجاد Unique Constraint روی فیلدهای دیتابیس با روش Code First به چه شکلی است؟
دیتابیس من Sql Ce 4.0 SP1 هستش و از Entity Framework 5.0 هم استفاده میکنم.
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۸ ۱۲:۳۸

در متد Seed، متد زیر را فراخوانی کنید:

```
private static void createUniqueIndex(MyContext context, string tableName, string fieldName)
{
  try
  {
     context.Database.ExecuteSqlCommand("CREATE UNIQUE INDEX [IX_Unique_" + tableName + "_" + fieldName
+ "] ON [" + tableName + "]([" + fieldName + "] ASC);");
  }
  catch { }
}
```

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۵/۳۰ ۱۹:۴۹

واقعاً ممنون آقای نصیری.
حالا اگر بخوایم Unique Constraint رو همزمان روی دو فیلد اعمال کنیم به چه صورت خواهد بود؟ یعنی من توی جدول دیتابیسم CustomerId و ProductId دارم که میخوام هیچ دو رکوردی نباشه که این مقادیرش تکراری باشه.
بی نهایت ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۳۰ ۱۹:۵۴

مراجعه کنید به قسمت هشتم، بحث composite keys .

نویسنده: محسن کریمی
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۰:۵۵

سلام

```
using System.ComponentModel.DataAnnotations;
```

در کلاس userconfig باید کامل بشه به این:

```
using System.ComponentModel.DataAnnotations.Schema;
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۱:۲۲

در **EF 5** جای یک سری از کلاس‌ها تغییر کرده. مثلا ویژگی‌های ComplexType، ForeignKey و ... به فضای نام System.ComponentModel.DataAnnotations.Schema منتقل شده‌اند. در همین حد تغییر جهت کامپایل مجدد کد کفایت می‌کند.

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۲:۲۶

در مورد ASP.NET Web API و UpShot و اینها که از EF 4.3 تو خودشون استفاده کردند چی ؟
متاسفانه دیگه نمی‌شه با assembly binding بشون بگیم که از EF 5 استفاده کنید
چون Runtime خطا می‌دن و مثلا می‌گن که System.ComponentMode.DataAnnotation.ForeignKey در Entity Framework 5 نیست !
بله نیست، چون رفته به DLL مربوط به Component Model.Data Annotation
راه حلی جز گرفتن سورس کد upshot و Build مجددش هست ؟
و غیر از عقب گرد به EF 3
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۳:۲۵

- البته EF 5 فقط یک نام تجاری است. نگارش اسمبلی آن 4.4.0.0 است.
- assembly binding هم باید کار کنه چون فضای نام System.ComponentModel.DataAnnotations.Schema داخل خود اسمبلی جدید EF هست؛ هرچند به ظاهر جزئی از یک اسمبلی دیگر به نظر می‌رسد، که در عمل اینطور نیست. به این ترتیب امکان استفاده از EF5 در برنامه‌های دات نت 4 هم هست.

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۵/۱۱ ۱۸:۲۰

سلام.
اگر بخواهیم برای فیلدی که آن را Identity کرده‌ایم، مقادیر Seed و Step را تغییر دهیم ، مثلاً هر دو را از یک به منفی یک تنظیم کنیم، راهکار چیست؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۵/۱۱ ۱۹:۲۶

برای هر قابلیتی که در تنظیمات نگاشت‌ها وجود ندارد و سفارشی است باید در تعاریف Migrations در متد Seed آن، SQL مرتبط را ذکر کنید. مانند « ایجاد ایندکس منحصربفرد در EF Code first » و یا « ایندکس منحصر به فرد با استفاده از Data Annotation در EF Code First ». اصول و روش کار یکی است؛ فقط کوئری SQL ایی که باید اجرا شود، بنابر نیاز تفاوت می‌کند.

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۳/۰۹/۱۷ ۲۲:۵۸

سلام. توی EF 6 متد HasKey کجاست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۸ ۰:۱۰

در همان فضای نام System.Data.Entity.ModelConfiguration

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۳/۰۹/۱۸ ۰:۱۳

بله ممنون. من اشتباها بجای EntityTypeConfiguration از ComplexTypeConfiguration استفاده می‌کردم که توی اون HasKey وجود نداشت.

عنوان: EF Code First #7
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۰ ۱۱:۰۱:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

مدیریت روابط بین جداول در EF Code first به کمک Fluent API

EF Code first بجای اتلاف وقت شما با نوشتن فایل‌های XML تهیه نگاشت‌ها یا تنظیم آن‌ها با کد، رویه Convention over configuration را پیشنهاد می‌دهد. همین رویه، جهت مدیریت روابط بین جداول نیز برقرار است. روابط one-to-one، one-to-many، many-to-many و موارد دیگر را بدون یک سطر تنظیم اضافی، صرفا بر اساس یک سری قراردادهای توکار می‌تواند تشخیص داده و اعمال کند. عموما زمانی نیاز به تنظیمات دستی وجود خواهد داشت که قراردادهای توکار رعایت نشوند و یا برای مثال قرار است با یک بانک اطلاعاتی قدیمی از پیش موجود کار کنیم.

مفاهیمی به نام‌های Principal و Dependent

در EF Code first از یک سری واژه‌های خاص جهت بیان ابتدا و انتهای روابط استفاده شده است که عدم آشنایی با آن‌ها درک خطاهای حاصل را مشکل می‌کند:

الف) Principal : طرفی از رابطه است که ابتدا در بانک اطلاعاتی ذخیره خواهد شد.

ب) Dependent : طرفی از رابطه است که پس از ثبت Principal در بانک اطلاعاتی ذخیره می‌شود.

Principal می‌تواند بدون نیاز به Dependent وجود داشته باشد. وجود Dependent بدون Principal ممکن نیست زیرا ارتباط بین این دو توسط یک کلید خارجی تعریف می‌شود.

کدهای مثال مدیریت روابط بین جداول

در دنیای واقعی، همه‌ی مثال‌ها به مدل بلاگ و مطالب آن ختم نمی‌شوند. به همین جهت نیاز است یک مدل نسبتا پیچیده‌تر را در اینجا بررسی کنیم. در ادامه کدهای کامل مثال جاری را مشاهده خواهید کرد:

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Customer
    {
        public int Id { set; get; }
        public string FirstName { set; get; }
        public string LastName { set; get; }

        public virtual AlimentaryHabits AlimentaryHabits { set; get; }
        public virtual ICollection<CustomerAlias> Aliases { get; set; }
        public virtual ICollection<Role> Roles { get; set; }
        public virtual Address Address { get; set; }
    }
}
```

```
namespace EF_Sample35.Models
{
    public class CustomerAlias
    {
        public int Id { get; set; }
        public string Aka { get; set; }

        public virtual Customer Customer { get; set; }
    }
}
```

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Role
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<Customer> Customers { set; get; }
    }
}
```

```
namespace EF_Sample35.Models
{
    public class AlimentaryHabits
    {
        public int Id { get; set; }
        public bool LikesPasta { get; set; }
        public bool LikesPizza { get; set; }
        public int AverageDailyCalories { get; set; }

        public virtual Customer Customer { get; set; }
    }
}
```

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Address
    {
        public int Id { set; get; }
        public  string City { set; get; }
        public  string StreetAddress { set; get; }
        public  string PostalCode { set; get; }

        public virtual ICollection<Customer> Customers { set; get; }
    }
}
```

همچنین **تعاریف نگاشت‌های برنامه** نیز مطابق کدهای زیر است:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample35.Models;

namespace EF_Sample35.Mappings
{
    public class CustomerAliasConfig : EntityTypeConfiguration<CustomerAlias>
    {
        public CustomerAliasConfig()
        {
            // one-to-many
            this.HasRequired(x => x.Customer)
                .WithMany(x => x.Aliases)
                .WillCascadeOnDelete();
```

```
        }
    }
}
```

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample35.Models;

namespace EF_Sample35.Mappings
{
    public class CustomerConfig : EntityTypeConfiguration<Customer>
    {
        public CustomerConfig()
        {
            // one-to-one
            this.HasOptional(x => x.AlimentaryHabits)
                .WithRequired(x => x.Customer)
                .WillCascadeOnDelete();

            // many-to-many
            this.HasMany(p => p.Roles)
                .WithMany(t => t.Customers)
                .Map(mc =>
                   {
                       mc.ToTable("RolesJoinCustomers");
                       mc.MapLeftKey("RoleId");
                       mc.MapRightKey("CustomerId");
                   });

            // many-to-one
            this.HasOptional(x => x.Address)
                .WithMany(x => x.Customers)
                .WillCascadeOnDelete();
        }
    }
}
```

به همراه **Context** زیر:

```
using System.Data.Entity;
using System.Data.Entity.Migrations;
using EF_Sample35.Mappings;
using EF_Sample35.Models;

namespace EF_Sample35.DataLayer
{
    public class Sample35Context : DbContext
    {
        public DbSet<AlimentaryHabits> AlimentaryHabits { set; get; }
        public DbSet<Customer> Customers { set; get; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new CustomerConfig());
            modelBuilder.Configurations.Add(new CustomerAliasConfig());

            base.OnModelCreating(modelBuilder);
        }
    }

    public class Configuration : DbMigrationsConfiguration<Sample35Context>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(Sample35Context context)
        {
            base.Seed(context);
```

```
        }
    }
}
```

که نهایتا منجر به تولید چنین **ساختاری در بانک اطلاعاتی** می‌گردد:

توضیحات کامل کدهای فوق:

تنظیمات روابط one-to-one و یا one-to-zero

زمانیکه رابطه‌ای 1..0 و یا 1..1 است، مطابق قراردادهای توکار EF Code first تنها کافی است یک navigation property را که بیانگر ارجاعی است به شیء دیگر، تعریف کنیم (در هر دو طرف رابطه).
برای مثال در مدل‌های فوق یک مشتری که در حین ثبت اطلاعات اصلی او، «ممکن است» اطلاعات جانبی دیگری (AlimentaryHabits) نیز از او تنها در طی یک رکورد، دریافت شود. قصد هم نداریم یک ComplexType را تعریف کنیم. نیاز است جدول AlimentaryHabits جداگانه وجود داشته باشد.

```
namespace EF_Sample35.Models
{
    public class Customer
    {
        // ...
        public virtual AlimentaryHabits AlimentaryHabits { set; get; }
    }
}
```

```
namespace EF_Sample35.Models
{
    public class AlimentaryHabits
    {
        // ...
        public virtual Customer Customer { get; set; }
    }
}
```

در اینجا خواص virtual تعریف شده در دو طرف رابطه، به EF خواهد گفت که رابطه‌ای، 1:1 برقرار است. در این حالت اگر برنامه را اجرا کنیم، به خطای زیر برخواهیم خورد:

```
Unable to determine the principal end of an association between
the types 'EF_Sample35.Models.Customer' and 'EF_Sample35.Models.AlimentaryHabits'.
The principal end of this association must be explicitly configured using either
the relationship fluent API or data annotations.
```

EF تشخیص داده است که رابطه 1:1 برقرار است؛ اما با قاطعیت نمی‌تواند طرف Principal را تعیین کند. بنابراین باید اندکی به او کمک کرد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample35.Models;

namespace EF_Sample35.Mappings
{
    public class CustomerConfig : EntityTypeConfiguration<Customer>
    {
        public CustomerConfig()
        {
            // one-to-one
            this.HasOptional(x => x.AlimentaryHabits)
                .WithRequired(x => x.Customer)
                .WillCascadeOnDelete();
        }
    }
}
```

همانطور که ملاحظه می‌کنید در اینجا توسط متد WithRequired طرف Principal و توسط متد HasOptional، طرف Dependent تعیین شده است. به این ترتیب EF می‌توان یک رابطه 1:1 را تشکیل دهید.
توسط متد WillCascadeOnDelete هم مشخص می‌کنیم که اگر Principal حذف شد، لطفا Dependent را به صورت خودکار حذف کن.

توضیحات ساختار جداول تشکیل شده:
هر دو جدول با همان خواص اصلی که در دو کلاس وجود دارند، تشکیل شده‌اند.

فیلد Id جدول AlimentaryHabits اینبار دیگر Identity نیست. اگر به تعریف قید FK_AlimentaryHabits_Customers_Id دقت کنیم، در اینجا مشخص است که فیلد Id جدول AlimentaryHabits، به فیلد Id جدول مشتری‌ها متصل شده است (یعنی در آن واحد هم primary key است و هم foreign key). به همین جهت به این روش one-to-one association with shared primary key هم گفته می‌شود (کلید اصلی جدول مشتری با جدول AlimentaryHabits به اشتراک گذاشته شده است).

تنظیمات روابط one-to-many

برای مثال همان مشتری فوق را درنظر بگیرید که دارای تعدادی نام مستعار است:

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Customer
    {
        // ...
        public virtual ICollection<CustomerAlias> Aliases { get; set; }
    }
}
```

```
namespace EF_Sample35.Models
{
    public class CustomerAlias
    {
        // ...
        public virtual Customer Customer { get; set; }
    }
}
```

همین میزان تنظیم کفایت می‌کند و نیازی به استفاده از Fluent API برای معرفی روابط نیست.
در طرف Principal، یک مجموعه یا لیستی از Dependent وجود دارد. در Dependent هم یک navigation property معرف طرف Principal اضافه شده است.
جدول CustomerAlias اضافه شده، توسط یک کلید خارجی به جدول مشتری مرتبط می‌شود.

سؤال: اگر در اینجا نیز بخواهیم CascadeOnDelete را اعمال کنیم، چه باید کرد؟
پاسخ: جهت سفارشی سازی نحوه تعاریف روابط حتما نیاز به استفاده از Fluent API به نحو زیر می‌باشد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample35.Models;

namespace EF_Sample35.Mappings
{
    public class CustomerAliasConfig : EntityTypeConfiguration<CustomerAlias>
    {
        public CustomerAliasConfig()
        {
            // one-to-many
            this.HasRequired(x => x.Customer)
                .WithMany(x => x.Aliases)
                .WillCascadeOnDelete();
        }
    }
}
```

اینکار را باید در کلاس تنظیمات CustomerAlias انجام داد تا بتوان Principal را توسط متد HasRequired به Customer و سپس

dependent را به کمک متد WithMany مشخص کرد. در ادامه می‌توان متد WillCascadeOnDelete یا هر تنظیم سفارشی دیگری را نیز اعمال نمود.
متد HasRequired سبب خواهد شد فیلد Customer_Id، به صورت not null در سمت بانک اطلاعاتی تعریف شود؛ متد HasOptional عکس آن است.

تنظیمات روابط many-to-many

برای تنظیم روابط many-to-many تنها کافی است دو سر رابطه ارجاعاتی را به یکدیگر توسط یک لیست یا مجموعه داشته باشند:

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Role
    {
        // ...
        public virtual ICollection<Customer> Customers { set; get; }
    }
}
```

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Customer
    {
        // ...
        public virtual ICollection<Role> Roles { get; set; }
    }
}
```

همانطور که مشاهده می‌کنید، یک مشتری می‌تواند چندین نقش داشته باشد و هر نقش می‌تواند به چندین مشتری منتسب شود.
اگر برنامه را به این ترتیب اجرا کنیم، به صورت خودکار یک رابطه many-to-many تشکیل خواهد شد (بدون نیاز به تنظیمات نگاشت‌های آن). نکته جالب آن تشکیل خودکار جدول ارتباط دهنده واسط یا اصطلاحا join-table می‌باشد:

```
CREATE TABLE [dbo].[RolesJoinCustomers](
 [RoleId] [int] NOT NULL,
 [CustomerId] [int] NOT NULL,
)
```

سؤال: نام‌های خودکار استفاده شده را می‌خواهیم تغییر دهیم. چکار باید کرد؟
پاسخ: اگر بانک اطلاعاتی برای بار اول است که توسط این روش تولید می‌شود شاید این پیش فرض‌ها اهمیتی نداشته باشد و نسبتا هم مناسب هستند. اما اگر قرار باشد از یک بانک اطلاعاتی موجود که امکان تغییر نام فیلدها و جداول آن وجود ندارد استفاده کنیم، نیاز به سفارشی سازی تعاریف نگاشت‌ها به کمک Fluent API خواهیم داشت:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample35.Models;

namespace EF_Sample35.Mappings
{
    public class CustomerConfig : EntityTypeConfiguration<Customer>
    {
        public CustomerConfig()
        {
```

```
            // many-to-many
            this.HasMany(p => p.Roles)
                .WithMany(t => t.Customers)
                .Map(mc =>
                   {
                       mc.ToTable("RolesJoinCustomers");
                       mc.MapLeftKey("RoleId");
                       mc.MapRightKey("CustomerId");
                   });
        }
    }
}
```

تنظیمات روابط many-to-one

در تکمیل مدل‌های مثال جاری، به دو کلاس زیر خواهیم رسید. در اینجا تنها در کلاس مشتری است که ارجاعی به کلاس آدرس او وجود دارد. در کلاس آدرس، یک navigation property همانند حالت 1:1 تعریف نشده است:

```
namespace EF_Sample35.Models
{
    public class Address
    {
        public int Id { set; get; }
        public  string City { set; get; }
        public  string StreetAddress { set; get; }
        public  string PostalCode { set; get; }
    }
}
```

```
using System.Collections.Generic;

namespace EF_Sample35.Models
{
    public class Customer
    {
       // …
        public virtual Address Address { get; set; }
    }
}
```

این رابطه توسط EF Code first به صورت خودکار به یک رابطه many-to-one تفسیر خواهد شد و نیازی به تنظیمات خاصی ندارد. زمانیکه جداول برنامه تشکیل شوند، جدول Addresses موجودیتی مستقل خواهد داشت و جدول مشتری با یک فیلد به نام Address_Id به جدول آدرس‌ها متصل می‌گردد. این فیلد نال پذیر است؛ به عبارتی ذکر آدرس مشتری الزامی نیست.
اگر نیاز بود این تعاریف نیز توسط Fluent API سفارشی شوند، باید خاصیت public virtual ICollection<Customer> Customers به کلاس Address نیز اضافه شود تا بتوان رابطه زیر را توسط کدهای برنامه تعریف کرد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample35.Models;

namespace EF_Sample35.Mappings
{
    public class CustomerConfig : EntityTypeConfiguration<Customer>
    {
        public CustomerConfig()
        {
            // many-to-one
            this.HasOptional(x => x.Address)
                .WithMany(x => x.Customers)
```

```
                .WillCascadeOnDelete();
        }
    }
}
```

متد HasOptional سبب می‌شود تا فیلد Address_Id اضافه شده به جدول مشتری‌ها، null پذیر شود.

نظرات خوانندگان

نویسنده: MehdiPayervand
تاریخ: ۱۳۹۱/۰۲/۲۰ ۱۶:۵۳:۵۲

سلام جناب مهندس، تشکر از لطفتتون، اگه ممکنه کدهای این سری رو هم توی کدپلس آپ کنین. ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۰ ۲۰:۰۵:۳۲

سلام، از اینجا می‌تونید دریافت کنید: (^)

نویسنده: amir
تاریخ: ۱۳۹۱/۰۲/۲۰ ۲۲:۳۹:۰۴

سلام خیلی ممنون بابت مطالب بسیار مفیدتون. اگر ممکنه نحوه ارتباط یک Sql View رو با EF Code First هم توضیح بدید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۱ ۰۰:۳۶:۵۳

استفاده از View نکته خاص و اضافه‌تری نداره؛ از این لحاظ که عموما به Viewها به شکل یک جدول فقط خواندنی نگاه می‌شود. بنابراین یک کلاس تعریف کنید حاوی فیلدهای همان View. بعد هم یک data annotations برای مثال Table را بالای این کلاس قرار دهید (اگر نیاز بود از نام خاصی که جزو اصول نامگذاری کلاس‌ها در سی شارپ نیست استفاده کند).

نویسنده: amir
تاریخ: ۱۳۹۱/۰۲/۲۱ ۱۳:۲۰:۵۶

سلام دیتابیس من به صورت خلاصه از دو تا جدول تشکیل شده یکی مشتریان (Customers) و یکی هم دریافت و پرداخت ها (Payments). حالا میخوام وقتی یک مشتری حذف میشه کل دریافت پرداخت هاش هم حذف بشه. از کد زیر استفاده کردم ولی نمیدونم چرا تو دیتابیس یه فیلد اضافی Customer_ID رو به جدول دریافت پرداخت هام اضافه میکنه در حالی که من خودم یه فیلد CustomerID گذاشته بودم!

```
()modelBuilder.Entity
(HasRequired(s => s.Customer.
()WithMany.
;()WillCascadeOnDelete.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۱ ۱۳:۳۵:۱۲

لطف تعریف دقیق کلاس‌های مدلتون رو اینجا قرار بدید و لینک بدید: pastebin.com

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۱ ۱۳:۵۴:۴۷

ضمنا در قسمت هشتم (قسمت بعدی)، در مورد Self Referencing Entity بحث شده. نگاشت شما خیلی شبیه به آن است. در کل نیاز است تعریف دقیق مدل‌ها و روابط تعریف شده رو دید.

نویسنده: amir
تاریخ: ۱۳۹۱/۰۲/۲۱ ۱۵:۱۵:۳۸

اینقدر باهاش ور رفتم تا درست شد (با استفاده از متد HasForeignKey). الان مشکلی که دارم در مورد Cascade Update هستش.
اگر اینم تو قسمت های بعدی توضیح بدید عالی میشه.

فقط یه خواهش دیگه که داشتم (البته مرتبط با این موضوع نیست)، اینکه اگر وقت کردید در مورد WPF و MVVM مطالب بیشتری بذارید چون این دو موضوع الان خیلی طرفدار دارن ولی یه خورده سخت هستن!.

من روزی چند بار به سایت شما میام و از مطالب مفیدتون استفاده میکنم. واقعاً ازتون ممنونم.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۷:۳۷

د صورتی که یک رابطه many-to - many داشته باشیم و ابتدا مثل رابطه Role , Customer ذخیره سازی ارتباط این جداول به چه صورت است ؟
در حالت عادی برای هر ذخیره سازی هم اطلاعات Role و هم Customer ذخیره می‌گردد .
آیا باید یه Entity واسط هم در نظر بگیریم و پس از ثبت اطلاعات مثلا Customer با ID مربوط به Role از طریق Entity واسط اطلاعات ارتباط این 2 Entity ذخیره شود ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۹:۵۸

خیر. مدیریت این جدول واسط کاملا خودکار است.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۱/۰۵/۲۲ ۱۵:۳

با تشکر از پاسخ دهی شما به سوالات؛ موقع Create درست اعمال می‌شود، اما هنگام Edit جدول واسط به روز نمی‌گردد.
مثلا برای دو جدول User , Role که نقش‌های یک کاربر بوسیله یک []string به اکشن Edit پاس داده شده
کد مربوطه به صورت زیر می‌باشد

```
[HttpPost]
        public ActionResult Edit(User user, string[] tags)
        {
            if (ModelState.IsValid)
            {
                List<Role> roles = new List<Role>();
                foreach (var item in tags)
                {
                    Role role = db.Roles.Find(long.Parse(item));
                    roles.Add(role);
                }
                user.Roles = roles;

                db.Entry(user).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(user);
        }
```

اما جدول واسط در این قسمت به روز نمی‌شود . متاسفانه چیز خاصی در این رابطه پیدا نکردم و مجددا مزاحم شما شدم .

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۲ ۱۶:۳۴

قبل از ویرایش این سطر را اضافه کنید

```
if(user.Roles != null && user.Roles.Any())
   user.Roles.Clear();
```

همچنین اگر itemهای دریافتی primary key هستند می‌تونید از متد attach بجای مراجعه به بانک اطلاعاتی، استفاده کنید:

```
ctx.Roles.Attach(new Role { Id = item });
```

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۱/۰۵/۲۲ ۲۳:۲۸

سلام
در action مربوط به Edit که در بالا امده است فیلد Roles برابر null می‌باشد . دلیل این رو نمی‌دانم شاید مشکل از bind فرم هست اما entity که در متد Get مقدار دهی میشه Roles مقدار دارد .
در مورد attach میشه توضیح بدید . البته با این هم نتونستم کاری انجام بدم .
در اخر ناچار شدم ابتدا User رو یک بار find کنم و سپس با مقدار مدل مقدار دهی کنم به نظر خودم که کار درستی نیست .
خوشحال میشم که با راه حل اساسی آشنا بشم .
امکانش هست .

کدها رو به این صورت تغییر دادم که مشکلم برطرف شد ولی فکر میکنم که بدون find هم باید راه حلی باشه که بلد نیستم

```
[HttpPost]
        public ActionResult Edit(User user, string[] tags)
        {
            User Currentuser = db.Users.Find(user.Id);

            Currentuser.FirstName = user.FirstName;
            Currentuser.LastName = user.LastName;
            if (ModelState.IsValid)
            {
                List<Role> roles = new List<Role>();
                if (Currentuser.Roles != null && Currentuser.Roles.Any())
                    Currentuser.Roles.Clear();
                foreach (var item in tags)
                {
                    Role role = db.Roles.Find(long.Parse(item));
                    roles.Add(role);
                }
                Currentuser.Roles = roles;

                db.Entry(Currentuser).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(Currentuser);
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۳ ۰:۳۷

اگر با attach جواب نگرفتید، لیست نقش‌ها رو با یک رفت و برگشت هم می‌توان بدست آورد:

```
var listOfActualRoles = db.Roles.Where(x => tags.Contains(x.Id)).ToList();
```

نویسنده: alireza
تاریخ: ۱۳۹۱/۱۰/۲۱ ۹:۰

سلام وحید جان ممنون از این همه لطف

من یک پروژه را با CodeFirst شروع کردم اما یه جایی اشتباه کردم فکر کنم اشتباهم توی یکی از Mappingها باشه. اگه لطف کنید ببینید مشکل چیه.بدون استفاده از Mapping مشکلی نیست و دیتا بیس با روابطی که میخوام ایجاد میشه اما وقتی از Mapping استفاده میکنم با این خطا مواجه میشم:

```
{"Sequence contains more than one matching element"}
```

چندتا کلاس‌ها به شکل زیر هست:

```
public class Driver
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string NationalCode { get; set; }
        public string CellPhone { get; set; }
        public string LicenseNumber { get; set; }
        public bool IsDriverAssistance { get; set; }

        [InverseProperty("Driver")]
        public virtual ICollection<Transference> Transferences { get; set; }
        [InverseProperty("DriverAssistance")]
        public virtual ICollection<Transference> TransferencesForAssistance { get; set; }
        [InverseProperty("Driver")]
        public virtual ICollection<Tanker> Tankers { get; set; }
        [InverseProperty("DriverAssistance")]
        public virtual ICollection<Tanker> TankersForAssistance { get; set; }

    }
```

```
public class Transference
    {
        public string Id { get; set; }
        public DateTime Date { get; set; }
        public Int16 Lytrazh { get; set; }
        public bool IsEMS { get; set; }
        public DateTime LoadingDate { get; set; }
        public DateTime DeliveryDate { get; set; }
        [InverseProperty("Transferences")]
        public virtual Driver Driver { get; set; }
        [InverseProperty("TransferencesForAssistance")]
        public virtual Driver DriverAssistance { get; set; }
        public virtual TypeOfTanker TypesOfTanker { get; set; }
        public virtual Tanker Tanker { get; set; }
        public virtual Consumer Consumer { get; set; }

    }
```

فکر کنم مشکل از این کلاس زیر باشه:

```
public class TransferenceConfig : EntityTypeConfiguration<Transference>
    {
        public TransferenceConfig()
        {
            // one-to-many
            this.HasRequired(x => x.Consumer)
                .WithMany(x => x.Transferences);

                // one-to-many
            this.HasRequired(x => x.TypesOfTanker)
                .WithMany(x => x.Transferences);

            // one-to-many
            this.HasRequired(x => x.Tanker)
                .WithMany(x => x.Transferences);
```

```
            // one-to-many
            this.HasRequired(x => x.Driver)
                .WithMany(x => x.Transferences);

            // one-to-many
            this.HasRequired(x => x.DriverAssistance)
                .WithMany(x => x.Transferences);

        }
    }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۱ ۱۸:۲۳

- مشکل کلاس کانفیگ فوق در این است که از یک طرف InverseProperty تعریف کردید، از طرف دیگر در حالت تنظیمات Fluent، این مورد رعایت نشده. مثلا DriverAssistance باید به TransferencesForAssistance (مطابق InverseProperty تعریف شده) مرتبط می‌شد و الی آخر (الان همگی به یک مورد مرتبط شدن).
- در کل نیازی به کلاس کانفیگ فوق ندارید. حذفش کنید. EF می‌تونه روابط one-to-many رو بدون کانفیگ خاصی تشخیص بده. علت وجود قسمت هفتم، اعمال یک سری تنظیمات اضافه‌تر است نسبت به تنظیمات پیش فرض. مثلا اگر از نام‌های پیش فرض خرسند نیستید، اینجا می‌تونید توسط Fluent API خیلی از این موارد رو سفارشی سازی کنید و تغییر بدید. البته شرطش هم این است که از ICollection برای معرفی موارد one-to-many استفاده کنید (که اینکار در کلاس Driver انجام شده، همچنین یک سر دیگر آن به صورت virtual در کلاس مقابل وجود دارد. به علاوه مطلب نحوه تعریف صحیح کلیدهای خارجی را هم اضافه کنید تا طراحی بهتری داشته باشید).

نویسنده: علیرضا پایدار
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۱:۲۷

ممنون از پاسخ.
درست می‌فرمایید من میتونستم کلاس کانفیگ را حذف کنم اما میخواستم با کانفیگ تست کنم که نتونستم.
البته تنظیمات اضافه هم قراره وقتی این مشکل رفع شد اضافه نمایم مثل تنظیم حداکثر طول فیلد،یا عنوان مناسب برای کلید خارجی و NOT NULL از این جور تنظیمات که خودتون توی مطالب قبلی ارائه نمودید.
فرمودید:"- مشکل کلاس کانفیگ فوق در این است که از یک طرف InverseProperty تعریف کردید، از طرف دیگر در حالت تنظیمات Fluent، این مورد رعایت نشده. مثلا DriverAssistance باید به TransferencesForAssistance (مطابق InverseProperty تعریف شده) مرتبط می‌شد و الی آخر (الان همگی به یک مورد مرتبط شدن). "
منظور اینه به شکل زیر تبدیل بشه:

```
public class TransferenceConfig : EntityTypeConfiguration<Transference>
   {
       public TransferenceConfig()
       {
           // one-to-many
           this.HasRequired(x => x.Consumer)
               .WithMany(x => x.Transferences);

               // one-to-many
           this.HasRequired(x => x.TypesOfTanker)
               .WithMany(x => x.Transferences);

           // one-to-many
           this.HasRequired(x => x.Tanker)
               .WithMany(x => x.Transferences);

           // one-to-many
           this.HasRequired(x => x.Driver)
               .WithMany(x => x.Transferences);
 //-------------------------------------------------------------------------------------
           // one-to-many
           this.HasRequired(x => x.DriverAssistance)
               .WithMany(x => x.TransferencesForAssistance);
//-------------------------------------------------------------------------------------

       }
   }
```

بخشی که بین 2 تا خط نقطه چین قرار گرفته.بله؟
بجای اینکه از InverseProperty --> Attribute بشه آیا معادلی توی Fluent API داره؟

بازم ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۱:۳۳

زمانیکه از Fluent API استفاده می‌کنید نیازی به ذکر Attributes ندارید؛ چون طرفین یک ارتباط و ریز مشخصات آن‌ها با کدنویسی دقیقا (و بدون هیچگونه قرارداد خاصی) مشخص می‌شوند. یک نمونه رو در مثال شما عنوان کردم، مثلا DriverAssistance از یک کلاس به TransferencesForAssistance کلاس دیگر مرتبط شده. به این ترتیب نیازی به ذکر ویژگی خاصی برای مشخص سازی مجدد آن نیست (چون دیگر جای حدس و گمان و پیش‌فرضی باقی نمی‌ماند. صریحا تنظیمات مشخص شده‌اند). برای سایر موارد هم باید به همین ترتیب عمل کنید.

نویسنده: مسعود
تاریخ: ۱۳۹۱/۱۱/۰۲ ۹:۲۱

سلام
مدیریت روابط n-n در کلاسهای Poco به چه صورت است؟ من یک برنامه tier-2 دارم و برای Entity هایم state گرفتم و سمت کلاینت وضعیت entityها رو مدیریت میکنم و سمت سرور اونا رو روی DBContext اعمال میکنم.
دو کلاس Role و Permission دارم که باهم رابطه n-n دارند؛ حال اگه یک Role چند Permission داشته باشه و بخوام یکی از اونا رو حذف کنم و یا آپدیت کنم بهم پیغام خطا میده(An error occurred while saving entities that do not expose foreign key properties for their relationships. The EntityEntries property will return null because a single e ...) و داخل inner exception هم مینویسه Violation of PRIMARY KEY constraint 'PK_dbo.SecurityRolePermission'. Cannot insert() duplicate key in object 'dbo.SecurityRolePermission'
یعنی در حالتهای آپدیت و حذف هم میخواد insert انجام بده (با ef profiler هم چک کردم)البته فکر کنم تا حدودی معلوم باشه چرا اینکار رو میکنه؛ چون مدیریتی روی state اون کلاس واسط(RolePermission) انجام نمیشه، ممکنه بگید چطوری این مشکل رفع میشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۲ ۱۳:۳۰

در مطلب « کار با کلیدهای اصلی و خارجی در EF Code first » توضیح دادم چرا و در چه صورتی رکورد تکراری تولید میشه.

نویسنده: سارا زرمهر
تاریخ: ۱۳۹۱/۱۱/۰۳ ۱۲:۱

سلام
توی کلاس Context ما کدام موجودیت‌ها در Dbset میگزاریم؟ در این جا چرا شما فقط AlimentaryHabits وCustomer رو توی Context گذاشتید؟

```
namespace EF_Sample35.DataLayer
{
 public class Sample35Context : DbContext
 {
 public DbSet<AlimentaryHabits> AlimentaryHabits { set; get; }
 public DbSet<Customer> Customers { set; get; }
 ...
 ...
 }
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۳ ۱۲:۱۷

- قرار دادن تمام کلاس‌های شرکت کننده در تشکیل جداول، حالت پیش فرض و معمول است. از این جهت که برای ثبت اطلاعات جداگانه در هر کدام نیاز به DbSet متناظر خواهد بود.
+ EF توانایی یافتن روابط و تشکیل جداول متناظر را بر اساس روابط بین کلاس‌ها، دارا است. اگر به تصویر اسکیمای حاصل دقت کنید این مساله مشهود است.
- در کل در این «مثال» ذکر دو مورد جهت برآوردن مقصود توضیحات داده شده کافی بوده.

نویسنده: حسین غلامی
تاریخ: ۱۳۹۱/۱۱/۱۹ ۱۷:۴۲

سلام؛
در SQL SERVER قسمتی داریم به نام INSERT And UPDATE Specification که دو گزینه‌ی Delete Rule و Update Rule برای ارتباطها وجود دارند.
در اینجا شما WillCascadeOnDelete رو بیان کردید ولی من هر چه دنبال متدی متناظر با Update Rule میگردم وجود نداره.
یه چیزی مثل : WillCascadeOnUpdate
آیا در EF ساپورت نمیشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۱۹ ۱۸:۱۷

خیر. ON UPDATE CASCADE عموما به معنای به روز رسانی primary key است که در جداول دیگر ارجاع دارد و از دیدگاه EF یک کلید اصلی، read only است.
به نظر من کار درستی انجام دادن. زمانیکه نیاز به به‌روز رسانی primary key دارید یعنی طراحی بانک اطلاعاتی شما یا نرمال نیست یا مشکل داره.
البته می‌تونید رویه ذخیره شده درست کنید برای اینکارها و دستی مسایل رو به زور اعمال کنید ولی به صورت پیش فرض خارج از سیستم Tracking آن که به صورت خودکار اطلاعات اشیاء مرتبط را به روز می‌کند، Cascade Update دیگری وجود ندارد.

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۱۱/۲۱ ۱۸:۳۲

سلام

من هم با نظر شما در مورد بروز رسانی کلید اصلی موافقم.

اما واقعیت متفاوت تره.

من تو یه نرم افزار شماره پرسنلی که نوعش هم عدد صحیح بود کردم کلید اصلی.

بعد از 6 ماه متوجه شدن که شماره پرسنلی یکی از کارمندان رو اشتباه وارد کردن!

البته من از ef database first استفاده میکردم که اونجا هم مثل بقیه ormها اجازه update کردن کلید اصلی رو نمیده.

نویسنده: سعید

تاریخ: ۱۳۹۱/۱۱/۲۱ ۱۹:۴۹

همه این‌ها به طراحی بر می‌گرده. می‌تونستید شماره پرسنلی رو به صورت unique تعریف کنید و کلید اصلی رو یک فیلد auto increment، تا مشکل نداشته باشید. مثل آدرس ایمیل کاربران در یک بانک اطلاعاتی. این آدرس باید منحصربفرد باشه به ازای یک کاربر در سایت. یک کاربر باز هم می‌تونه از این نوع فیلدهای unique داشته باشه در یک جدول. مثل نام کاربر و یا مثل کد ملی.

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۱۱/۲۱ ۲۰:۱۷

سعید جان میدونم که اینکار میشه- مطمئنا شماره ملیشون رو uniqe کردم!

ما برای انتخاب کلید اصلی دو حالت داریم -

1- استفاده از کلیدهای طبیعی مثل شماره پرسنلی

2- استفاده از کلیدهای جانشین مثل یک فیلد identity - این حالت موقعی استفاده میشه که کلید طبیعی نداشته باشیم

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۲۱ ۲۰:۲۸

زمانیکه شماره پرسنلی رو تبدیل به کلید اصلی می‌کنید یعنی تکرار داده در جداول مختلفی که به آن ارتباط پیدا می‌کنند و نیاز به آپدیت تمام جداول مرتبط با تغییر حتی یک نقطه در آن. به نظر شما این نوع دیتابیس نرمال شده است؟

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۱۱/۲۱ ۲۱:۱۴

تکرار داده در جداول مختلفی که به آن ارتباط پیدا می‌کنند ??

تکرار چه داده ای؟

سعید جان مطمئنا کلید شما تو جداول برای رابطه استفاده میشه حالا چه طبیعی باشه چه جانشین.

با فرض اینکه با 10 جدول ارتباط داشته باشه با n تعداد رکورد . حالا 6 ماهی یک بار این اتفاق چه ایرادی داره؟

بگذریم . موفق باشید

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۲۱ ۲۲:۲۵

- تکرار داده‌ای که قرار هست ویرایش بشه. فرض کن که یک لینک از کارمند مورد نظر ایجاد شده با شماره پرسنلی که pk است. الان این لینک یاد داشت شده دیگه کار نمی‌کنه. این یک مثال ساده است. یا به روز رسانی تمام رکوردهای یک جدول با یک update که تریگر هم روش تعریف شده ممکنه مشکل ساز بشه یا اصلا کار نکنه.

- در ef هر موجودیت نیاز به یک کلید منحصربفرد داره برای شناسایی و از این کلید در سیستم ردیابی خودش استفاده می‌کنه. اگر این کلید قرار باشه تغییر کنه، ef نیاز داره تا یک وهله جدید رو خلق کنه و نمونه قبلی رو نابود. به این ترتیب عملکردش بهم می‌خوره و مجبور میشه رکورد جدید ثبت کنه بجای آپدیت قبلی. البته این کارها هم بدعتی نیست چون طراحی اون برای اساس

اصول domain driven design انجام شده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۲ ۱۵:۱

اگر نیاز به یک مثال تکمیلی مشابه دیگر دارید که اکثر روابط در آن مطرح شده باشد به مطلب زیر مراجعه کنید:
Creating a More Complex Data Model for an ASP.NET MVC Application

نویسنده: مسعود2
تاریخ: ۱۳۹۱/۱۱/۲۲ ۱۵:۶

ممنون ولی متأسفانه اونجا هم این رابطه (1..0 - 1..0) رو نگفته.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۲ ۱۷:۱۹

- در مثال قسمت هفتم، رابطه مشتری با عادت‌های آن «one-to-zero- **or** -one» است. یک مشتری رو میشه تعریف کرد، صرفنظر از عادت‌های ویژه او؛ البته نه برعکس.
- در مثال سایت MVC ، رابطه دپارتمان و ادمین آن هم «one-to-zero-or-one» است. به این نحو هم تعریف شده

```
modelBuilder.Entity<Department>()
    .HasOptional(x => x.Administrator);
```

بدون اضافه کردن قسمت WithRequired و با داشتن یک Id نال پذیر در کلاس دپارتمان:

```
public int? InstructorID { get; set; }
```

اگر به تعاریف آن دقت کنید، کلاس Instructor رابطه‌ای با دپارتمان نداره اما رابطه دپارتمان با ادمین آن که از نوع Instructor است نال پذیر تعریف شده تا رابطه «یک به صفر یا یک» میسر شود.
- رابطه کلاس‌های Instructor و OfficeAssignment مثال سایت MVC مانند مثال قسمت هفتم فوق است و متد WithRequired رو ذکر کرده.

نویسنده: مسعود2
تاریخ: ۱۳۹۱/۱۱/۲۲ ۱۸:۳۱

یعنی میفرمایید من چه رابطه ای بین این دو کلاس تعریف کنم تا رابطه 1..0-1..0 داشته باشم؟

```
public class Order
    {
        public int OrderId { get; set; }
        public virtual Quotation Quotation { get; set; }
    }
    public class Quotation
    {
        public int QuotationId { get; set; }
        public virtual Order Order { get; set; }
    }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۲ ۲۱:۴۴

با این تنظیمات

```
public class OrderMap : EntityTypeConfiguration<Order>
    {
        public OrderMap()
        {
            this.HasOptional(x => x.Quotation)
                .WithOptionalPrincipal()
                .Map(x => x.MapKey("OrderId"));
        }
    }

    public class QuotationMap : EntityTypeConfiguration<Quotation>
    {
        public QuotationMap()
        {
            this.HasOptional(x => x.Order)
                .WithOptionalPrincipal()
                .Map(x => x.MapKey("QuotationId"));
        }
    }
```

با این خروجی

نویسنده: مسعود2
تاریخ: ۱۳۹۱/۱۱/۲۳ ۲۲:۲۶

ممنون، جواب داد.

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۲۷ ۱۹:۱۳

چرا برای مشخص کردن Principal، همیشه در کلاس maping جدولی که در رابطه Dependent است باید اقدام کنیم . ایا قانون تو کار خود ef است ؟

نویسنده: میهمان
تاریخ: ۱۳۹۱/۱۱/۳۰ ۱۷:۵۹

با سلام
اگر ما این چنین ساختار را داشته باشیم مانند مثال ذکر شده در این پست ، در مورد قسمت زیر

```
public CustomerConfig()
        {
            // many-to-many
            this.HasMany(p => p.Roles)
                .WithMany(t => t.Customers)
                .Map(mc =>
                   {
                       mc.ToTable("RolesJoinCustomers");
                       mc.MapLeftKey("RoleId");
                       mc.MapRightKey("CustomerId");
                   });
        }
}
```

با حذف یک Customer و یا Role ، رکوردهای مربوط به آن در جدول RolesJoinCustomers به صورت خود کار حذف می‌شود؟ اگر نه ، راه حل چیست ؟ چون من برای حذف رکورد ، با error برخورد میکنم
مورد دوم : اگر یک customer به عنوان مثال با تعدادی customer دیگر در ارتباط باشد(با تعدادی customer نه با یک customer) در این صورت چگونه باید پیاده سازی شود ؟ در صورت یک Customer ، رکوردهای مربوط به آن به صورت خود کار حذف می‌شود؟ اگر نه ، راه حل چیست ؟
با تشکر فراوان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۴ ۹:۴۶

در مطلب « بررسی تفصیلی روابط many-to-many » بیشتر توضیح دادم.

نویسنده: نوید
تاریخ: ۱۳۹۱/۱۲/۰۷ ۱۰:۰

سلام؛ اگر ما جداولی داشته باشیم که ارتباط many-to-many داشته باشند مثل کالا و انبار و بخواهیم که یک اطلاعات اضافی ای در جدول واسط آنها ذخیره شود (مثلا تعداد کالاها در هر انبار) . در این شرایط باید جدول واسط را خودمان ایجاذ کنیم یا این کار را نیز میتوان توسط EF-Code First انجام داد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۷ ۱۰:۲۲

- در مطلب « بررسی تفصیلی روابط many-to-many » توضیح داده شده.
- تعداد ستون‌های جدول واسط خارج از دسترس شما است و توسط EF مدیریت می‌شود.
- تعداد کالا در هر انبار را یا کوئری بگیرید که روش انجام کار در انتهای مطلب یاد شده با مثال عنوان شده یا اینکه یک فیلد محاسبه شده در جدول انبار ایجاد کنید و نه در جدول واسط.
تعریف این فیلد اضافی در جدول واسط بی‌معنا است؛ چون در این جدول، در هر سطر، رابطه بین یک کالا و یک انبار ذخیره می‌شود. بنابراین نگهداری تعداد کل کالاهای یک انبار خارج از مسئولیت یک ردیف این جدول واسط است.

نویسنده: dream
تاریخ: ۱۳۹۱/۱۲/۱۸ ۱۲:۳

با سلام؛
من یه مشکلی داشتم ، می‌خواستم بدونم برای اینکه بتونیم بین سه جدول ارتباط many-to-many داشته باشیم چه جوری باید پیاده سازی کنیم،
مثلا کلاس "دانشجو"و"درس" و استاد" که ID هر کدوم توی یه جدول واسط قرار بگیره مثل ارتباط Many-to-many بین دو جدول
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۸ ۱۲:۳۴

- نیازی به رابطه many-to-many در تمام حالات مثال شما نیست.
رابطه دانشجو و درس چند به چند است.
رابطه درس و استاد چند به چند است.
نیازی نیست بین استاد و دانشجو رابطه مستقیمی تعریف شود.
نیاز به جدول چهارمی وجود دارد به نام «واحدهای اخذ شده» که در اینجا ID یک درس و یک استاد و یک دانشجو ثبت می‌شود.
رابطه‌ها هم یک به چند است. یک دانشجو چند واحد اخذ شده می‌تواند داشته باشد. یک استاد چند واحد ارائه شده را می‌تواند اداره کند.

+ مراجعه کنید به بحث بررسی تفصیلی رابطه چند به چند و کامنت‌های آن و لینکی که در آن به راه حل خاصی اشاره شده که کار جدول واسط را شبیه سازی می‌کند با دو رابطه یک به چند.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۱:۸

سلام.ممنون از توضیحات خوبتون.
من یک رابطه **many-to-one** بین جداول **Project** و **ProjectRow** دارم که به این صورت map شده:

```
HasOptional ( c => c.Project ).WithMany (c => c.ProjectRowCollection).HasForeignKey(c =>
c.ProjectID).WillCascadeOnDelete();
```

حالا وقتی میخوام یک ProjectRow رو حذف کنم به این صورت عمل میکنم:

```
ProjectRowCollection.Remove(ProjectRowItem);
```

اما وقتی یک ردیف پروژه رو حذف میکنم به جای اینکه ردیف رو از جدول حذف کنه فقط کلید خارجی رو NULL میکنه مگر اینکه مستقیم از خود ProjectRow ردیف رو حذف کنم.مشکل از کجا میتونه باشه؟!
ممنون از اینکه وقت گذاشتید و خوندید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۱:۳۷

- علت به Optional و Required بودن روابط بر می‌گردد. حالت Required یعنی فرزند، بدون والد نمی‌تواند وجود داشته باشد؛ برعکس حالت optional. بنابراین فقط در حالت Required حذف فرزندان در صورت حذف والد صورت خواهد گرفت.
- جزئیات بیشتر از زبان طراحان EF:
Deleting orphans with Entity Framework
+ طراحی پیش فرض است؛ مطابق مستندات MSDN:

```
If a foreign key on the dependent entity is nullable, Code First does not set cascade delete on the
relationship,
and when the principal is deleted the foreign key will be set to null.
```

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۲:۱۶

خیلی ممنونم.

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۴/۰۴ ۱۴:۳۴

راستش هنوز این مشکل برای من حل نشده،اینطور که من فهمیدم برای حذف یه فرزند باید مستقیما خودش رو حذف کنیم،با استفاده از حذف کردنش از Collection مربوط به والدش نمیشه،درسته؟ببینید نمیخوام والد رو حذف کنم،میخام از طریق لیستی که از فرزندها توی والد هست یکی از فرزندها رو حذف کنم.آیا امکانش هست؟

```
Project.ProjectRowCollection.Remove(ProjectRowItem);
```

یعنی من مجبورم تو مثال بالا مستقیم ProjectRowItem رو حذف کنم با استفاده از دستور بالا نمیشه؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۴ ۱۸:۴

- در هر حالتی، زمانیکه یک شیء کامل را به همراه Id تحت نظر سیستم ردیابی آن دارید (مثل ProjectRowItem)، ساده‌ترین راه حذف آن، ابتدا حذف آن از DbSet مرتبط، مانند ctx.ProjectRowItems.Remove و بعد فراخوانی SaveChanges است (جهت اعمال نهایی تغییرات به بانک اطلاعاتی).
- این شیء اگر تحت نظر سیستم ردیابی نباشد، فراخوانی متد Remove اثری نخواهد داشت. اطلاعات بیشتر : ^ و ^
- زمانیکه فقط یک شیء تحت ردیابی را از یک لیست حذف می‌کنید، این مورد فقط به معنای null کردن ID آن است؛ چون فرمان اصلی حذف خود شیء صادر نشده است. فقط دیگر علاقمند نیستید که این رابطه برقرار باشد.

نویسنده: سارا محمدی
تاریخ: ۱۳۹۲/۱۲/۰۱ ۱۰:۲

ممنون از مطلب خوبتان
من رابطه مشتری و آدرس را متوجه نشدم یعنی هر آدرس میتواند برای چند مشتری باشد و آیا کلید جدول آدرس کلید خارجی هم هست اگه ممکنه بیشتر توضیح بدید ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۱ ۱۰:۹

قسمت «تنظیمات روابط many-to-one» و تصویر ساختار ذیل عبارت «که نهایتا منجر به تولید چنین ساختاری در بانک اطلاعاتی می‌گردد » را مطالعه کنید.

نویسنده: حمید حسین وند
تاریخ: ۱۳۹۳/۰۱/۲۴ ۱۶:۴۹

نتونستم از روابط استفاده کنم. (یک به چند)

```
using System;
using System.Collections.Generic;
using System.Data.Entity.ModelConfiguration;
using System.Linq;
using System.Text;

namespace DomainClasses.EntityConfiguration
{
    public class UserConfig : EntityTypeConfiguration<User>
    {
        public UserConfig()
        {
            HasRequired(x=>x.Username).WithMany(x=>x.

        }
    }
}
```

Aggregate<>
All<>
Any<>
AsEnumerable<>
AsParallel
AsParallel<>
AsQueryable
AsQueryable<>
Average<>

علت اینکه میخوام از رابطه one to more استفاده کنم اینه که چندین جدول دیگه دارم که همه شون مرتبط به جدول User هستند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۴ ۱۷:۲

- در قسمت HasRequired که Username نباید تعریف شود. در اینجا یک سر دیگر رابطه باید معرفی گردد. همان روابط و کلاس‌هایی که به صورت virtual در کدها آمده.

- در متن ذکر کردم «همین میزان تنظیم کفایت می‌کند و نیازی به استفاده از Fluent API برای معرفی روابط نیست.»

برای بسیاری از تنظیمات EF Code first، اگر پیش فرض‌های آن‌را رعایت کنید، نیازی به هیچگونه تنظیم اضافه‌تری ندارید. مثلا برای رابطه one-to-many فقط کافی است در دو سر رابطه (نه فقط یک سر آن)، تنظیمات زیر را داشته باشید:

```
// یک سایت که چندین بلاگ دارد
public class Site
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual ICollection<Blog> Blogs { set; get; }
}

public class Blog
{
    public int Id { get; set; }
    public string Name { set; get; }

    [ForeignKey("SiteId")]
    public virtual Site Site { get; set; }
    public int SiteId { set; get; }
```

```
}
```

همین مقدار کافی است و پیش فرض‌ها را پوشش می‌دهد. تنظیمات Fluent برای زمانی است که می‌خواهید پیش‌فرض‌ها را بازنویسی کنید. مثلا نام جدول خودکار تشکیل شده توسط آن مدنظر شما نیست. یا حالت بسیار خاصی از روابط مانند مدل‌های خودارجاع دهنده باید تشکیل شود و در این حالت فقط حالت Fluent است که پاسخگوی یک چنین سناریوهایی است.

نویسنده: حمید حسین وند
تاریخ: ۱۳۹۳/۰۱/۲۴ ۱۷:۳۳

بله شما درست می‌فرمایید اما اگه بخوام وقتی رکوردی از جدول User حذف میشه رکوردهای مربوط به این یوزر در جداول دیگه (حدود 5 جدول) حذف بشه باید از WillCascadeOnDelete استفاده کنم.
می‌تونم به صورت زیر استفاده کنم اما میخوام ببینم چرا نمی‌تونم به صورت یک به چند استفاده کنم.

```
HasMany(x => x.Ads).WithRequired(x => x.User).WillCascadeOnDelete();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۴ ۱۷:۴۵

- در قسمت HasRequired که Username نباید تعریف شود. در اینجا یک سر دیگر رابطه باید معرفی گردد. همان روابط و کلاس‌هایی که به صورت virtual در کدها آمده. HasRequired با IsRequired متفاوت است.
+ حذف آبشاری به صورت پیش فرض فعال است (برای مواردی که کلید خارجی نال پذیر نیست). نیازی به فعال سازی دستی آن نیست.

نویسنده: حمید حسین وند
تاریخ: ۱۳۹۳/۰۱/۲۵ ۱۰:۷

سلام؛ کد زیر رو نوشتم طبق اون چیزی که توی این لینک که داده بودید اما وقتی حذف انجام میدم فقط کلیدهای خارجی رو نال میکنه و خود رکورد رو از جدول اصلی حذف میکنه.

```
modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۵ ۱۰:۳۹

- کدهایی که در ماخذ رسمی ذکر شدند ، برای حذف پیش فرض‌های EF هست. به صورت پیش فرض OneToManyCascadeDeleteConvention وجود دارد. اگر بخواهید در همه جا آن‌را حذف کنید، modelBuilder.Conventions.Remove را بر روی آن فراخوانی کنید. اگر نیاز است فقط در یک رابطه‌ی خاص این مورد حذف شود از متد WillCascadeOnDelete با پارامتر false استفاده کنید.
- همچنین مطابق این ماخذ: اگر کلید خارجی مدنظر نال پذیر باشد (مانند نوع‌های nullable صریح و یا string و امثال آن)، حذف آبشاری را اعمال نمی‌کند. فقط یک سر رابطه را نال کرده و آن‌را حذف می‌کند.

```
If a foreign key on the dependent entity
```

is nullable

```
, Code First does
not set cascade delete on the relationship, and when the principal is
deleted the foreign key will be set to null
```

```
If a foreign key on the dependent entity
```

is not nullable

```
, then Code First sets cascade delete on the relationship
```

نویسنده: جواد
تاریخ: ۱۳۹۳/۰۲/۰۵ ۱۰:۳۶

سلام؛ من یک جدول دارم که در اون کتاب‌های امانت گرفته شده را ثبت می‌کنم.یعنی این که در اون کلید خارجی کتاب و همچنین کلید خارجی اون عضو وجود دارد.حالا وقتی که می‌خواهم یک رکورد را از این جدول حذف بکنم به ارور زیر بر می‌خورم.
.The object cannot be deleted because it was not found in the ObjectStateManager

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۰۵ ۱۱:۳۲

یعنی سیستم tracking اطلاعی از وجود شیء شما ندارد (^ و ^).

نویسنده: علی
تاریخ: ۱۳۹۳/۰۷/۱۹ ۰:۴

سلام؛ من دوتا مدل دارم که ارتباط یک به یک دارن . یک کلاس ImageGallery و یک کلاس Attachment که تمامی فایل هامو تو دیتابیس قرار میدم . اینجا هم این کلاس نگه دارنده تصویر ImageGallery من هست. میخوام وقتی حذف میشه به صورت ImageGallery خودکار تصویرش هم حذف شه.
این کد fluent Api من هست

```
modelBuilder.Entity<ImageGallery>().HasOptional(T =>
T.Pic).WithOptionalPrincipal().WillCascadeOnDelete(true);
```

جالب اینجاست که وقتی دستی از تو دیتابیس رکورد ImageGallery حذف میشه تصویرش هم حذف میشه ولی با EF توی کنترلر که حذف میکنم عکسه حذف نمیشه. اینم کد سمت کنترلر :

```
db.ImageGalleries.Remove(db.ImageGalleries.Find(imageGallery_ID));
db.SaveChanges();
```

ممنون میشم اگه راهنمایی بفرمایید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۹ ۰:۵۷

کمی بالاتر توضیح دادم « ... طراحی پیش فرض است ... »

نویسنده: علیرضا م
تاریخ: ۱۳۹۳/۰۷/۲۷ ۸:۴۲

سلام؛ فرض کنیم چند جدول داشته باشیم (Principal) که هر کدام از آنها بتوانند با tblAddress ارتباط 1 / 0..1 داشته باشند. در برنامه قصد داریم با بررسی tblAddress تشخیص دهیم که کدام جدول به جدول فوق لینک شده است. EF توانایی اجرای چنین ساختاری را دارد؟ چندین بار سعی در اجرای چنین ساختاری کردم اما خطا اعلام میکند که یک سر رابطه باید * باشد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۷ ۱۶:۴۲

one-to-zero- **or** -one را در صفحه‌ی جاری جستجو کنید.

نویسنده: علیرضا م
تاریخ: ۱۳۹۳/۰۷/۲۹ ۱۰:۲۶

سلام
شاید من سوالم را بد مطرح کرده باشم
اگر رابطه یک به (یک یا صفر) باشد جدولی که (یک یا صفر) است کلید اصلی‌اش را از جدول دیگر رابطه میگیرد.
حال اگر چند جدول با یک جدول رابطه یک به (یک یا صفر) داشته باشند، جدولی که کلید اصلی آن Identity نیست ، کلید یکتای آن و کلید خارجی آن چطور تعریف میشود؟

مدل مد نظر را در اینجا قرار دادم.
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۹ ۱۱:۳۶

اگر توضیحات one-to-one association with shared primary key در متن فوق کافی نبوده، یک مثال کامل آن‌را در اینجا مطالعه کنید: « Associations in EF Code First CTP5: Part 2 – Shared Primary Key Associations »

نویسنده: حمیدرضا کبیری
تاریخ: ۱۳۹۳/۰۹/۱۰ ۱۸:۵

زمانی که از EF Database first استفاده میکنم ، جدولی مثل RolesJoinCustomers در مثال بالا جزء جداول قرار نمیگیرد ! چطور میتوانم از طریق database first به این جدول دستیابی پیدا کنم و ردیفی را اضافه کنم یا با جداول دیگه عمل join را انجام دهم ؟ لطفا راهنمایی کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۰ ۱۸:۳۵

در حالت Code first هم دسترسی مستقیمی به این جدول وجود ندارد. باید از خواص راهبری (navigation properties) برای افزودن ردیف‌ها و یا کار با ارتباطات مختلف استفاده کرد. اطلاعات بیشتر: « بررسی تفصیلی روابط many-to-many »

عنوان: EF Code First #8
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۱ ۱۳:۴۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

ادامه بحث بررسی جزئیات نحوه نگاشت کلاس‌ها به جداول، توسط EF Code first

استفاده از Viewهای SQL Server در EF Code first

از Viewها عموما همانند یک جدول فقط خواندنی استفاده می‌شود. بنابراین نحوه نگاشت اطلاعات یک کلاس به یک View دقیقا همانند نحوه نگاشت اطلاعات یک کلاس به یک جدول است و تمام نکاتی که تا کنون بررسی شدند، در اینجا نیز صادق است. اما ...
الف) بر اساس تنظیمات توکار EF Code first، نام مفرد کلاس‌ها، حین نگاشت به جداول، تبدیل به اسم جمع می‌شوند. بنابراین اگر View ما در سمت بانک اطلاعاتی چنین تعریفی دارد:

```
Create VIEW EmployeesView
AS
SELECT id,
       FirstName
FROM   Employees
```

در سمت کدهای برنامه نیاز است به این شکل تعریف شود:

```
using System.ComponentModel.DataAnnotations;

namespace EF_Sample04.Models
{
    [Table("EmployeesView")]
    public class EmployeesView
    {
        public int Id { set; get; }
        public string FirstName { set; get; }
    }
}
```

در اینجا به کمک ویژگی Table، نام دقیق این View را در بانک اطلاعاتی مشخص کرده‌ایم. به این ترتیب تنظیمات توکار EF بازنویسی خواهد شد و دیگر به دنبال EmployeesViews نخواهد گشت؛ یا جدول متناظر با آن‌را به صورت خودکار ایجاد نخواهد کرد.
ب) View شما نیاز است دارای یک فیلد Primary key نیز باشد.
ج) اگر از مهاجرت خودکار توسط MigrateDatabaseToLatestVersion استفاده کنیم، پیغام خطای زیر را دریافت خواهیم کرد:

```
There is already an object named 'EmployeesView' in the database.
```

علت این است که هنوز جدول dbo.__MigrationHistory از وجود آن مطلع نشده است، زیرا یک View، خارج از برنامه و در سمت بانک اطلاعاتی اضافه می‌شود.
برای حل این مشکل می‌توان همانطور که در قسمت‌های قبل نیز عنوان شد، EF را طوری تنظیم کرد تا کاری با بانک اطلاعاتی نداشته باشد:

```
Database.SetInitializer<Sample04Context>(null);
```

به این ترتیب EmployeesView در همین لحظه قابل استفاده است.
و یا به حالت امن مهاجرت دستی سوئیچ کنید:

```
Add-Migration Init -IgnoreChanges
Update-Database
```

پارامتر IgnoreChanges سبب می‌شود تا متدهای Up و Down کلاس مهاجرت تولید شده، خالی باشد. یعنی زمانیکه دستور Update-Database انجام می‌شود، نه Viewایی دراپ خواهد شد و نه جدول اضافه‌ای ایجاد می‌گردد. فقط جدول dbo.__MigrationHistory به روز می‌شود که هدف اصلی ما نیز همین است.
همچنین در این حالت کنترل کاملی بر روی کلاس‌های Up و Down وجود دارد. می‌توان CreateTable اضافی را به سادگی از این کلاس‌ها حذف کرد.

ضمن اینکه باید دقت داشت یکی از اهداف کار با ORMs، فراهم شدن امکان استفاده از بانک‌های اطلاعاتی مختلف، بدون اعمال تغییری در کدهای برنامه می‌باشد (فقط تغییر کانکشن استرینگ، به علاوه تعیین Provider جدید، باید جهت این مهاجرت کفایت کند). بنابراین اگر از View استفاده می‌کنید، این برنامه به SQL Server گره خواهد خورد و دیگر از سایر بانک‌های اطلاعاتی که از این مفهوم پشتیبانی نمی‌کنند، نمی‌توان به سادگی استفاده کرد.

استفاده از فیلدهای XML اس کیوال سرور

در حال حاضر پشتیبانی توکاری توسط EF Code first از فیلدهای ویژه XML اس کیوال سرور وجود ندارد؛ اما استفاده از آن‌ها با رعایت چند نکته ساده، به نحو زیر است:

```
using System.ComponentModel.DataAnnotations;
using System.Xml.Linq;

namespace EF_Sample04.Models
{
    public class MyXMLTable
    {
        public int Id { get; set; }

        [Column(TypeName = "xml")]
        public string XmlValue { get; set; }

        [NotMapped]
        public XElement XmlValueWrapper
        {
            get { return XElement.Parse(XmlValue); }
            set { XmlValue = value.ToString(); }
        }
    }
}
```

در اینجا توسط TypeName ویژگی Column، نوع توکار xml مشخص شده است. این فیلد در طرف کدهای کلاس‌های برنامه، به صورت string تعریف می‌شود. سپس اگر نیاز بود به این خاصیت توسط LINQ to XML دسترسی یافت، می‌توان یک فیلد محاسباتی را همانند خاصیت XmlValueWrapper فوق تعریف کرد. نکته دیگری را که باید به آن دقت داشت، استفاده از ویژگی NotMapped

می‌باشد، تا EF سعی نکند خاصیتی از نوع XElement را (یک CLR Property) به بانک اطلاعاتی نگاشت کند.

و همچنین اگر علاقمند هستید که این قابلیت به صورت توکار اضافه شود، می‌توانید اینجا رای دهید !

نحوه تعریف Composite keys در EF Code first

کلاس نوع فعالیت زیر را درنظر بگیرید:

```
namespace EF_Sample04.Models
{
    public class ActivityType
    {
        public int UserId { set; get; }
        public int ActivityID { get; set; }
    }
}
```

در جدول متناظر با این کلاس، نباید دو رکورد تکراری حاوی شماره کاربری و شماره فعالیت یکسانی باهم وجود داشته باشند. بنابراین بهتر است بر روی این دو فیلد، یک کلید ترکیبی تعریف کرد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample04.Models;

namespace EF_Sample04.Mappings
{
    public class ActivityTypeConfig : EntityTypeConfiguration<ActivityType>
    {
        public ActivityTypeConfig()
        {
            this.HasKey(x => new { x.ActivityID, x.UserId });
        }
    }
}
```

در اینجا نحوه معرفی بیش از یک کلید را در متد HasKey ملاحظه می‌کنید.

یک نکته:

اینبار اگر سعی کنیم مثلا از متد db.ActivityTypes.Find با یک پارامتر استفاده کنیم، پیغام خطای «The number of primary key values passed must match number of primary key values defined on the entity» را دریافت خواهیم کرد. برای رفع آن باید هر دو کلید، در این متد قید شوند:

```
var activity1 = db.ActivityTypes.Find(4, 1);
```

ترتیب آن‌ها هم بر اساس ترتیبی که در کلاس ActivityTypeConfig، ذکر شده است، مشخص می‌گردد. بنابراین در این مثال، اولین پارامتر متد Find، به ActivityID اشاره می‌کند و دومین پارامتر به UserId.

بررسی نحوه تعریف نگاشت جداول خود ارجاع دهنده (Self Referencing Entity)

سناریوهای کاربردی بسیاری را جهت جداول خود ارجاع دهنده می‌توان متصور شد و عموما تمام آن‌ها برای مدل سازی اطلاعات

چند سطحی کاربرد دارند. برای مثال یک کارمند را درنظر بگیرید. مدیر این شخص هم یک کارمند است. مسئول این مدیر هم یک کارمند است و الی آخر. نمونه دیگر آن، طراحی منوهای چند سطحی هستند و یا یک مشتری را درنظر بگیرید. مشتری دیگری که توسط این مشتری معرفی شده است نیز یک مشتری است. این مشتری نیز می‌تواند یک مشتری دیگر را به شما معرفی کند و این سلسله مراتب به همین ترتیب می‌تواند ادامه پیدا کند.
در طراحی بانک‌های اطلاعاتی، برای ایجاد یک چنین جداولی، یک کلید خارجی را که به کلید اصلی همان جدول اشاره می‌کند، ایجاد خواهند کرد؛ اما در EF Code first چطور؟

```
using System.Collections.Generic;

namespace EF_Sample04.Models
{
    public class Employee
    {
        public int Id { set; get; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        //public int? ManagerID { get; set; }
        public virtual Employee Manager { get; set; }
    }
}
```

در این کلاس، خاصیت Manager دارای ارجاعی است به همان کلاس؛ یعنی یک کارمند می‌تواند مسئول کارمند دیگری باشد. برای تعریف نگاشت این کلاس به بانک اطلاعاتی می‌توان از روش زیر استفاده کرد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample04.Models;

namespace EF_Sample04.Mappings
{
    public class EmployeeConfig : EntityTypeConfiguration<Employee>
    {
        public EmployeeConfig()
        {
            this.HasOptional(x => x.Manager)
                .WithMany()
                //.HasForeignKey(x => x.ManagerID)
                .WillCascadeOnDelete(false);
        }
    }
}
```

با توجه به اینکه یک کارمند می‌تواند مسئولی نداشته باشد (خودش مدیر ارشد است)، به کمک متد HasOptional مشخص کرده‌ایم که فیلد Manager_Id را که می‌خواهی به این کلاس اضافه کنی باید نال پذیر باشد. توسط متد WithMany طرف دیگر رابطه مشخص شده است.
اگر نیاز بود فیلد Manager_Id اضافه شده نام دیگری داشته باشد، یک خاصیت nullable مانند ManagerID را که در کلاس Employee مشاهده می‌کنید، اضافه نمائید. سپس در طرف تعاریف نگاشت‌ها به کمک متد HasForeignKey، باید صریحا عنوان کرد که این خاصیت، همان کلید خارجی است. از این نکته در سایر حالات تعاریف نگاشت‌ها نیز می‌توان استفاده کرد، خصوصا اگر از یک بانک اطلاعاتی موجود قرار است استفاده شود و از نام‌های دیگری بجز نام‌های پیش فرض EF استفاده کرده است.

مثال‌های این سری رو از این آدرس هم می‌تونید دریافت کنید: (^)

نظرات خوانندگان

نویسنده: Dsictco
تاریخ: ۱۳۹۱/۰۲/۲۱ ۲۰:۰۹:۱۷

سلام
با تشکر از آموزش های مفید شما
آیا امکان Bind کردن گرید به EF وجود دارد؟منظورم مثل ADO.NET. اونجا راحت می تونیم هر فیلدی که میخواهیم به هر ستونی Bind کنیم، این کار رو با EF هم میشه انجام داد؟یا باید کدنویسی کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۱ ۲۲:۵۰:۴۹

این مورد به توانایی‌های LINQ شما بر می‌گردد. در اینجا کوئری‌ها رو باید با LINQ نوشت و سپس مباحث Projection و امثال آن برای تهیه لیست مورد نظر جهت Bind به گریدها می‌تونه مدنظر باشه.
یک قسمت رو به مروری سریع به کوئری نوشتن در EF اختصاص خواهم داد.

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۷/۲۲ ۹:۴۷

سلام در قسمت Self Referencing Entity اگر بخواهیم کلید خارجی نام دیگری داشته باشد طبق گفته شما که عمل کردم خطای Sequence contains no elements را میدهد.

```
using System.Collections.Generic;

namespace EF_Sample04.Models
{
    public class Employee
    {
        public int Id { set; get; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        public int? ManagerID { get; set; }
        public virtual Employee Manager { get; set; }
    }
}
```

```
using EF_Sample04.Models;
using System.Data.Entity.ModelConfiguration;
namespace EF_Sample04.Mappings
{
    public class EmployeeConfig : EntityTypeConfiguration<Employee>
    {
        public EmployeeConfig()
        {
            this.HasOptional(x => x.Manager)
                .WithMany()
                .HasForeignKey(x => x.ManagerID)
                .WillCascadeOnDelete(false);
        }
    }
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۲۲ ۱۰:۴۰

مدل مشکلی نداره (تست شده). اطلاعات بیشتر: « استثنای Sequence contains no elements در حین استفاده از LINQ »
ضمن اینکه مطلب فوق در اینجا کامل شده:
« مباحث تکمیلی مدل‌های خود ارجاع دهنده در EF Code first »

نویسنده: Programer
تاریخ: ۱۳۹۱/۰۷/۲۲ ۱۱:۴۰

علت خطای این قسمت به علت کد زیر بود

```
public ActionResult Index()
{
            NewsContext db = new NewsContext();
            var Query = db.Employee.ToList()
            return View(Query);
}
```

که با تغییر کد به شکل زیر حل شد

```
public ActionResult Index(){
            NewsContext db = new NewsContext();
            var Query = db.Employee.ToList().Where(x=>x.ManagerID==null).ToList();
            return View(Query);
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۲۲ ۱۲:۲۴

چقدر خوب که این مطلب رو تکمیل کردید (هرچند View متناظر در اینجا هم باید ذکر می‌شد).
همچنین چقدر خوب می‌شد زمانیکه سؤال اصلی رو اینجا پرسیدید، موارد فوق رو هم ذکر می‌کردید. چون افراد نمی‌تونند از راه دور کار شما را مشاهده یا دیباگ کنند یا دقیقا بدانند که چه کدی را نوشته‌اید که این خطا را داده. به همین جهت سعی می‌کنند حدس بزنند که در مرحله آخر و پس از نگاشت‌ها، چکار کرده‌اید که خطای فوق حاصل شده.

مطلب خوبی در این زمینه: « نحوه صحیح گزارش دادن یک باگ »

نویسنده: Programer
تاریخ: ۱۳۹۱/۰۷/۲۲ ۱۲:۳۷

مرسی از راهنماییتون

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۸:۱۴

میشه در مورد این کوئری بیشتر توضیح بدین؟

```
db.Employee.ToList().Where(x=>x.ManagerID==null).ToList();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۹:۱

در این مطلب « مباحث تکمیلی مدل‌های خود ارجاع دهنده در EF Code first » بحث شده

عنوان: SQL Server CE و ثبت متون طولانی در EF Code first
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۲ ۱۴:۲۸
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, SQL Server CE

زمانیکه در EF Code first تعریف خاصیتی به نحو زیر باشد

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

در حین کار با SQL Server به صورت خودکار به nvarchar max نگاشت می‌شود. اما همین تعریف در SQL Server CE به nvarchar 4000 نگاشت خواهد شد؛ چون این بانک اطلاعاتی نوع‌های max دار را پشتیبانی نمی‌کند.
بنابراین اگر هدف، ثبت اطلاعات در فیلدی از نوع ntext در این بانک اطلاعاتی باشد باید به یکی از دو روش زیر عمل کرد:

```
[MaxLength]
public string Content { get; set; }
```

بله. فقط کافی است یک MaxLength را بالای خاصیت قرار داد (بدون تعیین طول آن) تا به صورت خودکار در SQL Server CE به ntext نگاشت شود و یا می‌توان نوع ستون را صریحا تعیین کرد:

```
[Column(TypeName = "ntext")]
public string Text { get; set; }
```

روش اول بهتر است از این جهت که با بانک‌های اطلاعاتی مختلف سازگاری بهتری دارد. برای مثال نوع ntext در SQL Server کامل، منسوخ شده درنظر گرفته می‌شود اما اگر از ویژگی MaxLength در اینجا استفاده گردد به صورت خودکار به nvarchar max نگاشت خواهد شد و در SQL Server CE به ntext .
بنابراین قید MaxLength بر روی خواصی که قرار است حاوی متونی طولانی باشند، می‌تواند به عنوان یک کار مفید جهت سازگاری با بانک‌های مختلف، به شمار آید.

عنوان: **بازنویسی ساده‌تر پیش فرض‌های EF Code first در نگارش 6 آن**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۶ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

فرض کنید مطابق اصول نامگذاری که تعیین کرده‌اید، تمام جداول بانک اطلاعاتی شما باید با پیشوند tbl شروع شوند. برای انجام اینکار در نگارش‌های قبلی EF Code first می‌بایستی از ویژگی Table جهت مزین کردن تمامی کلاس‌ها استفاده می‌شد و یا به ازای تک تک موجودیت‌ها، یک کلاس تنظیمات ویژه را افزود و سپس از متد ToTable برای تعیین نامی جدید، استفاده می‌شد. در EF 6 امکان بازنویسی ساده‌تر پیش فرض‌های تعیین نام جداول، طول فیلدها و غیره، پیش بینی شده‌اند که در ادامه تعدادی از آن‌ها را مرور خواهیم کرد.

تعیین پیشوندی برای نام کلیه‌ی جداول بانک اطلاعاتی

اگر نیاز باشد تا به تمامی جداول تهیه شده، بر اساس نام کلاس‌های مدل‌های برنامه، یک پیشوند tbl اضافه شود، می‌توان با بازنویسی متد OnModelCreating کلاس Context برنامه شروع کرد:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // TableNameConvention
            modelBuilder.Types()
                        .Configure(entity => entity.ToTable("tbl" + entity.ClrType.Name));

            base.OnModelCreating(modelBuilder);
        }
```

سپس متد modelBuilder.Types، کلیه موجودیت‌های برنامه را در اختیار قرار داده و در ادامه می‌توان برای مثال از متد ToTable، برای تعیین نامی جدید به ازای کلیه کلاس‌های مدل‌های برنامه استفاده کرد.

تعیین نام دیگری برای کلید اصلی کلیه‌ی جداول برنامه

فرض کنید نیاز است کلیه PKها، با پیشوند نام جدول جاری در بانک اطلاعاتی تشکیل شوند. یعنی اگر نام PK مساوی Id است و نام جدول Menu، نام کلید اصلی نهایی تشکیل شده در بانک اطلاعاتی باید MenuId باشد و نه Id.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // PrimaryKeyNameConvention
            modelBuilder.Properties()
                        .Where(p => p.Name == "Id")
                        .Configure(p => p.IsKey().HasColumnName(p.ClrPropertyInfo.ReflectedType.Name +
"Id"));

            base.OnModelCreating(modelBuilder);
        }
```

این مورد نیز با بازنویسی متد OnModelCreating کلاس Context و سپس استفاده از متد modelBuilder.Properties برای دسترسی به کلیه خواص در حال نگاشت، قابل انجام است. در اینجا کلیه خواصی که نام Id دارند، توسط متد IsKey تبدیل به PK شده و سپس به کمک متد HasColumnName، نام دلخواه جدیدی را خواهند یافت.

تعیین حداکثر طول کلیه فیلدهای رشته‌ای تمامی جداول بانک اطلاعاتی

اگر نیاز باشد تا پیش فرض MaxLength تمام خواص رشته‌ای را تغییر داد، می‌توان از پیاده سازی اینترفیس جدید IStoreModelConvention کمک گرفت:

```
public class StringConventions : IStoreModelConvention<EdmProperty>
    {
```

```
        public void Apply(EdmProperty property, DbModel model)
        {
            if (property.PrimitiveType.PrimitiveTypeKind == PrimitiveTypeKind.String)
            {
                property.MaxLength = 450;
            }
        }
    }
```

در اینجا MaxLength کلیه خواص رشته‌ای در حال نگاشت به بانک اطلاعاتی، به 450 تنظیم می‌شود. سپس برای معرفی آن به برنامه خواهیم داشت:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Add<StringConventions>();
            base.OnModelCreating(modelBuilder);
        }
```

توسط متد modelBuilder.Conventions.Add، می‌توان قراردادهای جدید سفارشی را به برنامه افزود.

نظم بخشیدن به تعاریف قراردادهای پیش فرض

اگر علاقمند نیستید که کلاس Context برنامه را شلوغ کنید، می‌توان با ارث بری از کلاس پایه Convention، قراردادهای جدید را تعریف و سپس توسط متد modelBuilder.Conventions.Add، کلاس نهایی تهیه شده را به برنامه معرفی کرد.

```
public class MyConventions : Convention
    {
        public MyConventions()
        {
            // PrimaryKeyNameConvention
            this.Properties()
                .Where(p => p.Name == "Id")
                .Configure(p => p.IsKey().HasColumnName(p.ClrPropertyInfo.ReflectedType.Name + "Id"));

            // TableNameConvention
            this.Types()
                .Configure(entity => entity.ToTable("tbl" + entity.ClrType.Name));
        }
    }
```

مثال‌های بیشتر

اگر به مستندات EF 6 مراجعه کنید ، مثال‌های بیشتری را در مورد بکارگیری اینترفیس IStoreModelConvention و یا بازنویسی قراردادهای موجود، خواهید یافت .

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۲/۱۱/۲۲ ۲۲:۵

سلام ،
من StringConventions رو مطابق گفته شما انجام دادم ولی کار نکرد . هیچ خطایی هم نداد . فکر نمی‌کنید که از <> modelBuilder.Conventions.AddBefore باید استفاده کنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۳ ۰:۵۰

بعد از MaxLengthAttributeConvention باید اضافه شود تا تنظیمات پیش فرض آن‌را بازنویسی کند و چون کلاس پایه MaxLengthAttributeConvention ، کلاس Convention است باید از روش زیر استفاده کرد:

```
public class CustomMaxLengthConvention : Convention
    {
        public CustomMaxLengthConvention()
        {
            this.Properties<string>().Configure(p => p.HasMaxLength(450));
        }
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
       modelBuilder.Conventions.AddAfter<MaxLengthAttributeConvention>(new
CustomMaxLengthConvention());
```

نویسنده: مسعود سنائی
تاریخ: ۱۳۹۳/۰۴/۲۷ ۱۴:۳۴

از این امکان چطور برای Ignore کردن یک Property میشه استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۲۷ ۱۵:۰

```
//برای یک خاصیت مشخص در یک کلاس مشخص
modelBuilder.Entity<Department>().Ignore(t => t.Budget);

// برای یک کلاس مشخص
modelBuilder.Ignore<OnlineCourse>();

// برای نام خاصیتی مشخص در تمام کلاس‌های نگاشت شده
modelBuilder.Types().Configure(c => c.Ignore("IsDeleted"));

// صرفنظر کردن از تمام اینام‌ها در تمام کلاس‌های نگاشت شده
modelBuilder.Types().Configure(typeConfiguration =>
{
    foreach (var property in typeConfiguration.ClrType
        .GetProperties().Where(p => p.PropertyType.IsEnum))
    {
        typeConfiguration.Ignore(property);
    }
});


// صرفنظر کردن از خواصی که با یک نام مشخص شروع می‌شوند در تمام کلاس‌ها
modelBuilder.Types().Configure(typeConfiguration =>
{
    foreach (var property in typeConfiguration.ClrType
        .GetProperties().Where(p => p.Name.StartsWith("someName")))
    {
        typeConfiguration.Ignore(property);
    }
});
```

نویسنده: مسعود سنائی
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۷:۶

من با EF6 از

```
modelBuilder.Types().Configure(c=>c.Ignore("IsDeleted"));
```

استفاده کردم ولی خطای زیر رو میگیرم:

```
You cannot use Ignore method on the property 'IsDeleted' on type 'MyEntity'
because this type inherits from the type 'BaseEntity' where this property is mapped.
To exclude this property from your model, use NotMappedAttribute or Ignore method on the base type.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۰۲ ۲۲:۴۴

```
modelBuilder.Entity<BaseEntity>().Ignore(p => p.IsDeleted);
```

عنوان: مباحث تکمیلی مدل‌های خود ارجاع دهنده در EF Code first
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۳ ۱۴:۲۴
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC

در مورد طراحی Self Referencing Entities پیشتر مطلبی را در این سایت مطالعه کرده‌اید .
یک مثال دیگر آن می‌تواند نظرات چند سطحی در یک سایت باشند. نحوه تعریف آن با مطالبی که در قسمت هشتم عنوان شود تفاوتی نمی‌کند؛ اما ... زمانیکه نوبت به نمایش آن فرا می‌رسد، چند نکته اضافی را باید درنظر گرفت. ابتدا مثال کامل زیر را در نظر بگیرید:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;

namespace EFGeneral
{
    public class BlogComment
    {
        public int Id { set; get; }

        [MaxLength]
        public string Body { set; get; }

        public virtual BlogComment Reply { set; get; }
        public int? ReplyId { get; set; }
        public ICollection<BlogComment> Children { get; set; }
    }

    public class MyContext : DbContext
    {
        public DbSet<BlogComment> BlogComments { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // Self Referencing Entity
            modelBuilder.Entity<BlogComment>()
                        .HasOptional(x => x.Reply)
                        .WithMany(x => x.Children)
                        .HasForeignKey(x => x.ReplyId)
                        .WillCascadeOnDelete(false);

            base.OnModelCreating(modelBuilder);
        }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            var comment1 = new BlogComment { Body = "نظر من این است که" };
            var comment12 = new BlogComment { Body = "پاسخی به نظر اول", Reply = comment1 };
            var comment121 = new BlogComment { Body = "پاسخی به پاسخ به نظر اول", Reply = comment12 };

            context.BlogComments.Add(comment121);
            base.Seed(context);
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());

            using (var ctx = new MyContext())
            {
                var list = ctx.BlogComments
```

```
                        //.where ...
                        .ToList() // fills the childs list too
                        .Where(x => x.Reply == null) // for TreeViewHelper
                        .ToList();

                if (list.Any())
                {

                }
            }
        }
    }
}
```

در مثال فوق کلاس نظرات به صورت خود ارجاع دهنده (خاصیت Reply به همین کلاس اشاره می‌کند) تعریف شده است تا کاربران بتوانند تا هر چند سطح لازم، به یک نظر خاص، پاسخ دهند.
در اینجا یک چنین جدولی با اطلاعاتی که ملاحظه می‌کنید تشکیل خواهند شد:

یک نظر ارائه شده و سپس دو نظر تو در توی دیگر برای این نظر ثبت شده است.

اولین نکته اضافه‌تری که نسبت به قسمت هشتم قابل ملاحظه است، تعریف خاصیت جدید Children به نحو زیر می‌باشد:

```
public class BlogComment
    {
        // ...
        public ICollection<BlogComment> Children { get; set; }
    }
```

این خاصیت تاثیری در نحوه تشکیل جدول ندارد. علت تعریف آن به توانمندی EF در پرکردن خودکار آن بر می‌گردد.
اگر به کوئری نوشته شده در متد RunTests دقت کنید، ابتدا یک ToList نوشته شده است. این مورد سبب می‌شود که تمام رکوردهای مرتبط دریافت شوند. مثلا در اینجا سه رکورد دریافت می‌شود. سپس برای اینکه حالت درختی آن حفظ شود، در مرحله بعد ریشه‌ها فیلتر می‌شوند (مواردی که reply آن‌ها null است). سپس این مورد تبدیل به list خواهد شد. اینبار اگر خروجی را بررسی کنیم، به ظاهر فقط یک رکورد است اما ... به نحو زیبایی توسط EF به شکل یک ساختار درختی، بدون نیاز به کدنویسی خاصی، منظم شده است:

سؤال:

برای نمایش این اطلاعات درختی و تو در تو در یک برنامه وب چکار باید کرد؟

تا اینجا که توانستیم اطلاعات را به نحو صحیحی توسط EF مرتب کنیم، برای نمایش آن‌ها در یک برنامه ASP.NET MVC می‌توان از یک TreeViewHelper سورس باز استفاده کرد.

البته کد آن در اصل برای استفاده از EF Code first طراحی نشده و نیاز به اندکی تغییر به نحو زیر دارد تا با EF هماهنگ شود (متد ToList و Count موجود در سورس اصلی آن باید به نحو زیر حذف و اصلاح شوند):

```
private void AppendChildren(TagBuilder parentTag, T parentItem, Func<T, IEnumerable<T>>
childrenProperty)
        {
            var children = childrenProperty(parentItem);
            if (children == null || !children.Any())
            {
                return;
            }
//...
```

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۶ ۹:۲۷

آیا میشه این خاصیت رو اینطور پیاده سازی کرد که:

```
public class Person
{
  // other properties

  [Required]
  public virtual Person RelatedPerson {get; set;}
}
```

حالا برای شروع در متد Seed یا معرفی اولین شخص با توجه به این که این RelatedPerson نمیتواند خالی باشد چطور میتوان خود شخص را به این پراپرتی معرفی کرد و بعنوان روت اشخاص از آن استفاده کرد و مپ آن به چه شکل است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۹:۳۱

نمی‌شود. ویژگی Required را حذف کنید تا فیلد nullable شود. در مورد مباحث اولیه نگاشت آن به این مطلب مراجعه کنید.

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۶ ۹:۳۵

در مورد NHibernate هم این محدودیت وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۹:۳۸

این محدودیت نیست. کار یک ORM نگاشت اطلاعات کلاس‌های شما به جداول بانک اطلاعاتی است. زمانیکه سمت بانک اطلاعاتی این فیلد باید null پذیر باشد چون ریشه یک درخت تشکیل شده والدی ندارد، سمت کدهای شما هم به همین ترتیب باید تعریف شود تا نگاشت به نحو صحیحی صورت گیرد.

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۶ ۹:۵۴

در مورد تعریف ORM حق با شماست ولیبا اینکه میشه این رکورد رو تو بانک ایجاد کرد، پس شاید راهی برای اینکار در ORM هم باشه، من کلاس رو این شکلی تعریف کردم و مطمئن هستم حین این ذخیره سازی که این رکورد، رکورد اول جدول هم هست مثلا شناسه اون حتما با یک ذخیره میشه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۹:۵۷

اولین رکورد یا به عبارتی ریشه یک درخت، ریشه‌ای ندارد. این ریشه نال رو چطور در بانک اطلاعاتی تعریف و ذخیره می‌کنید؟ بحث ما Id اولین رکورد نیست. بحث کلیدخارجی است که باید نال پذیر باشد و به همین جدول هم اشاره می‌کند و نمایانگر رکورد والد یک رکورد خاص است (جدول خود ارجاع دهنده). در همین مطلب جاری به تصویر اول دقت کنید. بحث ما فیلد ReplyId است که نال پذیر است. این ReplyId کلید خارجی اشاره‌کننده به همین جدول هم هست.

نویسنده: مهدی پایروند

تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۶

موضوعی که من میخوام براس تو ORM راهی پیدا کنم مربوط به اینه که بتونم کلاسم رو تغییر ندم و از بتونم فیلد پرنت رو غیر نال تعریف کنم، این اسکریپت رو ببینید:

```
CREATE TABLE [dbo].[Tree](
[Id] [int] IDENTITY(1,1) NOT NULL,
[Name] [nchar](10) NOT NULL,
[ParentId] [int] NOT NULL,
 CONSTRAINT [PK_Tree] 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].[Tree]  WITH CHECK ADD  CONSTRAINT [FK_Tree_Tree] FOREIGN KEY([ParentId])
REFERENCES [dbo].[Tree] ([Id])
GO

ALTER TABLE [dbo].[Tree] CHECK CONSTRAINT [FK_Tree_Tree]
GO
```

حالا برای ایجاد رکورد اولی میدونید که شناسه اون یک هست و برای همین میشه رکورد فیلد [ParentId] رو پر کرد من این امتحان رو انجام دادم و مشکلی پیش نیومد

	Id	Name	ParentId
1	1	Root	1

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۹

فیلد parent-id رو دستی مقدار دهی کردی؟ یا با EF مقدار دهی شد؟

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۱۵

تو این مورد با بانک ولی میگم حتما باید برای سمت ORM هم راهی باشه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۱۷

parent-id کلید خارجی است. برای مقدار دهی آن (ثبت اولیه رکورد مرتبط) اگر null پذیر نباشد نیاز است حتما رکورد اشاره کننده به آن وجود خارجی داشته باشد.
به همین دلیل باید در این حالت خاص آن‌را نال پذیر تعریف کرد؛ چون رکورد ریشه، والدی ندارد و کلید خارجی آن نال خواهد بود.
همچنین در این حالت خاص مورد بحث ما، کلید خارجی به خود جدول جاری اشاره می‌کند (و اگر نال پذیر نباشد کل رکورد ریشه، در بار اول ثبت آن، قابل ذخیره سازی نیست).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۳۷

برای درک بهتر این موضوع، سعی کنید دستور زیر را اجرا کنید (از management studio استفاده نکنید):

```
INSERT INTO [Tree]
           ([Name]
           ,[ParentId])
     VALUES
           ('12'
           ,2)
```

قابل ثبت نیست. ضمنا امکان مقدار دهی دستی ParentId هم در اینجا تا زمانیکه رکورد ثبت نشده باشد، میسر نیست (کاری که management studio به صورت دستی انجام داده، چند مرحله کار بوده نه صرفا یک insert معمولی).

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۴۵

بله ممنون، همین موردی که میگید چند مرحله رو باید بشه با ORM بدست آورد، با SQL Profiler میشه از کارکرد SQL Managment نمونه برداری کرد، البته شاید این کار خروجی مناسبی برای سمت کد نداشته باشه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۰:۵۰

طراحی مدنظر شما اصلا متداول نیست. برای مطالعه بیشتر مراجعه کنید به این مقاله:

SQL Server CTE Basics

در اینجا هم ManagerID int **NULL** تعریف شده.

نویسنده: Mina
تاریخ: ۱۳۹۱/۰۹/۱۷ ۲۳:۴۳

با سلام؛
اگر ما در مدل یک فیلد به عنوان مثال status داشته بایم و بخواهیم آنهایی که status شان ok است را بر گردانیم

```
var model = _efComment.List(p => p.PostId == postId);
var list = model.ToList()
.Where(p => p.ComentStatus == ComentStatus.Ok)
.Where(x => x.Reply == null)
.ToList();
```

کد بالا جواب نمیده چون اگه پدرش status باوضعیت ok داشته باشه فرزندانش با هر وضعیتی باش میاره
راه حلی به نظرتون میاد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۹/۱۸ ۰:۱۴

همان ابتدای کار از && برای فیلتر کردن استفاده کنید

```
p => p.PostId == postId && p.ComentStatus == ComentStatus.Ok
```

نویسنده: میلاد محسنی
تاریخ: ۱۳۹۱/۱۱/۱۹ ۲۲:۲۹

با سلام.
برای تهیه مدل‌های خود ارجاع دهنده، نظرتان در خصوص استفاده از hierarchyID چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۱۹ ۲۳:۶

قابل انتقال نیست به سایر بانک‌های اطلاعاتی. یکی از اهداف کار با ORMها مستقل کردن برنامه از بانک اطلاعاتی است. مثلا بتونید با SQL CE هم کار کنید که این نوع داده رو پشتیبانی نمی‌کنه و خیلی از بانک‌های اطلاعاتی دیگر نیز به همین ترتیب.

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۱/۱۲/۰۱ ۱۹:۴۴

با سلام

بنده مدل زیر را دارم که مربوط به صفحاتی هستند که والد هم دارند.

```
public class Page
    {
        public virtual int Id { get; set; }
        public virtual string Title { get; set; }
        public virtual DateTime? CreatedDate { get; set; }
        public virtual DateTime? ModifiedDate { get; set; }
        public virtual string Body { get; set; }
        public virtual string Keyword { get; set; }
        public virtual string Description { get; set; }
        public virtual string Status { get; set; }
        public virtual bool? CommentStatus { get; set; }
        public virtual int? Order { get; set; }
        public virtual User User { get; set; }
        public virtual User EditedByUser { get; set; }
        public virtual ICollection<Comment> Comments { get; set; }
        public virtual int? ParentId { get; set; }
        public virtual Page Parent { get; set; }
        public virtual ICollection<Page> Children { get; set; }
    }
```

و با دستور زیر می‌خواهم از آن کوئری بگیرم:

```
this._pages.ToList().Where(page => page.Parent == null).ToList();
```

دستور فوق به خوبی کار می‌کنه. ولی وقتی با که دستوراتی که توسط mini-profiler لاگ شده را می‌بینیم که اخطار duplicate reader را می‌دهد.

برای هر page موجود دستور زیر را صادر می‌کند

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Title] AS [Title],
[Extent1].[CreatedDate] AS [CreatedDate],
[Extent1].[ModifiedDate] AS [ModifiedDate],
[Extent1].[Body] AS [Body],
[Extent1].[Keyword] AS [Keyword],
[Extent1].[Description] AS [Description],
[Extent1].[Status] AS [Status],
[Extent1].[CommentStatus] AS [CommentStatus],
[Extent1].[Order] AS [Order],
[Extent1].[ParentId] AS [ParentId],
[Extent2].[Id] AS [Id1],
[Extent2].[Title] AS [Title1],
[Extent2].[CreatedDate] AS [CreatedDate1],
[Extent2].[ModifiedDate] AS [ModifiedDate1],
[Extent2].[Body] AS [Body1],
[Extent2].[Keyword] AS [Keyword1],
[Extent2].[Description] AS [Description1],
[Extent2].[Status] AS [Status1],
[Extent2].[CommentStatus] AS [CommentStatus1],
[Extent2].[Order] AS [Order1],
[Extent2].[ParentId] AS [ParentId1],
[Extent2].[User_Id] AS [User_Id],
[Extent2].[EditedByUser_Id] AS [EditedByUser_Id],
[Extent1].[User_Id] AS [User_Id1],
[Extent1].[EditedByUser_Id] AS [EditedByUser_Id1]
FROM  [dbo].[Pages] AS [Extent1]
LEFT OUTER JOIN [dbo].[Pages] AS [Extent2] ON [Extent1].[ParentId] = [Extent2].[Id
```

می‌خواستم ببینم کاری میشه کرد تا سربار این کوئری را کمتر کرد؟

در ضمن اگر بخواهم viewmodel را طوری تعریف کنم تا فیلدهای اضافی مانند createddate و user و... که در هنگام نمایش منوی آبشاری به آنها نیازی ندارم چه کار باید کرد؟ چون من هر کاری کردم نتونستم parent را برای viewmodel به خوبی تعریف کنم.
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۱ ۲۱:۲۹

- این خروجی SQL لاگ شده مطلب جاری (با تمام توضیحات و نگاشت‌های آن) توسط برنامه مطمئن SQL Server Profiler است:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Body] AS [Body],
[Extent1].[ReplyId] AS [ReplyId]
FROM [dbo].[BlogComments] AS [Extent1]
```

منطقی هم هست. چون در ToList اول، کار با دیتابیس تمام و قطع می‌شود. ToList دوم سمت کلاینت اجرا می‌شود. یعنی تشکیل درخت نهایی توسط امکانات LINQ to *Objects* انجام می‌شود و نه هیچ کار اضافه‌ای در سمت سرور.
- اگر اینجا join اضافی پیدا کردید ... حتما مشکلی در تنظیمات نگاشت‌ها دارید.
- اگر duplicate reader دارید شاید بخاطر lazy loading سایر خواص راهبری است که تعریف کردید مانند User و EditByUser و غیره. این‌ها اگر قرار است نمایش داده شوند، پیش از ToList اول باید توسط متد الحاقی Include به صورت eager loading تعریف شوند تا lazy loading و duplicate reader نداشته باشید.
- برای فیلتر فیلدهای اضافی، پیش از ToList اول، با استفاده از Projection و نوشتن یک Select، موارد مورد نیاز را انتخاب کنید.

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۱/۱۲/۰۱ ۲۲:۳۰

ممنون آقای نصیری

با راهنمایی شما مشکل Duplicate Reader با نوشتن دستور به شکل زیر حل شد.

```
this._pages.Include(page => page.Children).ToList().Where(page => page.Parent == null).ToList();
```

الان مشکلی دارم اینه که نمیتونم یه select خوب بنویسم تا فقط مواردی را که میخوام برگردونم.
مثلا من viewmodel را این شکلی تعریف کردم.

```
public class NavBarModel
    {
        public virtual int Id { get; set; }
        public virtual string Title { get; set; }
        public virtual string Status { get; set; }
        public virtual int? Order { get; set; }
        public virtual NavBarModel Parent { get; set; }
        public virtual ICollection<Page> Children { get; set; }
    }
```

توی select زدن نمیدونم چه شکلی باید کد بزنم تا parent و children هم یه صورت خودکار پر شوند.

در حقیقت من می‌خوام این موارد را در یک navigation bar به صورت منوی آبشاری نشون بدم.

در ضمن شما می‌تونید در نشان دادن اطلاعات فوق به صورت منوی آبشاری با امکان تعریف فرزند تو در تو بدون محدودیت راهنماییم کنید.من کدی واسش نوشتم ولی متاسفانه برای پیاده سازی نامحدود بودن فرزندان منو مشکل دارم.

ممنون

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۱:۴۰

سلام

راستش من منوهای چند سطحی پویا را برای bootstrap navbar نوشتم. الگوریتمش را مجبور شدم به صورت بازگشتی بنویسم.اگر کسی می‌تونه به صورت غیر بازگشتی بگه ممنونش میشم.
این کد یه partialpage برای navbar هست که هرکسی برای bootstrap به راحتی میتونه استفاده کنه.

```
@model IEnumerable<DomainClasses.Page>
@helper ShowNavBar(IEnumerable<DomainClasses.Page> pages)
{

    foreach (var page in pages)
    {
        if (page != null)
        {
            if (page.Children.Count == 0)
            {
                <text><li><a tabindex="-1" href="#">@page.Title</a></li></text>
            }

            if (page.Children.Count > 0 && page.Parent == null)
            {
```

```
                <text><li class="dropdown"><a class="dropdown-toggle" id="dLabel" role="button" data-
toggle="dropdown" data-target="#" href="/page.html">@page.Title<b class="caret"></b></a><ul
class="dropdown-menu" role="menu" aria-labelledby="dLabel"><li><a tabindex="-1"
href="#">@page.Title</a></li></text>
                @ShowNavBar(page.Children)
                @:</ul></li>
            }

            if (page.Children.Count > 0 && page.Parent != null)
            {
                <text><li class="dropdown-submenu"><a tabindex="-1" href="#">@page.Title</a><ul
class="dropdown-menu"></text>
                @ShowNavBar(page.Children)
                @:</ul></li>
            }

        }
    }

}
<div class="navbar" style="margin-bottom: 10px;">
    <div class="navbar-inner">
        <a class="brand" href="www.google.com">IT-EBOOK</a>
        <ul class="nav">
            <li class="active"><a href="#">خانه</a></li>
            <li><a href="#">ورود</a></li>
            @ShowNavBar(Model)
            <li><a href="#">ارتباط با ما</a></li>
        </ul>
        <div class="input-append pull-left visible-desktop" style="margin-top: 5px;">
            <input class="span6 search-input" id="Text1" type="text">
            <button class="btn btn-primary" type="button">جست و جو</button>
            <button class="btn btn-info btn-advanced-search" type="button">پیشرفته</button>
        </div>
    </div>
</div>
```

الان تنها مشکلم اینه که فیلدهای اضافی هم کوئری گرفته میشه.میدونم فیلدهای اضافی(بر اساس مدلی که ذکر کردم) را چگونه با استفاده از select حذف کنم اما توی viewmodel نمیدونم چه جوری children را از اطلاعات پر کنم؟

ممنون

نویسنده: ali.rezayee
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۳:۵۶

با سلام و تشکر از از این مطلب عالی.

یک سوال:

اگر جدول BlogComment با یک یا چند جدول دیگر رابطه داشته باشد (مثلاً جدول کاربر، جدول دسترسی و غیره)، چه راه حلی وجود دارد تا در روش خود ارجاع دهنده سایر رابطه‌ها شرکت نکنند، یعنی فقط تعدادی فیلد از جدول BlogComment انتخاب شود. در حالت فعلی تمام فیلدها از تمام جدولهای مرتبط استخراج میشوند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۴:۴۰

- جدول نظرات اگر با موجودیتی به نام کاربر سر و کار دارد، این خاصیت عموما به صورت virtual تعریف می‌شود یعنی lazy loading قرار است رخ دهد. به عبارتی با ToList اولیه اگر از متدی مانند Include استفاده نشده باشد (برای eager loading اطلاعات وابسته)، به هیچ عنوان خواص lazy بارگذاری نخواهند شد. شاید عنوان کنید که من با استفاده از امکانات دیباگر VS.NET اگر نود مربوط به User رو باز کنم، اطلاعاتش هست. پاسخ این است که بله. دقیقا در همین لحظه که نود رو باز کردید یک کوئری برای دریافت اطلاعات یوزر به سرور ارسال شده و نه پیش از آن. البته می‌شود lazy loading را در EF کلا خاموش کرد یا حتی به صورت مقطعی با استفاده از متدی مانند AsNoTracking. اما حالت پیش فرض دقیقا چیزی است که عنوان شد.

بنابراین درحالت فعلی تمام فیلدهای وابسته از بانک اطلاعاتی استخراج نمی‌شوند مگر اینکه lazy loading را به نحوی تبدیل به eager loading کرده باشید.
- ضمنا در حالت فعلی اگر دقت کرده باشید پیش از ToList اول یک سه نقطه گذاشته شده است. یعنی اینجا می‌تونید where بنویسید. می‌تونید Select بنویسید و به صورت اختصاصی یک سری خاصیت مشخص رو انتخاب کنید و خیلی از کارهای دیگر.

نویسنده: ali.rezayee
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۵:۱۹

با تشکر فراوان از پاسخ کامل جنابعالی.
مشکل من از اینجا شروع شد که با Json.Net خواستم نتیجه این کوئری را به Json تبدیل کنم، من دستور زیر را نوشتم، پس از اولین اجرا رکورد اول دو فیلد Body و Id را دارد که کاملاً درست است، اما رکورد بعدی که فرزند رکورد اول است شامل تمام فیلدهای جدول BlogComment و تمام جداول مرتبط با آن است.
ممنون از شما به خاطر صرف وقت گرانبها.

```
var list = context.BlogComment.Where(p => p.UserId == 100)
                .Select(p => p.Body, p.Id, p.Children)
                .ToList();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۶:۱۰

از ویژگی Newtonsoft.Json.JsonIgnore برای عدم serialization یک خاصیت خاص به JSON استفاده کنید.

نویسنده: ali.rezayee
تاریخ: ۱۳۹۲/۰۱/۱۴ ۲۱:۴۰

با تشکر فراوان، مشکل من با Newtonsoft.Json.JsonIgnore حل شد.
اما من برای انتخاب تعدادی خصوصیت از کد زیر استفاده کردم، در نتیجه نهایی که در عکس مشخص است، فقط رکورد اول شامل فیلدهای مشخص شده توسط من است، فرزندان این رکورد حاوی تمام فیلدها هستند.
البته در این کدها از using استفاده نشده چون باعث خطای The ObjectContext instance میشود.
باز هم ممنون برای اختصاص وقت.

```
public partial class BlogComment
    {
        public BlogComment()
        {
            this.Children = new HashSet<BlogComment>();
        }

        public int Id { get; set; }
        public string Body { get; set; }
        public Nullable<System.DateTime> DateSend { get; set; }
        public Nullable<System.DateTime> DateRead { get; set; }
        public bool IsDeleted { get; set; }
        public int UserId { get; set; }

        [Newtonsoft.Json.JsonIgnore]
        public virtual BlogComment Reply { get; set; }

        public int? ReplyId { get; set; }
        public virtual ICollection<BlogComment> Children { get; set; }


    }
```

```
public JsonNetResult Index()
        {
            var ctx = new testEntities();
            var list = ctx.BlogComments
                .Where(p => p.Id == 1)
                .Select(p => new
                                {
                                    p.Id,
                                    p.Body,
                                    p.Children
                                })
            .ToList();

            JsonNetResult jsonNetResult = new JsonNetResult();
            jsonNetResult.Formatting = Formatting.Indented;

            jsonNetResult.SerializerSettings = new JsonSerializerSettings()
                                                    {
                                                        ReferenceLoopHandling =
ReferenceLoopHandling.Ignore
                                                    };
            jsonNetResult.Data = list;
            return jsonNetResult;
        }
```

```
[
  - {
      Id: 1,
      Body: "1 نظر",
    - Children: [
        - {
            - Children: [
                - {
                    Children: [ ],
                    Id: 4,
                    Body: "1-2 نظر",
                    DateSend: "2013-03-03T00:00:00",
                    DateRead: null,
                    IsDeleted: false,
                    UserId: 1,
                    ReplyId: 2
                  }
              ],
              Id: 2,
              Body: "1-1 نظر",
              DateSend: "2013-03-03T00:00:00",
              DateRead: null,
              IsDeleted: false,
              UserId: 1,
              ReplyId: 1
            }
      ]
    }
]
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۲۲:۵۹

علت این است که p.Children به تمام خواص عمومی شیء BlogComment اشاره می‌کند؛ این رو هم میشه یک سطح دیگر با Projection سبک‌تر کرد و یا بجای Projection در حالت شما ساده‌تر است که JsonIgnore را روی تمام خواصی قرار دهید که نباید توسط JSON.NET بررسی شود. با توجه به lazy loading، این خواص virtual توسط EF در بدو امر بارگذاری نمی‌شوند و

همچنین چون توسط JSON.NET به دلیل JsonIgnore معرفی شدن واکاوی مجدد نخواهند شد، بنابراین مشکلی از لحاظ کارآیی یا حجم بالای خروجی نخواهید داشت.

نویسنده: ali.rezayee
تاریخ: ۱۳۹۲/۰۱/۱۴ ۲۳:۳۲

ممنون از لطف شما.
عالی و جامع بود.

نویسنده: مولانا
تاریخ: ۱۳۹۲/۰۲/۱۲ ۲۰:۳۹

دوست عزیز.
از مطلب مفید شما متشکرم. من هم مشکل شما رو دارم. آیا به جواب رسیدید؟

نویسنده: مولانا
تاریخ: ۱۳۹۲/۰۲/۲۴ ۲۲:۴۶

باسلام.
برای مدل‌های خود ارجاع دهنده در هنگام حذف یک رکورد، خطای زیر بوجود می‌آید.

```
The DELETE statement conflicted with the SAME TABLE REFERENCE constraint
```

چاره کار چیست؟ (می خواهم با حذف یک فولدر تمام محتویات آن نیز حذف شوند.)

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۴/۱۰ ۱۱:۳۱

سلام
در مثال بالا چون نظرات با یک پست مرتبط هستند قاعدتا براساس یک پست خاص کامنت‌ها رو محدود می‌کنیم. ولی اگه بخواهیم یک کامنت خاص رو با زیر مجموعه هاش دریافت کنیم شرط جستجو رو چطور باید بنویسیم؟
راهکار زیر در sql هست که دیتابیس رو به sql server محدود می‌کنه و معادل لینک هم فکر نکنم داشته باشه.

```
DECLARE @Table TABLE(
    ID INT,
    ParentID INT,
    NAME VARCHAR(20)
)

INSERT INTO @Table (ID,ParentID,[NAME]) SELECT 1, NULL, 'A'
INSERT INTO @Table (ID,ParentID,[NAME]) SELECT 2, 1, 'B-1'
INSERT INTO @Table (ID,ParentID,[NAME]) SELECT 3, 1, 'B-2'
INSERT INTO @Table (ID,ParentID,[NAME]) SELECT 4, 2, 'C-1'
INSERT INTO @Table (ID,ParentID,[NAME]) SELECT 5, 2, 'C-2'


DECLARE @ID INT

SELECT @ID = 2

;WITH ret AS(
    SELECT*
    FROM@Table
    WHEREID = @ID
    UNION ALL
    SELECTt.*
    FROM@Table t INNER JOIN
    ret r ON t.ParentID = r.ID
)

SELECT  *
```

```
FROM    ret
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۱۰ ۱۳:۵۱

نیازی نیست زیاد پیچیده فکر کنید. یک لیست ساده رو واکشی کنید (مثلا یک کامنت که سه زیر مجموعه مرتبط دارد با یک select ساده)، اتصال نهایی آن‌ها در سمت کلاینت خودکار است. به عبارتی شکل دهی تو در تو در اینجا در سمت کلاینت انجام می‌شود و نه سمت سرور. سمت سرور آن، یک select معمولی از رکوردهای مورد نظر بیشتر نیست.

```
var specifiedCommentId = 4; //عدد 4 در اینجا شماره رکورد کامنتی است که دارای یک سری زیر کامنت است
var list = ctx.BlogComments
                .Include(x => x.Reply)
                .Where(x => x.Id >= specifiedCommentId && (x.ReplyId == x.Reply.Id || x.Reply == 
null))
                .ToList() // fills the childs list too
                .Where(x => x.Reply == null) // for TreeViewHelper
                .ToList();
```

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۴/۱۰ ۱۶:۱۰

ممنون ولی مشکلی با سمت کلاینت ندارم مشکل من شرطی هست که **بدون واکشی کل سطرها** ، **خود ارجاعی رو تا هر سطحی که هست پیمایش کنه** .

فکر کنم سوالم رو اینجوری مطرح کنم بهتر باشه، ما تعدادی کارمند داریم که عضو گروههای زیر هستند.گروه بندی خود ارجاع دهنده هست.

```
GroupRoot {id=1, Name="گروه اصلی",ParentId=null}
Group2 {id=2, Name="بازرگانی",ParentId=1}
Group3 {id=3, Name="فروش",ParentId=2}
Group4 {id=4, Name="فروش قطعات",ParentId=3}
Group5 {id=5, Name="خدمات",ParentId=1}
person1 {name="Ali",  GroupId=2}//بازرگانی
person2 {name="reza", GroupId=3}//فروش
person3 {name="iman", GroupId=3}//فروش
person4 {name="hamid",GroupId=4}//فروش قطعات
person5 {name="hasan",GroupId=4}//فروش قطعات
person6 {name="Ahmad",GroupId=5}//خدمات
person7 {name="vahid",GroupId=1}//گروه اصلی
```

گزارش 1: نمایش گروه بازگانی شناسه 2 به همراه زیر مجموعه هاش ؟{بازرگانی،فروش،فروش قطعات}

گزارش 2 : نمایس پرسنل گروه فروش و زیر مجموعه‌های گروه فروش؟{reza,iman,hamid,hasan}

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۱۰ ۱۶:۳۳

کوئری رو که در پاسخ سؤال قبلی شما نوشته شد، یکبار اجرا کنید.
تمام رکوردها رو واکشی نمی‌کنه. فقط از یک آی دی خاص شروع می‌کنه و مواردی رو که ReplyId اون‌ها با id‌ها یکی است انتخاب می‌کنه. یعنی فقط یک سری زیر خانواده خاص.

مسایل و پروژه‌های شخصی خودتون رو هم در انجمن‌ها پیگیری کنید.

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۴/۱۹ ۲۱:۲۱

برای تکمیل بحث، روش مایکروسافت برای حل مساله مدل‌های خود ارجاع دهنده:
Recursive or hierarchical queries using EF Code First and Migrations
از نوشتن مستقیم SQL و ایجاد View استفاده کرده.

نویسنده: ali.rezayee
تاریخ: ۱۳۹۲/۰۴/۲۴ ۱۲:۵۹

با سلام و عرض تشکر بابت این مطلب کامل

در خصوص مدل‌های خود ارجاع دهنده، اگر بخواهیم مثل عکس بالا هر نود جدیدی که ثبت میشود مقادیر امتیاز تمام والدهای آن تا ریشه تغییر کند بهترین راه حل چیست؟
به بیانی دیگر اگر به شکل دقت کنید هر نود جدیدی که ثبت میشود امتیاز تمامی والدهای آن تا ریشه به روز میشود، برای این کار راه حلی که به نظرم رسید این بود که با ثبت هر نود جدید، تمامی والدهای آن واکشی و مقدار امتیاز آن به روز شوند تا به ریشه برسیم؛ راه حل دیگر نوشتن تریگر در سمت دیتابیس است. فکر میکنم در هر دوحالت سربار زیادی داشته باشیم .آیا راه بهتری وجود دارد؟
ممنون

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۲۴ ۱۴:۷

کم هزینه‌ترین روش: جمع‌ها رو سمت کلاینت مدیریت کنید و اصلا اطلاعات جمع آن‌ها رو سمت سرور ثبت نکنید. زمانیکه این TreeView در حال رندر هست، جمع گره‌های والد رو بر اساس فرزندان محاسبه و نمایش بدید.

نویسنده: ali.rezayee
تاریخ: ۱۳۹۲/۰۴/۲۴ ۱۴:۲۳

بسیار ممنونم از پاسخ شما.
روش شما برای زمانی که تمام درخت لود شود و به کلاینت فرستاده شود کاملاً عالی است. اما اگر با Ajax مرحله به مرحله نودها لود شود دیگر این روش پاسخ نمیدهد چون پدر از امتیازات تمامی فرزندانش بی خبر است.
باز هم ممنون

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۲۴ ۱۴:۳۲

برای نمایش اول کار: جمع کل ریشه اصلی مساوی است با جمع فرزندانی که والد غیر null دارند. یک کوئری سبک بیشتر نیست. مابقی جمع‌های درخت رو میشه سمت کلاینت مدیریت کرد.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۲/۰۵/۰۷ ۱۳:۲۲

روش بهینه‌ی حذف پدر و فرزندان اون چیه؟! ... ممنون میشم اگه رو همین مثال لطف کنین

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۷ ۱۳:۳۰

- نمی‌تونید والد رو یک ضرب حذف کنید. از آخرین فرزند به اول ریشه (والد ریشه جاری) باید (یکی یکی) حذف کنید تا قیود تعریف شده مانع حذف اطلاعات نشوند.
- یا اینکه ... soft delete انجام بدید. یک فیلد bool تعریف کنید که این رکورد خاص deleted هست یا خیر.

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۵/۰۷ ۱۳:۳۲

من از این استفاده می‌کنم.(مدل کامنت و پاسخ هاش)

```
public void Remove(int id)
        {
            var selectedComment = _comments.Find(id);
            _comments.Where(x => x.ParentId == id).Load();
            _comments.Remove(selectedComment);
        }
```

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۲/۰۵/۰۷ ۱۴:۲۳

روش شما رو تست کردم ولی جواب نداد!... فکر میکنم این ساختاری که شما تعریف کردید برای حالت cascadeDelete جواب میده ...

طبق فرمایش آقای نصیری کاری که من کردم اینه (روی سیستم گروه و زیرگروه) :

1 - واکشی گروه انتخاب شده و فرزندان آن با استفاده از الگوریتم اول سطح :

```
private Stack<Group> GetChildsAndRoot(Group group)
        {
            var stack = new Stack<Group>();
            var queue = new Queue<Group>();
            stack.Push(group);
            queue.Enqueue(group);
            while (queue.Any())
            {
                var currGroup = queue.Dequeue();
                foreach (var child in currGroup.Childs)
                {
                    queue.Enqueue(child);
                    stack.Push(child);
                }
            }

            return stack;
        }
```

2- پاک کردن پشته‌ی بازگشتی :

```
var group = _groupRepository.GetByID(id);

var nodes = GetChildsAndRoot(group);
while (nodes.Any())
{
        _groupRepository.Delete(nodes.Pop());
}

     _unitOfWork.SaveChanges();
```

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۵/۰۷ ۱۵:۱۶

نه کار میده. اون هم در حالتی که cascade نباشه.

این کدی که نوشتم دقیقا مال پروژه‌ی IRIS هست. یک بار کدش را در اون پروژه ببینید؛ ضرری نداره.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۳/۰۳/۱۷ ۱۱:۱۸

با سلام.

چگونه می‌توان childها را نیز بر اساس فیلد دلخواه مرتب سازی کرد. برای مثال کوئری زیر تنها سطح اول را مرتب می‌کند و فرزندان را مرتب نمیکند.

```
var query = _tEntities.AsNoTracking()
                .Where(p => p.Parent_Id == null && p.IsActive == true)
                .OrderBy(sortExpression);
var result = query.ToList();
return result;
```

عنوان: بررسی تفصیلی رابطه Many-to-Many در EF Code first
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۴ ۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

رابطه چند به چند در مطالب EF Code first سایت جاری، در حد تعریف نگاشت‌های آن بررسی شده، اما نیاز به جزئیات بیشتری برای کار با آن وجود دارد که در ادامه به بررسی آن‌ها خواهیم پرداخت:

1) پیش فرض‌های EF Code first در تشخیص روابط چند به چند

تشخیص اولیه روابط چند به چند، مانند یک مطلب موجود در سایت و برچسب‌های آن؛ که در این حالت یک برچسب می‌تواند به چندین مطلب مختلف اشاره کند و یا برعکس، هر مطلب می‌تواند چندین برچسب داشته باشد، نیازی به تنظیمات خاصی ندارد. همینقدر که دو طرف رابطه توسط یک ICollection به یکدیگر اشاره کنند، مابقی مسایل توسط EF Code first به صورت خودکار حل و فصل خواهند شد:

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Data.Entity.ModelConfiguration;

namespace Sample
{
    public class BlogPost
    {
        public int Id { set; get; }

        [StringLength(maximumLength: 450, MinimumLength = 1), Required]
        public string Title { set; get; }

        [MaxLength]
        public string Body { set; get; }

        public virtual ICollection<Tag> Tags { set; get; } // many-to-many

        public BlogPost()
        {
            Tags = new List<Tag>();
        }
    }

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

        [StringLength(maximumLength: 450), Required]
        public string Name { set; get; }

        public virtual ICollection<BlogPost> BlogPosts { set; get; } // many-to-many

        public Tag()
        {
            BlogPosts = new List<BlogPost>();
        }
    }

    public class MyContext : DbContext
    {
        public DbSet<BlogPost> BlogPosts { get; set; }
        public DbSet<Tag> Tags { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
```

```
        protected override void Seed(MyContext context)
        {
            var tag1 = new Tag { Name = "Tag1" };
            context.Tags.Add(tag1);

            var post1 = new BlogPost { Title = "Title...1", Body = "Body...1" };
            context.BlogPosts.Add(post1);

            post1.Tags.Add(tag1);

            base.Seed(context);
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());

            using (var ctx = new MyContext())
            {
                var post1 = ctx.BlogPosts.Find(1);
                if (post1 != null)
                {
                    Console.WriteLine(post1.Title);
                }
            }
        }
    }
}
```

در این مثال، رابطه بین مطالب ارسالی در یک سایت و برچسب‌های آن به صورت many-to-many تعریف شده است و همینقدر که دو طرف رابطه توسط یک ICollection به هم اشاره می‌کنند، رابطه زیر تشکیل خواهد شد:

در اینجا تمام تنظیمات صورت گرفته بر اساس یک سری از پیش فرض‌ها است. برای مثال نام جدول واسط تشکیل شده، بر اساس تنظیم پیش فرض کنار هم قرار دادن نام دو جدول مرتبط تهیه شده است.
همچنین بهتر است بر روی نام برچسب‌ها، یک ایندکس منحصربفرد نیز تعیین شود: (^ و ^)

2) تنظیم ریز جزئیات روابط چند به چند در EF Code first

تنظیمات پیش فرض انجام شده آنچنان نیازی به تغییر ندارند و منطقی به نظر می‌رسند. اما اگر به هر دلیلی نیاز داشتید کنترل بیشتری بر روی جزئیات این مسایل داشته باشید، باید از Fluent API جهت اعمال آن‌ها استفاده کرد:

```
public class TagMap : EntityTypeConfiguration<Tag>
    {
        public TagMap()
        {
            this.HasMany(x => x.BlogPosts)
                .WithMany(x => x.Tags)
                .Map(map =>
                    {
                        map.MapLeftKey("TagId");
```

```
                        map.MapRightKey("BlogPostId");
                        map.ToTable("BlogPostsJoinTags");
                    });
        }
    }

    public class MyContext : DbContext
    {
        public DbSet<BlogPost> BlogPosts { get; set; }
        public DbSet<Tag> Tags { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new TagMap());
            base.OnModelCreating(modelBuilder);
        }
    }
```

در اینجا توسط متد Map، نام کلیدهای تعریف شده و همچنین جدول واسط تغییر داده شده‌اند:

3) حذف اطلاعات چند به چند

برای حذف تگ‌های یک مطلب، کافی است تک تک آن‌ها را یافته و توسط متد Remove جهت حذف علامتگذاری کنیم. نهایتا با فراخوانی متد SaveChanges، حذف نهایی انجام و اعمال خواهد شد.

```
using (var ctx = new MyContext())
            {
                var post1 = ctx.BlogPosts.Find(1);
                if (post1 != null)
                {
                    Console.WriteLine(post1.Title);
                    foreach (var tag in post1.Tags.ToList())
                        post1.Tags.Remove(tag);
                    ctx.SaveChanges();
                }
            }
```

در اینجا تنها اتفاقی که رخ می‌دهد، حذف اطلاعات ثبت شده در جدول واسط BlogPostsJoinTags است. Tag1 ثبت شده در متد Seed فوق، حذف نخواهد شد. به عبارتی اطلاعات جداول Tags و BlogPosts بدون تغییر باقی خواهند ماند. فقط یک رابطه بین آن‌ها که در جدول واسط تعریف شده است، حذف می‌گردد.

در ادامه اینبار اگر خود post1 را حذف کنیم:

```
var post1 = ctx.BlogPosts.Find(1);
                if (post1 != null)
                {
                    ctx.BlogPosts.Remove(post1);
                    ctx.SaveChanges();
                }
```

علاوه بر حذف post1، رابطه تعریف شده آن در جدول BlogPostsJoinTags نیز حذف می‌گردد؛ اما Tag1 حذف نخواهد شد. بنابراین دراینجا cascade delete ایی که به صورت پیش فرض وجود دارد، تنها به معنای حذف تمامی ارتباطات موجود در جدول میانی است و نه حذف کامل طرف دوم رابطه. اگر مطلبی حذف شد، فقط آن مطلب و روابط برچسب‌های متعلق به آن از جدول میانی حذف می‌شوند و نه برچسب‌های تعریف شده برای آن.
البته این تصمیم هم منطقی است. از این لحاظ که اگر قرار بود دو طرف یک رابطه چند به چند با هم حذف شوند، ممکن بود با حذف یک مطلب، کل بانک اطلاعاتی خالی شود! فرض کنید یک مطلب دارای سه برچسب است. این سه برچسب با 20 مطلب دیگر هم رابطه دارند. اکنون مطلب اول را حذف می‌کنیم. برچسب‌های متناظر آن نیز باید حذف شوند. با حذف این برچسب‌ها طرف دوم رابطه آن‌ها که چندین مطلب دیگر است نیز باید حذف شوند!

4) ویرایش و یا افزودن اطلاعات چند به چند

در مثال فوق فرض کنید که می‌خواهیم به اولین مطلب ثبت شده، تعدادی تگ جدید را اضافه کنیم:

```
var post1 = ctx.BlogPosts.Find(1);
                if (post1 != null)
                {
                    var tag2 = new Tag { Name = "Tag2" };
                    post1.Tags.Add(tag2);
                    ctx.SaveChanges();
                }
```

در اینجا به صورت خودکار، ابتدا tag2 ذخیره شده و سپس ارتباط آن با post1 در جدول رابط ذخیره خواهد شد.

در مثالی دیگر اگر یک برنامه ASP.NET را درنظر بگیریم، در هربار ویرایش یک مطلب، تعدادی Tag به سرور ارسال می‌شوند. در ابتدای امر هم مشخص نیست کدامیک جدید هستند، چه تعدادی در لیست تگ‌های قبلی مطلب وجود دارند، یا اینکه کلا از لیست برچسب‌ها حذف شده‌اند:

```
//نام تگ‌های دریافتی از کاربر
                var tagsList = new[] { "Tag1", "Tag2", "Tag3" };

                //بارگذاری یک مطلب به همراه تگ‌های آن
                var post1 = ctx.BlogPosts.Include(x => x.Tags).FirstOrDefault(x => x.Id == 1);
                if (post1 != null)
                {
                    //ابتدا کلیه تگ‌های موجود را حذف خواهیم کرد
                    if (post1.Tags != null && post1.Tags.Any())
                        post1.Tags.Clear();

                    //سپس در طی فقط یک کوئری بررسی می‌کنیم کدامیک از موارد ارسالی موجود هستند
                    var listOfActualTags = ctx.Tags.Where(x => tagsList.Contains(x.Name)).ToList();
                    var listOfActualTagNames = listOfActualTags.Select(x => x.Name.ToLower()).ToList();

                    //فقط موارد جدید به تگ‌ها و ارتباطات موجود اضافه می‌شوند
                    foreach (var tag in tagsList)
                    {
                        if (!listOfActualTagNames.Contains(tag.ToLowerInvariant().Trim()))
                        {
                            ctx.Tags.Add(new Tag { Name = tag.Trim() });
                        }
                    }
                    ctx.SaveChanges(); // ثبت موارد جدید

                    //موارد قبلی هم حفظ می‌شوند
                    foreach (var item in listOfActualTags)
                    {
                        post1.Tags.Add(item);
                    }
                    ctx.SaveChanges();
                }
```

در این مثال فقط تعدادی رشته از کاربر دریافت شده است، بدون Id آن‌ها. ابتدا مطلب متناظر، به همراه تگ‌های آن توسط متد Include دریافت می‌شود. سپس نیاز داریم به سیستم ردیابی EF اعلام کنیم که اتفاقاتی قرار است رخ دهد. به همین جهت تمام

تگ‌های مطلب یافت شده را خالی خواهیم کرد. سپس در یک کوئری، بر اساس نام تگ‌های دریافتی، معادل آن‌ها را از بانک اطلاعاتی دریافت خواهیم کرد؛ کوئری tagsList.Contains به where in در طی یک رفت و برگشت، ترجمه می‌شود:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name]
FROM [dbo].[Tags] AS [Extent1]
WHERE [Extent1].[Name] IN (N'Tag1',N'Tag2',N'Tag3')
```

آن‌هایی که جدید هستند به بانک اطلاعاتی اضافه شده (بدون نیاز به تعریف قبلی آن‌ها)، آن‌هایی که در لیست قبلی برچسب‌های مطلب بوده‌اند، حفظ خواهند شد.
لازم است لیست موارد موجود را (listOfActualTags) از بانک اطلاعاتی دریافت کنیم، زیرا به این ترتیب سیستم ردیابی EF آن‌ها را به عنوان رکوردی جدید و تکراری ثبت نخواهد کرد.

5) تهیه کوئری‌های LINQ بر روی روابط چند به چند

الف) دریافت یک مطلب خاص به همراه تمام تگ‌های آن:

```
ctx.BlogPosts.Where(p => p.Id == 1).Include(p => p.Tags).FirstOrDefault()
```

ب) دریافت کلیه مطالبی که شامل Tag1 هستند:

```
var posts = from p in ctx.BlogPosts
                from t in p.Tags
                where t.Name == "Tag1"
                select p;
```

و یا :

```
var posts = ctx.Tags.Where(x => x.Name == "Tag1").SelectMany(x => x.BlogPosts);
```

نظرات خوانندگان

نویسنده: MehRad
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۳:۲۶

با سلام؛ اگر بخواهیم به جدول tagblogpost یک آیتم دیگه اضافه کنیم مثلا تاریخ رو به این جدول اضافه کنیم باید به چه شکل این کار رو انجام دهیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۴ ۱۳:۵۷

پشتیبانی نمی‌شود. این جدول واسط در رابطه many-to-many به صورت داخلی توسط EF مدیریت می‌شود و از دسترس برنامه نویس خارج است.
البته یک راه حل برای آن در اینجا مطرح شده. رابطه دیگر many-to-many نیست. دو رابطه one-to-many تشکیل شده به جدول واسط.

نویسنده: حسین
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۳:۵۴

سلام . یه سوالی که به نظرم رسید و شما نگفتین رو می‌خوام بپرسم. الان توی این مثال اگه ما بخوایم به یه پست جدید تگی رو که موجوده اختصاص بدیم باید چی کار کنیم چون اگه به این صورت عمل کنیم

```
post1.Tags.Add(tag);
```

باید مشخص بشه این تگ یه new object هستش یا خیر یا تگی هست که یبار از طریق context انتخاب شده و داخل حافظه موجوده . اگه این تگ قبلا از طریق context آورده شده باشه آیا به صورت تگی که ازقبل در دیتابیس موجوده به پست مورد نظر تخصیص داده میشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۴:۶

توضیح دادم با مثال: «... در مثالی دیگر اگر یک برنامه ASP.NET را درنظر بگیریم ...»
listOfActualTags از بانک اطلاعاتی دریافت شده؛ بر اساس مواردی که موجود بوده.
به این ترتیب چون این تگ‌ها به سیستم ردیابی EF وارد می‌شوند و همچنین post1.Tags.Clear در ابتدای کار فراخوانی شده، استفاده از متد post1.Tags.Add(item) سبب ثبت مورد تکراری نخواهد شد.
کلا EF هر آیتمی رو که Id آن‌را از طریق دریافت اطلاعات از بانک اطلاعاتی در سیستم ردیابی خودش داشته باشه، جدید و تکراری ثبت نمی‌کنه. برای نمونه در حالت new Tag استفاده شده، این موارد جدید ثبت می‌شوند چون Id از قبل ثبت شده‌ای ندارند.
برای توضیحات بیشتر مراجعه کنید به مطلب نحوه استفاده از کلیدهای خارجی در EF . (حتی می‌شود یک شیء را بدون واکشی از دیتابیس به سیستم ردیابی وارد کرد؛ البته اگر Id آن‌را داشته باشید)

نویسنده: مرتضی
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۳:۱

سلام آقای نصیری بنده یه سوال داشتم بهترین روش برای پیاده سازی رابطه n به m برای طراحی صفحه ادمین چیه؟ببنید بنده توبه قسمت ادمین سایت یک صفحه برای هر جدول طراحی میکنم که از طریق اون بتونم اطلاعات جدول رو حذف و یا بروزرسانی و یا رکورد جدید وارد کنم حالا برای طراحی صفحه ای که جداول اون رابطه n به m دارن دچار مشکل شدم .مثلا فرض بکنید که جدول پست 1000 رکورد داره و جدول tags هم 500 رکورد .حالا برای وارد کردن یک رکورد در داخل جدول رابط باید ID یک رکورد پست و همچنین ID یک رکورد tags رو توبه جدول رابط قرار بدیم بنده خودم از dropdownlist استفاده کردم برای اینکار

که چون تعداد رکوردهای خیلی زیاده این روش جواب گو نیست به نظر شما بهترین روش چیه؟ اگه منظورم رو متوجه نشدید بگید توضیح بیشتری بدم.ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۳:۹

من در سایت جاری برای تعریف روابط چند به چند از افزونه TagIt استفاده کردم.

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۲/۰۳/۱۸ ۱:۲۶

سلام
در قسمت :

```
//فقط موارد جدید به تگ‌ها و ارتباطات موجود اضافه می‌شوند
foreach (var tag in data.Tags)
{
     if (!listOfActualTagNames.Contains(tag.ToLowerInvariant().Trim()))
     {
            _tag.Add(new Tag { Title = tag.Trim() });
      }
}
 _uow.SaveChanges(); // ثبت موارد جدید
```

فقط تگ‌ها در جدول خودشون ذخیره میشن و ارتباطی با پست مربوطه ایجاد نمیشه.
آیا نباید به کد زیر تغییر داد؟

```
if (!listOfActualTagNames.Contains(tag.ToLowerInvariant().Trim()))
 {
      var tags = new Tag { Title = tag.Trim() };
       _tag.Add(tags);
       posts.Tags.Add(tags);
  }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۸ ۱۰:۷

درسته. اصلاحیه کدهای مطلب فوق:

```
var newTag = ctx.Tags.Add(new Tag { Name = tag.Trim() });
post1.Tags.Add(newTag);
```

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۷:۱۹

فرض کنید من لازم دارم در یک برنامه مدیریت آزمایشگاه؛ کاربر بتونه ساختار تستها رو تعریف کنه و بگه اون تست چه پارامترهایی داره و مقادیر مجاز هر پارامتر چی‌ها هست و بعدش بر اساس این ساختار تعریف شده، مقادیر مشاهده شده در آزمایش واقعی رو برای این تست ثبت کنه بنابر این من این کلاسها رو طراحی کرده ام:

تا اینجا ساختار تست که شامل چند پارامتر با مقادیر مجازشون هست رو میشه با این کلاسها تعریف کرد.
حالا فرض کنید برای یک تست تعریف شده با این ساختار میخواهیم مقادیر مشاهده شده واقعی رو ثبت کنیم(ActualTest,ActualParameterValue). وقتی بخوام رنگ خون رو از لیست رنگهای مجاز تعریف شده برای این پارامتر انتخاب کنم، قاعدتا بایستی id مربوط به جدول واسط(TestAcceptableValueInTestParameter) به عنوان id رنگ انتخاب شده در جدول مقدار واقعی پارامتر ثبت بشه. به عبارت دیگه من با id کلاسی کار دارم که اگر با روش many-to-many ذکر شده برای EF کار کنم، همچین کلاسی جزو مدلهای من نیست. آیا EF راهکاری برای این موارد داره؟
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۸:۲۹

چرا رابطه TestParameter و TestAcceptedValue به صورت many-to-many تعریف شده؟ رنگ خون چندین مقدار دارد، اما عکس آن صادق نیست. یعنی یک رنگ خون را نمی‌شود به چندین TestParameter مختلف مانند قند خون یا سطح فلان هورمون انتساب داد.
مثال ساده آن کاربر و نقش‌های او است. یک کاربر می‌تواند چندین نقش داشته باشد (نویسنده، ادیتور و غیره). یک نقش می‌تواند به چندین کاربر منتسب شود (مثلا نقش ادیتور را می‌شود به ده‌ها کاربر انتساب داد). یعنی می‌شود از هر طرف این رابطه، یک رکورد را به چندین رکورد طرف دیگر ربط منطقی داد. اما در حالت مداخل یک آزمایش و مقادیر مجاز جهت یک مدخل، اینچنین نیست و رابطه one-to-many است.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۹:۵۹

فرض کنید آزمایش دیگری غیر از آزمایش خون تعریف شود، مثلا آزمایش ادرار؛ اونوقت این رنگ(و نه رنگ خون) در اونجا هم مورد استفاده پیدا میکند. بنابر این رابطه بایستی many-to-many باشد. یا حتی میتوان برای TestParameter واحد اندازه گیری(UOM) در نظر گرفت که برای آن هم همین مساله رخ میدهد(چون هر پارامتر تست میتواند چندین واحد اندازه گیری داشته باشد و هر واحد اندازه گیری در مورد چندین پارامتر تست استفاده شود).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۳ ۲۱:۵

- در مورد واحدهای اندازه‌گیری منطقی است.
- « آیا EF راهکاری برای این موارد داره؟ » مراجعه کنید به پاسخ اولین نظر مطرح شده. کمی بالاتر.

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۳/۰۹/۱۹ ۰:۱۹

سلام. من ساختاری شبیه به این دارم:

```
public class Person
{
        //...
        public virtual IList<Center> PreferedCenters { get; set; }
        public virtual IList<Center> ActiveCenters { get; set; }

        public Person()
        {
            PreferedCenters = new List<Center>();
            ActiveCenters = new List<Center>();
        }
}
```

و کلا Center هم به شکل زیره:

```
public class Center
{
    //...
    public virtual IList<Person> Persons { get; set; }

    public Center()
    {
        Persons = new List<Person>();
    }
}
```

مشکلی که دارم اینه که منطقا باید دوتا رابطه‌ی Many-to-many تشکیل بشه: ActiveCenters و PreferedCenters. ولی توی جدول خروجی EF، فقط ستونی به اسم Center_ID ساخته می‌شه و وقتی هم که می‌خوام به سیستم چیزی اضافه کنم اروری شبیه به این می‌گیرم:

```
An unhandled exception of type 'System.InvalidOperationException' occurred in EntityFramework.dll

Additional information: Multiplicity constraint violated. The role 'Center_Persons_Source' of the
relationship 'Yarigaran.DataLayer.Center_Persons' has multiplicity 1 or 0..1.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۹ ۰:۵۵

- به ازای هر سر رابطه‌ی چند به چند، باید یک ICollection در دو طرف وجود داشته باشد (ICollection حالت پیش‌فرض EF است و نه IList). در حالت شما، 2 مورد در کلاس شخص و دو مورد در کلاس مرکز. در غیر اینصورت رابطه‌ها یک به چند تفسیر می‌شوند. به علاوه در این حالت مشخص نیست که هر خاصیت به کدام خاصیت باید متصل شود چون InverseProperty ذکر نشده‌است.
- زمانیکه بیش از یک ستون یک جدول قرار است با یک جدول خاص دیگر رابطه‌ی چند به چند داشته باشند، نیاز به چندین جدول واسط خواهد بود که باید به صورت صریح مشخص شوند :

```
modelBuilder.Entity<Person>()
    .HasMany(u => u.PreferedCenters)
    .WithMany(t => t.Persons)
    .Map(x =>
    {
        x.MapLeftKey("PersonId");
        x.MapRightKey("CenterId");
        x.ToTable("Center_Persons1"); // جدول واسط یک
    });
```

```
modelBuilder.Entity<Person>()
    .HasMany(u => u.ActiveCenters)
    .WithMany(t => t.Persons)
    .Map(x =>
    {
        x.MapLeftKey("PersonId");
        x.MapRightKey("CenterId");
        x.ToTable("Center_Persons2"); // جدول واسط دو
    });
```

عنوان: **تقسیم جدول در Entity Framework Code First**
نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۳/۲۹ ۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

سناریو هایی هستند که در آن ها، تعداد ستون‌های یک جدول، بیش از اندازه زیاد می‌شوند و یا آن جدول حاوی فیلدهایی هست که منابع زیادی مصرف می‌کنند، به مانند فیلدهای متنی طولانی یا عکس. معمولا متوجه می‌شویم که در اکثر مواقع، به هنگام واکشی اطلاعات آن جدول، احتیاجی به داده‌های آن فیلدها نداریم و با واکشی بی مورد آن ها، سربار اضافه ای به سیستم تحمیل می‌کنیم، چرا که این داده‌ها ، منابع حافظه ای ما را به هدر می‌دهند.
برای مثال، جدول Post مدل بلاگ را در نظر بگیرید که در آن دو فیلد Body و Image تعریف شده اند.فیلد Body از نوع nvarchar max و فیلد Image از نوع varbinary max است و بدیهی است که این دو داده، به هنگام واکشی حافظه‌ی زیادی مصرف می‌کنند.موارد بسیاری وجود دارند که ما به اطلاعات این دو فیلد احتیاجی نداریم از جمله: نمایش پست‌های پر بازدید، پسته هایی که اخیرا ارسال شده اند و اصولا ما فقط به چند فیلد جدول Post احتیاج داریم و نه همه‌ی آن ها.

```
namespace SplittingTableSample.DomainClasses
{
    public class Post
    {
        public virtual int Id { get; set; }
        public virtual string Title { get; set; }
        public virtual DateTime CreatedDate { get; set; }
        public virtual string  Body { get; set; }
        public virtual byte[] Image { get; set; }
    }
}
```

دلیل اینکه در مدل فوق، تمامی خواص به صورت virtual تعریف شده اند، فعال سازی پروکسی‌های ردیابی تغییر است. اگر دستور زیر را برای واکشی اطلاعات post با id=1 انجام دهیم:

```
using (var context = new MyDbContext())
            {
                var post = context.Posts.Find(1);
            }
```

خروجی زیر را در SQL Server Profiler مشاهده خواهید کرد:

```
exec sp_executesql N'SELECT TOP (2)
[Extent1].[Id] AS [Id],
[Extent1].[Title] AS [Title],
[Extent1].[CreatedDate] AS [CreatedDate],
[Extent1].[Body] AS [Body],
[Extent1].[Image] AS [Image]
FROM [dbo].[Posts] AS [Extent1]
WHERE [Extent1].[Id] = @p0',N'@p0 int',@p0=1
```

همان طور که مشاهده می‌کنید، با اجرای دستور فوق تمامی فیلدهای جدول Posts که id آن‌ها برابر 1 بود واکشی شدند، ولی من تنها به فیلدهای Id و Title آن احتیاج داشتم. خب شاید بگویید که من به سادگی با projection، این مشکل را حل می‌کنم و تنها از فیلد هایی که به آن‌ها احتیاج دارم، کوئری می‌گیرم. همه‌ی این‌ها درست، اما projection هم مشکلات خود را دارد،به صورت پیش فرض، نوع بدون نام بر می‌گرداند و اگر بخواهیم این گونه نباشد، باید مقادیر آن را به یک کلاس(مثلا viewmodel) نگاشت کنیم و کلی مشکل دیگر.
راه حل دیگری که برای حل این مشکل ارائه می‌شود و برای نرمال سازی جداول نیز کاربرد دارد این است که، جدول Posts را به دو جدول مجزا که با یکدیگر رابطه‌ی یک به یک دارند تقسیم کنیم، فیلدهای پر مصرف را در یک جدول و فیلدهای حجیم و کم

مصرف را در جدول دیگری تعریف کنیم و سپس یک رابطه‌ی یک به یک بین آن دو برقرار می‌کنیم.
به طور مثال این کار را بر روی جدول Posts ، به شکل زیر انجام شده است:

```
namespace SplittingTableSample.DomainClasses
{
    public class Post
    {
        public virtual int Id { get; set; }
        public virtual string Title { get; set; }
        public virtual DateTime CreatedDate { get; set; }
        public virtual PostMetaData PostMetaData { get; set; }
    }
}
namespace SplittingTableSample.DomainClasses
{
    public class PostMetaData
    {
        public virtual int PostId { get; set; }
        public virtual string Body { get; set; }
        public virtual byte[] Image { get; set; }
        public virtual Post Post { get; set; }
    }
}
```

همان طور که می‌بینید، خواص حجیم به جدول دیگری به نام PostMetaData منتقل شده و با تعریف خواص راهبری ارجاعی در هر دو کلاس،رابطه‌ی یک به یک بین آن‌ها برقرار شده است.جز الزامات تعریف روابط یک به یک این است که، با استفاده از Fluent API یا Data Annotations ، طرف‌های Depenedent و Principal، صریحا به EF معرفی شوند.

```
namespace SplittingTableSample.DomainClasses
{
    public class PostMetaDataConfig : EntityTypeConfiguration<PostMetaData>
    {
        public PostMetaDataConfig()
        {
            HasKey(x => x.PostId);
            HasRequired(x => x.Post).WithRequiredDependent(x => x.PostMetaData);
        }
    }
}
```

اولین نکته ای که باید به آن توجه شود، این است که در کلاس PostMetaData، قوانین پیش فرض EF برای تعیین کلید اصلی نقض شده است و به همین دلیل، صراحتا با استفاده از متد HasKey ، کلید اصلی به EF معرفی شده است. نکته‌ی مهم دیگری که به آن باید توجه شود این است که هر دو سر رابطه به صورت Required تعریف شده است. دلیل این موضوع هم با توجه به مطلبی که قرار است گفته شود،کمی جلوتر خواهید فهمید. حال اگر تعاریف DbSetها را نیز اصلاح کنیم و دستور زیر را اجرا کنیم:

```
var post = context.Posts.Find(1);
```

خروجی sql زیر را مشاهده خواهید کرد:

```
exec sp_executesql N'SELECT TOP (2)
[Extent1].[Id] AS [Id],
[Extent1].[Title] AS [Title],
[Extent1].[CreatedDate] AS [CreatedDate]
FROM [dbo].[Posts] AS [Extent1]
WHERE [Extent1].[Id] = @p0',N'@p0 int',@p0=1
```

خیلی خوب! دیگر خبری از فیلدهای اضافی Body و Image نیست. دلیل اینکه در اینجا join بین دو جدول مشاهده نمی‌شود،

قابلیت lazy loading است، که با virtual تعریف کردن خواص راهبری حاصل شده است. پس lazy loading در اینجا واقعا مفید است.

اما راه حل ذکر شده نیز کاملا بدون ایراد نیست. مشکل اساسی آن تعدد تعداد جداول آن است. آیا جدول Post ، واقعا احتیاج به چنین سطح نرمال سازی و تبدیل آن به دو جدول مجزا را داشت؟ مطمئنا خیر. آیا واقعا راه حلی وجود دارد که ما در سمت کدهای خود با دو موجودیت مجزا کار کنیم، در صورتی که در دیتابیس این دو موجودیت، ساختار یک جدول را تشکیل دهند. در اینجا روشی مطرح می‌شود به نام تقسیم جدول (Table Splitting).

برای انجام این کار فقط چند تنظیم ساده لازم است:

1) فیلدهای موجودیت مورد نظر را به موجودیت‌های کوچکتر، نگاشت می‌کنیم.

2) بین موجودیت‌های کوچک تر، رابطه‌ی یک به یک که هر دو سر رابطه Required هستند، رابطه برقرار می‌کنم.

3) با استفاده از Fluent API یا DataAnnotations، تمامی موجودیت‌ها را به یک نام در دیتابیس نگاشت می‌کنیم.

برای مثال، تنظیمات Fluent برای کلاس Post و PostMetaData که رابطه‌ی بین آن‌ها یک به یک است را مشاهده می‌کنید:

```
namespace SplittingTableSample.DomainClasses
{
    public class PostConfig : EntityTypeConfiguration<Post>
    {
        public PostConfig()
        {
            ToTable("Posts");
        }
    }
}
namespace SplittingTableSample.DomainClasses
{
    public class PostMetaDataConfig : EntityTypeConfiguration<PostMetaData>
    {
        public PostMetaDataConfig()
        {
            ToTable("Posts");
            HasKey(x => x.PostId);
            HasRequired(x => x.Post).WithRequiredDependent(x => x.PostMetaData);
        }
    }
}
```

نکته مهم این است که در هر دو کلاس(**حتی کلاس Post**) باید با استفاده از متد ToTable، کلاس‌ها را به **یک نام** در دیتابیس نگاشت کنیم. در نتیجه با استفاده از متد ToTable در هر دو موجودیت، آنها در دیتابیس به جدولی به نام Posts نگاشت خواهند شد. تصویر زیر پس از اجرای برنامه، بیان گر این موضوع خواهد بود.

اگر دستورات زیر را اجرا کنید:

```
var post = context.Posts.Find(1);
Console.WriteLine(post.PostMetaData.Body);
```

خروجی زیر را در SQL Server Profiler مشاهده خواهید کرد:

برای متد Find خروجی زیر:

```
exec sp_executesql N'SELECT TOP (2)
[Extent1].[Id] AS [Id],
[Extent1].[Title] AS [Title],
[Extent1].[CreatedDate] AS [CreatedDate]
FROM [dbo].[Posts] AS [Extent1]
WHERE [Extent1].[Id] = @p0',N'@p0 int',@p0=1
```

و برای post.PostMetaData.Body دستور sql زیر را مشاهده می‌کنید:

```
exec sp_executesql N'SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Body] AS [Body],
[Extent1].[Image] AS [Image]
FROM [dbo].[Posts] AS [Extent1]
WHERE [Extent1].[Id] = @EntityKeyValue1',N'@EntityKeyValue1 int',@EntityKeyValue1=1
```

دلیل این که در اینجا ،دو دستور sql به دیتابیس ارسال شده است، فعال بودن ویژگی lazy loading ،به دلیل virtual تعریف کردن خواص راهبری موجودیت‌ها است.

حال اگر بخواهیم با یک رفت و آمد به دیتابیس کلیه اطلاعات را واکشی کنیم، می‌توانیم از Eager Loading استفاده کنیم:

```
var post = context.Posts.Include(x => x.PostMetaData).SingleOrDefault(x => x.Id == 1);
```

که خروجی sql آن نیز به شکل زیر است:

```
SELECT
[Limit1].[Id] AS [Id],
[Limit1].[Title] AS [Title],
[Limit1].[CreatedDate] AS [CreatedDate],
[Extent2].[Id] AS [Id1],
[Extent2].[Body] AS [Body],
[Extent2].[Image] AS [Image]
FROM   (SELECT TOP (2) [Extent1].[Id] AS [Id], [Extent1].[Title] AS [Title], [Extent1].[CreatedDate] AS
[CreatedDate]
FROM [dbo].[Posts] AS [Extent1]
WHERE 1 = [Extent1].[Id] ) AS [Limit1]
LEFT OUTER JOIN [dbo].[Posts] AS [Extent2] ON [Limit1].[Id] = [Extent2].[Id]
```

در نتیجه با کمک این تکنیک توانستیم، با چند موجودیت، در قالب یک جدول رفتار کنیم و از مزیت‌های آن همچون lazy loading، نیز بهره مند شویم.

دریافت کدهای این بخش: SplittingTable-Sample.rar

نظرات خوانندگان

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۲/۰۳/۲۹ ۱۹:۱۳

ممنون ... برا من که خیلی مفید بود :)

نویسنده: محمد
تاریخ: ۱۳۹۳/۰۵/۰۱ ۲۲:۳۹

مطلبی که ارائه دادید در مورد ef6 صدق نمی‌کنه و خطای اینکه این تیبل نمی‌تواند دو کلید داشته باشد را می‌دهد و این در حالی هست که مدل رو با ef5 انجام میدهیم مشکلی نداره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۲:۳۶

PostId را در کلاس PostMetaData تبدیل کنید به Id.

عنوان: **کار با کلیدهای اصلی و خارجی در EF Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۷:۵۱
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در حین کار با ارتباطات بین اشیاء و جداول، دانستن یک سری از نکات می‌توانند در کم کردن تعداد رفت و برگشت‌های به سرور مؤثر واقع شده و نهایتا سبب بالا رفتن سرعت برنامه شوند. از این دست می‌توان به یک سری نکات ریز همراه با primary-keys و foreign-keys اشاره کرد که در ادامه به آن‌ها پرداخته خواهد شد.
در ابتدا کلاس‌های مدل و Context برنامه را به شکل زیر درنظر بگیرید:

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

namespace TestKeys
{
    public class Bill
    {
        public int Id { get; set; }
        public decimal Amount { set; get; }
        public virtual Account Account { get; set; }
    }

    public class Account
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public class MyContext : DbContext
    {
        public DbSet<Bill> Bills { get; set; }
        public DbSet<Account> Accounts { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            var a1 = new Account { Name = "a1" };
            var a2 = new Account { Name = "a2" };

            var bill1 = new Bill { Amount = 100, Account = a1 };
            var bill2 = new Bill { Amount = 200, Account = a2 };

            context.Bills.Add(bill1);
            context.Bills.Add(bill2);
            base.Seed(context);
        }
    }

    public static class Test
    {
        public static void Start()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var ctx = new MyContext())
            {
                var bill1 = ctx.Bills.Find(1);
                Console.WriteLine(bill1.Amount);
            }
        }
    }
}
```

در اینجا کلاس صورتحساب و حساب مرتبط به آن تعریف شده‌اند. سپس به کمک DbContext این دو کلاس در معرض دید EF Code

first قرار گرفته‌اند و در کلاس Configuration نحوه آغاز بانک اطلاعاتی به همراه تعدادی رکورد اولیه مشخص شده است.

نحوه صحیح مقدار دهی کلید خارجی در EF Code first

تا اینجا یک روال متداول را مشاهده کردیم. اکنون سؤال این است که اگر بخواهیم اولین رکورد صورتحساب ثبت شده توسط متد Seed را ویرایش کرده و مثلا حساب دوم را به آن انتساب دهیم، بهینه‌ترین روش چیست؟ بهینه‌ترین در اینجا منظور روشی است که کمترین تعداد رفت و برگشت به بانک اطلاعاتی را داشته باشد. همچنین فرض کنید در صفحه ویرایش، اطلاعات حساب‌ها در یک Drop down list شامل نام و id آن‌ها نیز وجود دارد.

روش اول:

```
using (var ctx = new MyContext())
{
     var bill1 = ctx.Bills.Find(1);
     var a2 = new Account { Id = 2, Name = "a2" };
     bill1.Account = a2;
     ctx.SaveChanges();
}
```

این روش مخصوص تازه واردهای EF Code first است و آنطور که مدنظر آن‌ها است کار نمی‌کند.
به کمک متد Find اولین رکورد یافت شده و سپس بر اساس اطلاعات drop down در دسترس، یک شیء جدید حساب را ایجاد و سپس تغییرات لازم را اعمال می‌کنیم. در نهایت اطلاعات را هم ذخیره خواهیم کرد.
این روش به ظاهر کار می‌کنه اما حاصل آن ذخیره رکورد حساب سومی با id=3 در بانک اطلاعاتی است و سپس انتساب آن به اولین صورتحساب ثبت شده.
نتیجه: Id را دستی مقدار دهی نکنید؛ تاثیری ندارد. زیرا اطلاعات شیء جدید حساب، در سیستم tracking مرتبط با Context جاری وجود ندارد. بنابراین EF آن‌را به عنوان یک شیء کاملا جدید درنظر خواهد گرفت، صرفنظر از اینکه Id را به چه مقداری تنظیم کرده‌اید.

روش دوم:

```
using (var ctx = new MyContext())
{
    var bill1 = ctx.Bills.Find(1);
    var a2 = ctx.Accounts.Find(2);
    bill1.Account = a2;
    ctx.SaveChanges();
}
```

اینبار بر اساس Id دریافت شده از Drop down list، شیء حساب دوم را یافته و به صورتحساب اول انتساب می‌دهیم. این روش درست کار می‌کند؛ اما ... بهینه نیست. فرض کنید شیء جاری دارای 5 کلید خارجی است. آیا باید به ازای هر کلید خارجی یکبار از بانک اطلاعاتی کوئری گرفت؟
مگر نه این است که اطلاعات نهایی ذخیره شده در بانک اطلاعاتی متناظر با حساب صورتحساب جاری، فقط یک عدد بیشتر نیست. بنابراین آیا نمی‌شود ما تنها همین عدد متناظر را بجای دریافت کل شیء به صورتحساب نسبت دهیم؟
پاسخ: بله. می‌شود! ادامه آن در روش سوم.

روش سوم:
در اینجا بهترین کار و یکی از best practices طراحی مدل‌های EF این است که طراحی کلاس صورتحساب را به نحو زیر تغییر دهیم:

```
public class Bill
{
        public int Id { get; set; }
        public decimal Amount { set; get; }
```

```
        [ForeignKey("AccountId")]
        public virtual Account Account { get; set; }
        public int AccountId { set; get; }
}
```

به این ترتیب هم navigation property که سبب تعریف رابطه بین دو شیء و همچنین lazy loading اطلاعات آن می‌شود پابرجا خواهد بود و هم توسط خاصیت جدید AccountId که توسط ویژگی ForeignKey معرفی شده است، ویرایش اطلاعات آن دقیقا همانند کار با یک بانک اطلاعاتی واقعی خواهد شد.
اینبار به کمک خاصیت متناظر با کلید خارجی جدول، مقدار دهی و ویرایش کلیدهای خارجی یک شیء به سادگی زیر خواهد بود؛ خصوصا بدون نیاز به رفت و برگشت اضافی به بانک اطلاعاتی جهت دریافت اطلاعات متناظر با اشیاء تعریف شده به صورت navigation property :

```
using (var ctx = new MyContext())
{
    var bill1 = ctx.Bills.Find(1);
    bill1.AccountId = 2;
    ctx.SaveChanges();
}
```

وارد کردن یک شیء به سیستم Tracking

در قسمت قبل عنوان شد که Id را دستی مقدار دهی نکنید، چون تاثیری ندارد. سؤال: آیا می‌شود این شیء ویژه تعریف شده را به سیستم Tracking وارد کرد؟
پاسخ: بلی. به نحو زیر:

```
using (var ctx = new MyContext())
{
     var a2 = new Account { Id = 2, Name = "a2_a2" };
     ctx.Entry(a2).State = System.Data.EntityState.Modified;
     ctx.SaveChanges();
}
```

در اینجا شیء حساب دوم را به صورت دستی و بدون واکشی از بانک اطلاعاتی ایجاد کرده‌ایم. بنابراین از دیدگاه Context جاری هیچ ارتباطی به بانک اطلاعاتی نداشته و یک شیء جدید درنظر گرفته می‌شود (صرفنظر از Id آن). اما می‌توان این وضعیت را تغییر داد. فقط کافی است State آن‌را به نحوی که ملاحظه می‌کنید به Modified تغییر دهیم. اکنون اگر اطلاعات این شیء را ذخیره کنیم، دقیقا حساب با id=2 در بانک اطلاعاتی ویرایش خواهد شد و نه اینکه حساب جدیدی ثبت گردد.

نظرات خوانندگان

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۹:۲۲

سلام
اگر از الگوی unitofwork که پیاده سازیشو درس دادیداستفاده کرده باشیم نحوه استفاده از using ب چه صورت هستش؟

نویسنده: حمید بهروزی
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۹:۲۴

درود به آقای نصیری عزیز

بر اساس آموزه‌های شما پروژه ای در asp.net mvc4 و ef code first وunitofwork نوشته ام اما برای وهله سازی اینترفیس‌های سرویس و واحد کار برنامه، نمی‌خواهم از کتابخانه structuremap استفاده کنم . پروژه weapsy را مطالعه کردم منتها متوجه نشدم چه جایگزینی برای structuremap برگزیده است . لطفا راهنمایی بفرمایید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۹:۵۲

نیازی نیست به صورت مستقیم فراخوانی شود. کار using رها سازی منابع مرتبط با Context است. اینکار در آنجا در Application_EndRequest به صورت خودکار انجام می‌شود (با کد ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects که نوشته شد).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۹ ۲۰:۳

از Autofac استفاده کرده.

نویسنده: حمید بهروزی
تاریخ: ۱۳۹۱/۰۴/۲۰ ۸:۲۵

سپاس

و این نسبت به structuremap چگونه است ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۰ ۸:۴۲

از این دست کتابخانه‌ها زیاد است. استفاده از این‌ها بیشتر سلیقه‌ای است. یک نفر بر اساس سرعت این‌ها رو بررسی می‌کنه یک نفر بر اساس تعداد قابلیت‌ها. در کل نهایتا تمام این‌ها یک هدف رو برآورده می‌کنند و عموما در Syntax بیشتر با هم تفاوت دارند.

نویسنده: محمدی
تاریخ: ۱۳۹۱/۰۴/۲۲ ۱۰:۳۷

```
An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot
track multiple objects with the same key.
```

وقتی می‌خوام یک entity رو به context اتچ کنم قبلا نباید هیچ entity با این کلید در context وجود داشه باشه مشکل من وقتیه که یک entity رو با ریلیشن های اون اتچ می‌کنم ممکنه یک ریلیشن تکراری وجود داشته باشه که باعث خطای فوق میشه .
یک راه حل اینه که موجودیت‌ها رو به جای attach کردن دوباره از context فراخوانی کنم ولی مطمئنا راه حل اصولی نیست ، ممنون می‌شم راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۲ ۱۰:۵۸

در EF Code first می‌شود از این متد جهت بازنویسی مقادیر سیستم tracking استفاده کرد:

```
context.Entry(oldEntity).CurrentValues.SetValues(newEntity)
```

نویسنده: محمدی
تاریخ: ۱۳۹۱/۰۴/۲۲ ۱۱:۲۷

در نظر بگیرید از فرم انتخاب کالا 10 عدد کالا انتخاب کردیم و می‌خواهیم به پیش فاکتور اضافه کنیم و همزمان روی خود کالا هم تغییراتی بدیم و هر کالا هم سه تا ریلیشن داره و... (البته موضوع من سیستم بارنامه و...است و جهت مثال گفتم درج کالا)
قاعدتا باید آپشنی باشه که به بگیم که از ردگیری تغییرات و .. صرف نظر کنه که من از آپشن زیر استفاده کردم ولی برای اتچ تاثیری نداشت.

```
context.Customers.MergeOption = System.Data.Objects.MergeOption.NoTracking;
```

و یا موقع attach کردن به اون بگیم از ریلیشن‌ها صرف نظر کن. ویا ریلشین‌ها مورد نظر ما رو باز نویسی کن.
من پروژه رو با Database first شروع کردم البته قراره اونو به Code first تبدیل کنم که هنوز وقت نشده! برای Database first راهکاری نیست؟

یه سوالی که ذهنمو خیلی مشغول کرده چطور می‌تونم یک موجودیت رو از ObjectStateManager حذف کنیم به طوری که چیزی در مورد اون موجودیت ندونه ، با dettach کردن باز هم وضعیت اون موجودیت رو در خودش نگه می‌داره.
ببخشید که طولانی شد و ممنون از حوصله شما بابت پاسخگویی.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۲ ۱۱:۵۰

برای حالت database first جهت بازنویسی مقادیر سیستم tracking آن:

```
context.YourEntitySet.ApplyCurrentValues(newEntity)
```

همچنین امکان detach آن هم وجود دارد: (^)

نویسنده: محمدی
تاریخ: ۱۳۹۱/۰۴/۲۲ ۲۳:۵۷

ممنون با کد زیر مشکلم حل شد.

```
foreach (var item in frm.SelectedServices)
                {
                    ObjectStateEntry temp;
                    if (context.ObjectStateManager.TryGetObjectStateEntry(item.Customer.EntityKey, out
```

```
temp))
                    {
                        context.Detach(temp.Entity);
                    }
                    context.Services.Attach(item)
```

;

```
.
                    .
```

```
.
                }
```

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۸

سلام آقای نصیری،
AccountId پراپرتی ای که تو روش سوم به کار بردید اصطلاحاً بهش Foreign Key Property میگن،
حالا سوال بنده اینه که از این Foreign Key Property **فقط** توی رابطه‌های one-to-many(یا many-to-one) استفاده میکنن،درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۱۹

در هر رابطه‌ای که نیاز به تعریف کلید خارجی داشته باشد، بهتر است استفاده شود.
مثلا رابطه many-to-many نیازی به این تعریف ندارد چون مدیریت جدول واسط بین دو موجودیت مرتبط که حاوی کلیدهای خارجی به این دو جدول است، خودکار می‌باشد.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۲۸

درسته،ولی تو رابطه‌های one-to-one(و one-to-zero) هم این روش کاربرد نداره،چون کلید Dependent همین کلید خارجی میشه دیگه،اینم درسته دیگه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۴۰

جمله «در هر رابطه‌ای که نیاز به تعریف کلید خارجی داشته باشد، بهتر است استفاده شود.» نسبتا جامع و مانع است. چون در رابطه 1:1 خودبخود کلید اصلی، کلید خارجی اشتراکی هم هست و نیازی به تعریف مجدد آن نیست.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۴۸

بازم ممنونم.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۱:۲۰

سلام ...
این مدلو ببینید :

```
public class FileUpload
{
    public int FileUploadId { get; set; }
```

```
    public string FileName { get; set; }

}

public class Company
{
    public int CompanyId { get; set; }
    public string Name { get; set; }

    public FileUpload Logo { get; set; }
    public int? LogoId { get; set; }

    public FileUpload Catalog { get; set; }
    public int? CatalogId { get; set; }
}

public class Ads
{
    public int AdsId { get; set; }
    public string Name { get; set; }

    public FileUpload Picture { get; set; }
    public int? PictureId { get; set; }
}

public class TestContext : DbContext
{
    public DbSet<Company> Companies { get; set; }
    public DbSet<Ads> Adses { get; set; }
    public DbSet<FileUpload> FileUploads { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        modelBuilder.Entity<Ads>()
            .HasOptional(a => a.Picture)
            .WithMany()
            .HasForeignKey(a => a.PictureId)
            .WillCascadeOnDelete();

        modelBuilder.Entity<Company>()
           .HasOptional(a => a.Logo)
           .WithMany()
           .HasForeignKey(a => a.LogoId)
           .WillCascadeOnDelete();

        modelBuilder.Entity<Company>()
           .HasOptional(a => a.Catalog)
           .WithMany()
           .HasForeignKey(a => a.CatalogId)
           .WillCascadeOnDelete();
    }
}
```

اینو اگه ازش migration بگیرین متوجه ارورا‌ش میشین

من میخوام هر شرکت بتونه به صورت optional لوگو یا کاتالوگ داشته باشه و همین طور قسمت تبلیغات هم همین طور!

همین طور موقعی هم که مثلا یه شرکت پاک میشه لوگو و کاتالوگشم تو جدول FileUpload پاک شه (cascade delete)

همین طور هر رکورد FileUpload رو بتونم به صورت مستقیم حذف کنم

میشه این کارا که گفتمو کرد؟! چون واقعا یه همچین چیزی نیازه !

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۴:۰

طراحی رو می‌تونید ساده‌تر کنید با قابلیت توسعه بعدی. کلاس Ads رو حذف کنید. خواص لوگو و کاتالوگ رو هم حذف کنید. یک خاصیت به نام FileType به کلاس FileUpload اضافه کنید که می‌تونه تبلیغ، کاتالوگ، لوگو و بسیاری موارد دیگر که در آینده اضافه خواهند شد، باشد. بنابراین این FileType نیاز به یک کلاس جداگانه خواهد داشت برای مدیریت بهتر. استفاده از Enum هم پیشنهاد نمی‌شود چون توسط برنامه و کاربر قابل ویرایش نیست. در آخر یک خاصیت لیستی File هم از نوع FileUpload به کلاس شرکت اضافه کنید.

پ.ن.
این نوع سؤالات شخصی را لطفا در انجمن‌ها پیگیری کنید.

نویسنده: کیا
تاریخ: ۱۳۹۱/۰۸/۲۱ ۱۳:۱۶

من 3تا جدول زیر رو در بانک ساختم :

و کلاسها به صورت زیر تعریف کردم:

```
public class Tb1
    {
        public Tb1()
        {
            ListTb2 = new List<Tb2>();
        }
        public int Id { get; set; }
        public string NameTb1 { get; set; }

        public virtual ICollection<Tb2> ListTb2 { get; set; }
    }
    public class Tb2
    {
        public Tb2()
        {
            ListTb1 = new List<Tb1>();
        }
        public int Id { get; set; }
        public string NameTb2 { get; set; }

        public virtual ICollection<Tb1> ListTb1 { get; set; }
    }
```

و همینطور mapping :

```
public class Tb1Map : EntityTypeConfiguration<Tb1>
    {
        public Tb1Map()
        {
            this.HasKey(x => x.Id);

            this.HasMany(x => x.ListTb2)
```

```
                .WithMany(xx => xx.ListTb1)
                .Map
                (
                    x =>
                        {
                            x.MapLeftKey("Tb1Id");
                            x.MapRightKey("Tb2Id");
                            x.ToTable("Tb1Tb2");
                        }
                );

        }
    }

    public class Tb2Map : EntityTypeConfiguration<Tb2>
    {
        public Tb2Map()
        {
            this.HasKey(x => x.Id);
        }
    }
```

موقعی که در برنامه به صورت زیر استفاده می‌کنم:

```
var sv1 = new TableService<Tb1>(_uow);
            var sv2 = new TableService<Tb2>(_uow);

            var t1 = new Tb1 { NameTb1 = "T111" };
            sv1.Add(t1);
            //var res1= _uow.SaveChanges();

            var t2 = new Tb2 { NameTb2 = "T222" };
            sv2.Add(t2);
            //var res2 = _uow.SaveChanges();

            t1.ListTb2.Add(t2);
            var result = _uow.SaveChanges();
```

هنگام SaveChanges این خطا رو می‌ده:

```
An error occurred while saving entities that do not expose foreign key properties for their
relationships. The EntityEntries property will return null because a single entity cannot be identified
as the source of the exception. Handling of exceptions while saving can be made easier by exposing
foreign key properties in your entity types. See the InnerException for details.
```

همراه با innerException زیر:

```
{"The INSERT statement conflicted with the FOREIGN KEY constraint \"FK_Tb1Tb2_Tb2\". The conflict
occurred in database \"dbTest\", table \"dbo.Tb2\", column 'Id'.\r\nThe statement has been
terminated."}
```

در واقع همینطور که مشخصه من می‌خوام اون جدول رابطه رو در codeFirst حذف کنم یجورایی و رابطه رو بین 2 جدول اصلی بیارم. کجای کارم اشتباهه؟ و راهکارش چیه؟
من با پروفایلر هم نگاه کردم همه چی تا آخر داره پیش می‌ره!
(آیا ForeignKey رو باید طور دیگه ای تعریف کنم؟)
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۱ ۱۳:۵۷

مثال شما رو به صورت مستقل اجرا کردم، جداول ساخته شدند، رکوردها در هر سه جدول ثبت شدند و مشکلی مشاهده نمیشه.
اگر برای دیتابیس موجود قصد دارید mapping تعریف کنید ممکن است کلیدهای تعریف شده در آن کم یا زیاد باشند. بهتر است یک خروجی مستقل از کلاس‌های فوق تهیه کنید (اجازه بدید EF دیتابیس را تولید کند) و بعد با کار خودتون مقایسه کنید که چه

چیزهایی را کم و زیاد دارد.
اگر code-first عمل می‌کنید و دیتابیس قرار است از روی کدهای فوق تهیه شود، تمام نگاشت‌ها را حذف کنید (کلاس‌های Map تعریف شده را)، EF به راحتی روابط man-to-many را تشخیص داده و کلیدهای خارجی و جدول واسط را تهیه می‌کند. نام‌های پیش فرض آن هم از نظر من بسیار مناسب است و نیازی به تغییر ندارند. (تنها تغییری که با بودن کلاس‌های Map فوق حاصل میشه، تعیین نام فیلدهای جدول واسط است و زمانیکه code-first کار می‌کنید این نام‌ها مهم نیستند؛ چون با LINQ نهایتا قرار است کار کنید و خواص کلاس‌ها)

نویسنده: کیا
تاریخ: ۱۳۹۱/۰۸/۲۱ ۱۴:۲۷

ولی با این مثالی که زدم برای من رکوردها ثبت نشدند که! :-؟

اما این راههایی که گفتید رو رفتم بیشترشو و در نهایت مشکل با حذف خط 9 حل شد.
فکر کنم مشکل از 2بار ثبت شدن **موجودیتی تکراری از Tb2** در این جدول باشه! (گرچه علت رو نفهمیدم فکر کنم هنوز!) یکبار در خط 9 که مستقیما موجودیت را به نشانه Added علامت گذاری می‌کنم در ChangeTracker :

```
sv2.Add(t2);
```

و یکبار در خط 12 که همان موجودیت را در navigation property مربوط به موجودیتی از Tb1 مقدار دهی می‌کنم و در خط بعدش که SaveChanges را فراخوانی می‌کنم :

```
t1.ListTb2.Add(t2);
```

(گرچه فکر می‌کردم یک خطای منطقی باید رخ بدهد و حداقل 2بار بصورت duplicate و با id متفاوت ثبت بشه نه اینکه به اینصورت خطا بگیره)

من CodeFirst کار می‌کنم که البته بیشتر اوقات بانک ساخته شده هست و باید به بانک موجود mapping کنم. بیشتر می‌خواستم نحوه عملکرد ef رو روی روابط چند-ب-چند ببینم و اینکه روی یک بانک از قبل آماده به چه صورت باید انجامش بدم. از Ef PowerTools کمک می‌گیرم و روی مسایل حول اینها مشغول تست و کلنجار هستم. مخصوصا حالت‌های مختلف در روابط و جداول رابطه.
تشکر مجدد

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۱/۲۱ ۹:۲۰

سلام؛ من با این روشی که شما فرمودید کار کردم ولی موقع آپدیت با خطای زیر متوقف میشه.
A referential integrity constraint violation occurred: The property values that define the referential constraints are not consistent between principal and dependent objects in the relationship.

دقیقا هم زمانی که می‌خوام State رو به Modified تغییر بدم این خطا رخ می‌ده.اما وقتی که هم foreign key رو مقدار می‌دم و هم مقدار شی navigation property رو از DB واکشی می‌کنم و مقدار دهی می‌کنم درست کار می‌کنه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۲۱ ۹:۵۰

- کدهای این مطلب قبل از ارسال، آزمایش شده‌اند و همین کدهایی را که ملاحظه می‌کنید بدون مشکل کار می‌کنند.
- روش صحیح سؤال پرسیدن این است که مشخص کنید چه مدلی، چه کدی، چه مقادیری در حال استفاده هستند تا این خطا رخ

داده. (امکان باز تولید یک استثناء رو باید بتونید فراهم کنید)
- این خطا زمانی رخ می‌ده که مقادیر کلید اصلی و کلید خارجی در حال ثبت یکی نباشند.

نویسنده: محمد صاحب
تاریخ: ۱۳۹۲/۱۱/۱۲ ۱۵:۰

برای مواردی که کلید اصلی Identity نباشه راه حلی هست ؟

کد

```
namespace TestKeys
{
    class Program
    {
        public class Bill
        {
            [DatabaseGenerated(DatabaseGeneratedOption.None)]
            public string Id { get; set; }
            public decimal Amount { set; get; }
            [ForeignKey("AccountId")]
            public virtual Account Account { get; set; }
            public string AccountId { set; get; }
        }

        public class Account
        {
            [DatabaseGenerated(DatabaseGeneratedOption.None)]
            public string Id { get; set; }
            public string Name { get; set; }
        }

        public class MyContext : DbContext
        {
            public DbSet<Bill> Bills { get; set; }
            public DbSet<Account> Accounts { get; set; }
        }


        public class BillFromWebsrv
        {
            public string Id { get; set; }
            public decimal Amount { set; get; }
            public DateTime DateTime { get; set; }

            public Account Account { get; set; }

        }



        static void Main(string[] args)
        {
            Database.SetInitializer(new DropCreateDatabaseIfModelChanges<MyContext>());
            using (var ctx = new MyContext())
            {

                foreach (var dummyBill in DummyBills())
                {
                    var bl = new Bill { Id = dummyBill.Id, Amount = dummyBill.Amount, Account =
dummyBill.Account };

                    ctx.Bills.Add(bl);
                }
                ctx.SaveChanges();
            }


        }

        public static List<BillFromWebsrv> DummyBills()
        {
            return new List<BillFromWebsrv>
            {
                new BillFromWebsrv
                {
```

```
                    Id = "1",
                    Amount = 1231,
                    DateTime = DateTime.Now,
                    Account = new Account {Id = "1", Name = "ac1"}
                },
                new BillFromWebsrv
                {
                    Id = "2",
                    Amount = 1232,
                    DateTime = DateTime.Now,
                    Account = new Account {Id = "2", Name = "ac2"}
                },
                new BillFromWebsrv
                {
                    Id = "3",
                    Amount = 1233,
                    DateTime = DateTime.Now,
                    Account = new Account {Id = "2", Name = "ac2"}
                },
                new BillFromWebsrv
                {
                    Id = "4",
                    Amount = 1134,
                    DateTime = DateTime.Now,
                    Account = new Account {Id = "3", Name = "ac3"}
                }
            };
        }
    }
}
```

ارور

```
{"Violation of PRIMARY KEY constraint 'PK_dbo.Accounts'. Cannot insert duplicate key in object
'dbo.Accounts'. The duplicate key value is (2).\r\nThe statement has been terminated."}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۲ ۱۸:۴۹

Account هایی که اضافه می‌کنید تحت نظر Context نیستند؛ یک شیء ساده هستند که عنوان کردید، Add اش کن؛ EF هم دقیقا همینکار را انجام داده. زمانیکه به سومین رکورد با اکانتی دارای id=2 می‌رسد، کار متوقف می‌شود چون این id قبلا در رکورد قبلی سعی شده در بانک اطلاعاتی ذخیره شود.

بنابراین باید Context را بررسی کرد که Account در حال اضافه شدن، آیا قبلا در کش Local آن وجود خارجی داشته یا خیر. به این صورت:

```
using (var ctx = new MyContext())
            {
                foreach (var dummyBill in DummyBills())
                {
                    var account = dummyBill.Account;

                    var entry = ctx.Entry<Account>(account);
                    if (entry.State == EntityState.Detached)
                    {
                        var attachedEntity = ctx.Accounts.Local.SingleOrDefault(e => e.Id ==
account.Id);
                        if (attachedEntity != null)
                        {
                            // یعنی قبلا تحت نظر زمینه جاری قرار گرفته و نیازی به ثبت مجدد آن نیست
                            account = attachedEntity;
                        }
                    }

                    var bl = new Bill { Id = dummyBill.Id, Amount = dummyBill.Amount, Account = account
};
                    ctx.Bills.Add(bl);
                }
                ctx.SaveChanges();
            }
```

اگر قبلا تحت نظر قرار گرفته (در کش Accounts.Local موجود است)، باید از همان وهله موجود در Context استفاده شود و نه اینکه یک شیء جدید منقطع را درخواست داد که مجددا ثبت شود.

تنظیمات ارث بری کلاس‌ها در EF Code first

بانک‌های اطلاعاتی مبتنی بر SQL، تنها روابطی از نوع «has a» یا «دارای» را پشتیبانی می‌کنند؛ اما در دنیای شیءگرا روابطی مانند «is a» یا «هست» نیز قابل تعریف هستند. برای توضیحات بیشتر به مدل‌های زیر دقت نمائید:

```
using System;

namespace EF_Sample05.DomainClasses.Models
{
    public abstract class Person
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public DateTime DateOfBirth { get; set; }
    }
}
```

```
namespace EF_Sample05.DomainClasses.Models
{
    public class Coach : Person
    {
        public string TeamName { set; get; }
    }
}
```

```
namespace EF_Sample05.DomainClasses.Models
{
    public class Player : Person
    {
        public int Number { get; set; }
        public string Description { get; set; }
    }
}
```

در این مدل‌ها که بر اساس ارث بری از کلاس شخص، تهیه شده‌اند؛ بازیکن، یک شخص است. مربی نیز یک شخص است؛ و به این ترتیب خوانده می‌شوند:

```
Coach "is a" Person
Player "is a" Person
```

در EF Code first سه روش جهت کار با این نوع کلاس‌ها و کلا ارث بری وجود دارد که در ادامه به آن‌ها خواهیم پرداخت:

الف) Table per Hierarchy یا TPH

همانطور که از نام آن نیز پیدا است، کل سلسله مراتبی را که توسط ارث بری تعریف شده است، تبدیل به یک جدول در بانک اطلاعاتی می‌کند. این حالت، شیوه برخورد پیش فرض EF Code first با ارث بری کلاس‌ها است و نیاز به هیچگونه تنظیم خاصی ندارد.
برای آزمایش این مساله، کلاس Context را به نحو زیر تعریف نمائید و سپس اجازه دهید تا EF بانک اطلاعاتی معادل آن‌را تولید کند:

```
using System.Data.Entity;
using EF_Sample05.DomainClasses.Models;

namespace EF_Sample05.DataLayer.Context
{
    public class Sample05Context : DbContext
    {
        public DbSet<Person> People { set; get; }
    }
}
```

ساختار جدول تولید شده آن همانند تصویر زیر است:

People

	Column Name	Data Type	Allow Nulls
	Id	int	☐
	FirstName	nvarchar(MAX)	☑
	LastName	nvarchar(MAX)	☑
	DateOfBirth	datetime	☐
	Number	int	☑
	Description	nvarchar(MAX)	☑
	TeamName	nvarchar(MAX)	☑
▶	Discriminator	nvarchar(128)	☐
			☐

همانطور که ملاحظه می‌کنید، تمام کلاس‌های مشتق شده از کلاس شخص را تبدیل به یک جدول کرده است؛ به علاوه یک فیلد جدید را هم به نام Discriminator به این جدول اضافه نموده است. برای درک بهتر عملکرد این فیلد، چند رکورد را توسط برنامه به بانک اطلاعاتی اضافه می‌کنیم. حاصل آن به شکل زیر خواهد بود:

Results | Messages

	Id	FirstName	LastName	DateOfBirth	Number	Description	TeamName	Discriminator
1	1	Coach F1	Coach L1	1962-05-11 10:59:25.853	NULL	NULL	Team A	Coach
2	2	Coach F2	Coach L2	1962-05-11 10:59:29.883	NULL	NULL	Team B	Coach
3	3	Coach F1	Coach L1	1962-05-11 10:59:29.883	1	...	NULL	Player

از فیلد Discriminator جهت ثبت نام کلاس‌های متناظر با هر رکورد، استفاده شده است. به این ترتیب EF حین کار با اشیاء دقیقا می‌داند که چگونه باید خواص متناظر با کلاس‌های مختلف را مقدار دهی کند.

به علاوه اگر به ساختار جدول تهیه شده دقت کنید، مشخص است که در حالت TPH، نیاز است فیلدهای متناظر با کلاس‌های مشتق شده از کلاس پایه، همگی null پذیر باشند. برای نمونه فیلد Number که از نوع int تعریف شده، در سمت بانک اطلاعاتی نال پذیر تعریف شده است.

و برای کوئری نوشتن در این حالت می‌توان از متد الحاقی OfType جهت فیلتر کردن اطلاعات بر اساس کلاسی خاص، کمک گرفت:

```
db.People.OfType<Coach>().FirstOrDefault(x => x.LastName == "Coach L1")
```

سفارشی سازی نحوه نگاشت TPH

همانطور که عنوان شد، TPH نیاز به تنظیمات خاصی ندارد و حالت پیش فرض است؛ اما برای مثال می‌توان بر روی مقادیر و نوع ستون Discriminator تولیدی، کنترل داشت. برای این منظور باید از Fluent API به نحو زیر استفاده کرد:

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample05.DomainClasses.Models;

namespace EF_Sample05.DataLayer.Mappings
{
    public class CoachConfig : EntityTypeConfiguration<Coach>
    {
        public CoachConfig()
        {
            // For TPH
            this.Map(m => m.Requires(discriminator: "PersonType").HasValue(1));
        }
    }
}
```

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample05.DomainClasses.Models;

namespace EF_Sample05.DataLayer.Mappings
{
    public class PlayerConfig : EntityTypeConfiguration<Player>
    {
        public PlayerConfig()
        {
            // For TPH
            this.Map(m => m.Requires(discriminator: "PersonType").HasValue(2));
        }
    }
}
```

در اینجا توسط متد Map، نام فیلد discriminator به PersonType تغییر کرده. همچنین چون مقدار پیش فرض تعیین شده توسط متد HasValue عددی است، نوع این فیلد در سمت بانک اطلاعاتی به int null تغییر می‌کند.

ب) Table per Type یا TPT

در حالت TPT، به ازای هر کلاس موجود در سلسله مراتب تعیین شده، یک جدول در سمت بانک اطلاعاتی تشکیل می‌گردد. در جداول متناظر با Sub classes، تنها همان فیلدهایی وجود خواهند داشت که در کلاس‌های هم نام وجود دارد و فیلدهای کلاس پایه در آن‌ها ذکر نخواهد گردید. همچنین این جداول دارای یک Primary key نیز خواهند بود (که دقیقا همان کلید اصلی جدول پایه است که به آن Shared primary key هم گفته می‌شود). این کلید اصلی، به عنوان کلید خارجی اشاره کننده به کلاس یا جدول پایه نیز تنظیم می‌گردد:

برای تنظیم این نوع ارث بری، تنها کافی است ویژگی Table را بر روی Sub classes قرار داد:

```
using System.ComponentModel.DataAnnotations;

namespace EF_Sample05.DomainClasses.Models
{
    [Table("Coaches")]
    public class Coach : Person
    {
        public string TeamName { set; get; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;

namespace EF_Sample05.DomainClasses.Models
{
    [Table("Players")]
    public class Player : Person
    {
        public int Number { get; set; }
        public string Description { get; set; }
    }
}
```

یا اگر حالت Fluent API را ترجیح می‌دهید، همانطور که در قسمت‌های قبل نیز ذکر شد، معادل ویژگی Table در اینجا، متد ToTable است.

ج) Table per Concrete type یا TPC

در تعاریف ارث بری که تاکنون بررسی کردیم، مرسوم است کلاس پایه را از نوع abstract تعریف کنند. به این ترتیب هدف اصلی،

Sub classes تعریف شده خواهند بود؛ چون نمی‌توان مستقیما وهله‌ای را از کلاس abstract تعریف شده ایجاد کرد.
در حالت TPC، به ازای هر sub class غیر abstract، یک جدول ایجاد می‌شود. هر جدول نیز حاوی فیلدهای کلاس پایه می‌باشد (برخلاف حالت TPT که جداول متناظر با کلاس‌های مشتق شده، تنها حاوی همان خواص و فیلدهای کلاس‌های متناظر بودند و نه بیشتر). به این ترتیب عملا جداول تشکیل شده در بانک اطلاعاتی، از وجود ارث بری در سمت کدهای ما بی‌خبر خواهند بود.

Coaches

Column Name	Data Type	Allow Nulls
Id	int	☐
FirstName	nvarchar(MAX)	☑
LastName	nvarchar(MAX)	☑
DateOfBirth	datetime	☐
TeamName	nvarchar(MAX)	☑
		☐

Players

Column Name	Data Type	Allow Nulls
Id	int	☐
FirstName	nvarchar(MAX)	☑
LastName	nvarchar(MAX)	☑
DateOfBirth	datetime	☐
Number	int	☐
Description	nvarchar(MAX)	☑
		☐

برای پیاده سازی TPC نیاز است از Fluent API استفاده شود:

```
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.ModelConfiguration;
using EF_Sample05.DomainClasses.Models;

namespace EF_Sample05.DataLayer.Mappings
{
    public class PersonConfig : EntityTypeConfiguration<Person>
    {
        public PersonConfig()
        {
            // for TPC
            this.Property(x => x.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);
        }
    }
}
```

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample05.DomainClasses.Models;

namespace EF_Sample05.DataLayer.Mappings
{
    public class CoachConfig : EntityTypeConfiguration<Coach>
    {
        public CoachConfig()
        {
            // For TPH
            //this.Map(m => m.Requires(discriminator: "PersonType").HasValue(1));

            // for TPT
            //this.ToTable("Coaches");

            //for TPC
            this.Map(m =>
            {
                m.MapInheritedProperties();
                m.ToTable("Coaches");
            });
        }
    }
}
```

```
using System.Data.Entity.ModelConfiguration;
using EF_Sample05.DomainClasses.Models;

namespace EF_Sample05.DataLayer.Mappings
{
    public class PlayerConfig : EntityTypeConfiguration<Player>
    {
        public PlayerConfig()
        {
            // For TPH
            //this.Map(m => m.Requires(discriminator: "PersonType").HasValue(2));

            // for TPT
            //this.ToTable("Players");

            //for TPC
            this.Map(m =>
                     {
                         m.MapInheritedProperties();
                         m.ToTable("Players");
                     });
        }
    }
}
```

ابتدا نوع فیلد Id از حالت Identity خارج شده است. این مورد جهت کار با TPC ضروری است در غیراینصورت EF هنگام ثبت، به مشکل بر می‌خورد، از این لحاظ که برای دو شیء، به یک Id خواهد رسید و امکان ثبت را نخواهد داد. بنابراین در یک چنین حالتی استفاده از نوع Guid برای تعریف primary key شاید بهتر باشد. بدیهی است در این حالت باید Id را به صورت دستی مقدار دهی نمود.

در ادامه توسط متد MapInheritedProperties، به همان مقصود لحاظ کردن تمام فیلدهای ارث بری شده در جدول حاصل، خواهیم رسید. همچنین نام جداول متناظر نیز ذکر گردیده است.

سؤال : از این بین، بهتر است از کدامیک استفاده شود؟

- برای حالت‌های ساده از TPH استفاده کنید. برای مثال یک بانک اطلاعاتی قدیمی دارید که هر جدول آن 200 تا یا شاید بیشتر فیلد دارد! امکان تغییر طراحی آن هم وجود ندارد. برای اینکه بتوان به حس بهتری حین کارکردن با این نوع سیستم‌های قدیمی رسید، می‌شود از ترکیب TPH و ComplexTypes (که در قسمت‌های قبل در مورد آن بحث شد) برای مدیریت بهتر این نوع جداول در سمت کدهای برنامه استفاده کرد.
- اگر علاقمند به استفاده از روابط پلی‌مرفیک هستید (برای مثال در کلاسی دیگر، ارجاعی به کلاس پایه Person وجود دارد) و sub classes دارای تعداد فیلدهای کمی هستند، از TPH استفاده کنید.
- اگر تعداد فیلدهای sub classes زیاد است و بسیار بیشتر است از کلاس پایه، از روش TPT استفاده کنید.
- اگر عمق ارث بری و تعداد سطوح تعریف شده بالا است، بهتر است از TPC استفاده کنید. حالت TPT از join استفاده می‌کند و حالت TPC از union برای تشکیل کوئری‌ها کمک خواهد گرفت

نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۷/۳۰ ۲۰:۱۴

آقای نصیری با سلام.
من 4 تا Entity دارم و یک Entity پایه. استراژی پباده سازی اون هم TPT است . جدول پایه حدود 100 فیلد دارد. موجودیت‌های فرزند هم هرکدام حدود 30 ستون دارند. وقتی با EF Code First داده‌ها را واکشی میکنم کوئری بسیار سنگین را در SQL Server Profiler ایجاد می‌شود که وقتی آنرا در NotePad کپی میکنم چیزی حدود 60 کیلوبایت می‌شود. ضمناً به تمام ستون‌های جداول نیاز دارم تا آنها را در گرید نمایش دهم.
درصورت امکان راهنمایی کنید. heavyQuery.txt

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۳۰ ۲۱:۴۱

سنگینی یا سبکی یک کوئری بر اساس execution plan آن محاسبه می‌شود و نه حجم کوئری در notepad. بررسی این مورد هم نیاز به جزئیات دقیق کار شما دارد مانند ساختار کلاس‌ها و کوئری LINQ نوشته شده. (ضمن اینکه جوین **5** جدول با هم، با هر ابزاری کند است. این مورد ربطی به EF ندارد. باید طراحی کار خودتون رو بررسی و تصمیم گیری یا اصلاح کنید)

نویسنده: حسین
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۵:۵۵

سلام. میشه در حالت TPH برای فیلد discriminator یک پروپرتی تعریف کنیم؟ یعنی اگر بخواهیم به مقدار فیلد discriminator از طریق کد دسترسی داشته باشیم چکار باید کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۶:۳۲

- این فیلد ویژه به صورت خودکار توسط ساز و کار داخلی EF مدیریت می‌شود و امکان نوشتن یا خواندن مستقیم آن وجود ندارد. (مگر اینکه مستقیما SQL بنویسید و توسط SqlQuery آن‌را اجرا کنید)
- اما ... برای دسترسی به آن جهت یافتن نوعی خاص، فقط کافی هست بنویسید:

```
db.People.OfType<Coach>() // .Where ...
```

OfType مطابق نوع آرگومان جنریکی که دریافت می‌کنه، اطلاعات را بر اساس فیلد discriminator متناظر فیلتر خواهد کرد. مثلا در اینجا افرادی از نوع مربی فیلتر شدن.

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۲:۲۷

سلام
زمانی که از TPT استفاده میکنم و نیاز دارم که یکسری اطلاعات را از جدول پایه فراخوانی کنم بدون اینکه به جداول دیگه نیاز داشته باشم کوئری عجیب غریبی میسازه .
آیا روشی برای اصلاح این نوع کوئری‌ها هست ؟شاید هم من اشتباه استفاده کردم !

این یک تیکه از کوئری ساخته شده است که در آخر هم همه جداول رو با هم جوین میکند.

```
CASE WHEN ((CASE WHEN ([Extent1].[Discriminator] = N'Person') THEN cast(1 as bit) ELSE cast(0 as bit)
END) = 1) THEN CAST(NULL AS varchar(1)) WHEN (([Project6].[C1] = 1) AND ([Project6].[C1] IS NOT NULL)
AND ( NOT (([Project6].[C2] = 1) AND ([Project6].[C2] IS NOT NULL))) AND ( NOT (([Project6].[C3] = 1)
AND ([Project6].[C3] IS NOT NULL))) AND ( NOT (([Project6].[C4] = 1) AND ([Project6].[C4] IS NOT
```

```
NULL)))) THEN [Project6].[Name] WHEN (([Project6].[C2] = 1) AND ([Project6].[C2] IS NOT NULL)) THEN
[Project6].[Name] WHEN (([Project1].[C1] = 1) AND ([Project1].[C1] IS NOT NULL)) THEN CAST(NULL AS
varchar(1)) WHEN (([Project6].[C4] = 1) AND ([Project6].[C4] IS NOT NULL)) THEN [Project6].[Name] WHEN
(([Project2].[C1] = 1) AND ([Project2].[C1] IS NOT NULL)) THEN CAST(NULL AS varchar(1)) WHEN
(([Project6].[C3] = 1) AND ([Project6].[C3] IS NOT NULL)) THEN [Project6].[Name] END AS [C2],
CASE WHEN ((CASE WHEN ([Extent1].[Discriminator] = N'Person') THEN cast(1 as bit) ELSE cast(0 as bit)
END) = 1) THEN CAST(NULL AS varchar(1)) WHEN (([Project6].[C1] = 1) AND ([Project6].[C1] IS NOT NULL)
AND ( NOT (([Project6].[C2] = 1) AND ([Project6].[C2] IS NOT NULL))) AND ( NOT (([Project6].[C3] = 1)
AND ([Project6].[C3] IS NOT NULL))) AND ( NOT (([Project6].[C4] = 1) AND ([Project6].[C4] IS NOT
NULL)))) THEN [Project6].[Family] WHEN (([Project6].[C2] = 1) AND ([Project6].[C2] IS NOT NULL)) THEN
[Project6].[Family] WHEN (([Project1].[C1] = 1) AND ([Project1].[C1] IS NOT NULL)) THEN CAST(NULL AS
varchar(1)) WHEN (([Project6].[C4] = 1) AND ([Project6].[C4] IS NOT NULL)) THEN [Project6].[Family]
WHEN (([Project2].[C1] = 1) AND ([Project2].[C1] IS NOT NULL)) THEN CAST(NULL AS varchar(1)) WHEN
(([Project6].[C3] = 1) AND ([Project6].[C3] IS NOT NULL)) THEN [Project6].[Family] END AS [C3],
CASE WHEN ((CASE WHEN ([Extent1].[Discriminator] = N'Person') THEN cast(1 as bit) ELSE cast(0 as bit)
END) = 1) THEN CAST(NULL AS datetime2) WHEN (([Project6].[C1] = 1) AND ([Project6].[C1] IS NOT NULL)
AND ( NOT (([Project6].[C2] = 1) AND ([Project6].[C2] IS NOT NULL))) AND ( NOT (([Project6].[C3] = 1)
AND ([Project6].[C3] IS NOT NULL))) AND ( NOT (([Project6].[C4] = 1) AND ([Project6].[C4] IS NOT
NULL)))) THEN [Project6].[DateOfBirth] WHEN (([Project6].[C2] = 1) AND ([Project6].[C2] IS NOT NULL))
THEN [Project6].[DateOfBirth] WHEN (([Project1].[C1] = 1) AND ([Project1].[C1] IS NOT NULL)) THEN
CAST(NULL AS datetime2) WHEN (([Project6].[C4] = 1) AND ([Project6].[C4] IS NOT NULL)) THEN
[Project6].[DateOfBirth] WHEN (([Project2].[C1] = 1) AND ([Project2].[C1] IS NOT NULL)) THEN CAST(NULL
AS datetime2) WHEN (([Project6].[C3] = 1) AND ([Project6].[C3] IS NOT NULL)) THEN
[Project6].[DateOfBirth] END AS [C4],
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۳:۱

تنظیمات شما اشتباه است. وجود فیلد Discriminator در بانک اطلاعاتی به معنای استفاده از روش TPH است و نه TPT. در حالت TPH کل کلاس‌های مشتق شده از کلاس پایه، با آن یکی می‌شوند که نیاز به Discriminator برای تمایز قائل شدن بین آن‌ها وجود دارد (در یک جدول و نه در بیش از سه جدولی که در کوئری شما نامبرده شده).

در کل نیاز به بررسی کدهای شما و روابط آن هست. شاید خاصیت ارتباطی اضافه‌ای وجود دارد، شاید روابط صحیح تنظیم نشدن.

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۳:۲۷

```
public class Person:BaseEntity
    {
        public int PersonId { get; set; }
        [StringLength(100)]
        public string Title { get; set; }
        public PersonType PersonType { get; set; }
        public virtual ICollection<PrivacyPolicy> PrivacyPolicies { get; set; }


        public override string ToString()
        {
            return Title;
        }
    }
```

کلاس دوم

```
[Table("Organs")]
    public class Organ:Person
    {
        [StringLength(100)]
        public string FullName { get; set; }
        [StringLength(1000)]
        public string Address { get; set; }

        public override string ToString()
        {
            return FullName;
        }
        public Organ()
        {
            PersonType = PersonType.Organ;
        }
    }
```

تو این مثال از Discriminator استفاده کرده ولی بعضی وقتها کوئری‌ها به این شکل ایجاد میکنه

```
CASE WHEN (( NOT (([UnionAll2].[C7] = 1) AND ([UnionAll2].[C7] IS NOT NULL))) AND ( NOT
(([UnionAll2].[C8] = 1) AND ([UnionAll2].[C8] IS NOT NULL))) AND ( NOT (([UnionAll2].[C9] = 1) AND
([UnionAll2].[C9] IS NOT NULL)))) THEN CAST(NULL AS varchar(1)) WHEN (([UnionAll2].[C9] = 1) AND
([UnionAll2].[C9] IS NOT NULL)) THEN [UnionAll2].[C2] WHEN (([UnionAll2].[C8] = 1) AND
([UnionAll2].[C8] IS NOT NULL)) THEN CAST(NULL AS varchar(1)) END AS [C2],
CASE WHEN (( NOT (([UnionAll2].[C7] = 1) AND ([UnionAll2].[C7] IS NOT NULL))) AND ( NOT
(([UnionAll2].[C8] = 1) AND ([UnionAll2].[C8] IS NOT NULL))) AND ( NOT (([UnionAll2].[C9] = 1) AND
([UnionAll2].[C9] IS NOT NULL)))) THEN CAST(NULL AS varchar(1)) WHEN (([UnionAll2].[C9] = 1) AND
([UnionAll2].[C9] IS NOT NULL)) THEN CAST(NULL AS varchar(1)) WHEN (([UnionAll2].[C8] = 1) AND
([UnionAll2].[C8] IS NOT NULL)) THEN [UnionAll2].[C3] END AS [C3],
CASE WHEN (( NOT (([UnionAll2].[C7] = 1) AND ([UnionAll2].[C7] IS NOT NULL))) AND ( NOT
(([UnionAll2].[C8] = 1) AND ([UnionAll2].[C8] IS NOT NULL))) AND ( NOT (([UnionAll2].[C9] = 1) AND
([UnionAll2].[C9] IS NOT NULL)))) THEN CAST(NULL AS int) WHEN (([UnionAll2].[C9] = 1) AND
([UnionAll2].[C9] IS NOT NULL)) THEN CAST(NULL AS int) WHEN (([UnionAll2].[C8] = 1) AND
([UnionAll2].[C8] IS NOT NULL)) THEN [UnionAll2].[C4] END AS [C4],
```

زمان اجرای کوئری پایین هست ، ولی حجم کد تولید شده بالاهست.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۱ ۰:۹

بهتر هست این مسایل رو در انجمن‌ها پیگیری کنید. کوئری قبلی شما در مورد پروژه و DateOfBirth بود، کلاس‌هایی که ارائه دادید در مورد شخص و ارگان است با یک سری فیلد دیگر. صحبت از TPT بود بعد فیلد Discriminator داشتید.
کار می‌کنه سیستم؟ همین خوبه. دستی هم بخواهید با بیش از سه جدول با هم کار کنید باید جوین بنویسید.
موفق باشید

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۲/۱۱ ۰:۲۵

مهندس جان سوء تفاهم شده ، کوئری که گذاشتم قسمتی از کوئری بود ، من یک کلاس پایه دارم به نام Person و یکسری کلاس مثل Role, User,University,Organ و..... که از Person ارث بری میکنند .
Discriminator هم فالت خودم بود که برای یکی از کلاس‌ها فراموش کرده بودم با Table مزینش کنم.
بحث اصلی من سر کوئری حجیمی هست که تولید میشه.

تو این مسئله من نیازی به جوین ندارم و فقط میخواهم اطلاعات پایه خونده بشه نه بقیه کلاس‌ها ، که این مشکل را با پروجیکشن حل کردم . ولی مخواهم بودنم چرا همچین کوئری میسازه زمانی که ازTPT استفاده میکنم . اونم با این همه Case When و Union.

```
SELECT
CASE WHEN (( NOT (([Project7].[C1] = 1) AND ([Project7].[C1] IS NOT NULL))) AND ( NOT (([Project3].[C1]
= 1) AND ([Project3].[C1] IS NOT NULL))) AND ( NOT (([Project2].[C1] = 1) AND ([Project2].[C1] IS NOT
NULL))) AND ( NOT (([Project1].[C1] = 1) AND ([Project1].[C1] IS NOT NULL)))) THEN '0X' WHEN
(([Project7].[C1] = 1) AND ([Project7].[C1] IS NOT NULL) AND ( NOT (([Project7].[C2] = 1) AND
([Project7].[C2] IS NOT NULL))) AND ( NOT (([Project7].[C3] = 1) AND ([Project7].[C3] IS NOT NULL)))
AND ( NOT (([Project7].[C4] = 1) AND ([Project7].[C4] IS NOT NULL)))) THEN '0X0X' WHEN
(([Project7].[C2] = 1) AND ([Project7].[C2] IS NOT NULL)) THEN '0X0X0X' WHEN (([Project2].[C1] = 1) AND
([Project2].[C1] IS NOT NULL)) THEN '0X1X' WHEN (([Project7].[C4] = 1) AND ([Project7].[C4] IS NOT
NULL)) THEN '0X0X1X' WHEN (([Project3].[C1] = 1) AND ([Project3].[C1] IS NOT NULL)) THEN '0X2X' WHEN
(([Project7].[C3] = 1) AND ([Project7].[C3] IS NOT NULL)) THEN '0X0X2X' ELSE '0X3X' END AS [C1],
[Extent1].[PersonId] AS [PersonId],
[Extent1].[Title] AS [Title],
[Extent1].[PersonType] AS [PersonType],
```

آیا واقعاً این همه Case لازم داره ، یا بازم من اشتباه کردم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۱ ۱:۱

- کارآیی یک کوئری بر اساس execution plan آن بررسی می‌شود و نه حجم حاصل از فیلدهای درگیر در آن.
- هرگونه مشکلات و ناراحتی‌های خودتون رو در مورد طراحی EF در اینجا و یا اینجا ارسال کنید.

نویسنده: وحید م
تاریخ: ۱۳۹۲/۰۹/۲۹ ۱۷:۲۴

با سلام یک کلاس abstrac بنام person که یک سری خصوصیات دارد و یکسری کلاس از آن مشتق شده حال کلاس person چگونه می‌تواند به فیلدهای کلاس مشتق شده دسترسی داشته باشد مثل حالت tph که ما فقط به dbset مان person رادادیم وآن توانست جدولی با تمام فیلدهای کلاس مستق شده برایمان درست نماید ممنون. این روند فقط درcodefirst موجود است یا خاصیت oop است

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۹ ۱۸:۳۷

از متد الحاقی OfType برای دسترسی به خواص زیرکلاس‌ها استفاده کنید. مثالش در متن هست.

نویسنده: محمد رضا خزائی
تاریخ: ۱۳۹۲/۱۲/۱۷ ۱۴:۱۵

با سلام
در روش TPT اگر بخواهیم فقط اطلاعات جدول پایه (پدر) را select بزنیم،متاسفانه تمامی جداول مشتق شده با هم Union شده و بعد با جدول پایه Join میخورد.
آیا راهی وجود داره که فقط از جدول پایه Select زده شود؟
مرسی

نویسنده: محمد رضا خزائی
تاریخ: ۱۳۹۲/۱۲/۱۷ ۱۶:۱۴

پیدا کردم
باید کلاس پایه رو از حالت Abstract خارج کنیم.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۸/۱۷ ۲۳:۷

خیلی ممنون؛ من الان یه کلاس انتزاعی تعریف کردم که دو کلاس به نام‌های Article وpage ازش ارث بری می‌کنن. حالا توی کلاس انتزاعی من یه خاصیت دارم که لیستی از کلاس tag هست و میخوام که تگ‌های اضافه شده برای هر رکورد در یک جدول بشینه. حالا مشکلی که پیش میاد طبیعتا EF فقط یک جدول تگ میسازه که اون رو هم نمی‌دونم بر چه اساسی برای article کاربرد داره. آیا باید خاصیت تگ روی برای کلاس‌های فرزند جداگانه تعریف کنم یا راهکاری هم براش هست؟
حواسم به یک چیزی نبود که فقط تگ‌ها رو برای article ثبت می‌کرد. الان مشکلی که هست یک جدول تگ برای هر دو در نظر گرفته که اینطوری هم متوجه نمیشیم که کلید مربوطه مال جدول page میشه یا article؟ در این حالت من باید کلاس تگ رو برای هر کدوم جداگانه بنویسم که دو جدول بسازه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۸ ۰:۱

بله. رابطه many-to-many هست و هر کدام جدول و خاصیت جداگانه خاص خودشان را باید داشته باشند.

عنوان: **ایندکس منحصر به فرد با استفاده از Data Annotation در EF Code First**
نویسنده: پدرام جباری
تاریخ: ۱۳۹۱/۰۹/۲۲ ۲۳:۲۸
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Data Annotation, Unique, Index

در حال حاضر امکان خاصی برای ایجاد ایندکس منحصر به فرد در EF First Code وجود ندارد, برای این کار راه‌های زیادی وجود دارد مانند پست قبلی آقای نصیری, در این آموزش از Data Annotation و یا همان Attribute هایی که بالای Propertyهای مدل‌ها قرار می‌دهیم, مانند کد زیر :

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

        [Unique]
        public string Email { get; set; }

        [Unique("MyUniqueIndex",UniqueIndexOrder.ASC)]
        public string Username { get; set; }

        [Unique(UniqueIndexOrder.DESC)]
        public string PersonalCode{ get; set; }

        public string Password { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
```

همانطور که در کد بالا می‌بینید با استفاده از Attribute Unique ایندکس منحصر به فرد آن در دیتابیس ساخته خواهد شد.
ابتدا یک کلاس برای Attribute Unique به صورت زیر ایجاد کنید :

```
using System;

namespace SampleUniqueIndex
{
    [AttributeUsage(AttributeTargets.Property, Inherited = false, AllowMultiple = false)]
    public class UniqueAttribute : Attribute
    {
        public UniqueAttribute(UniqueIndexOrder order = UniqueIndexOrder.ASC) {
            Order = order;
        }
        public UniqueAttribute(string indexName,UniqueIndexOrder order = UniqueIndexOrder.ASC)
        {
            IndexName = indexName;
            Order = order;
        }
        public string IndexName { get; private set; }
        public UniqueIndexOrder Order { get; set; }
    }

    public enum UniqueIndexOrder
    {
        ASC,
        DESC
    }
}
```

در کد بالا یک Enum برای مرتب سازی ایندکس به دو صورت صعودی و نزولی قرار دارد, همانند کد ابتدای آموزش که مشاهده می‌کنید امکان تعریف این Attribute به سه صورت امکان دارد که به صورت زیر می‌باشد:

1. ایجاد Attribute بدون هیچ پارامتری که در این صورت نام ایندکس با استفاده از نام جدول و آن فیلد ساخته خواهد شد : IX_Unique_TableName_FieldName و مرتب ساری آن به صورت صعودی می‌باشد.
2. نامی برای ایندکس انتخاب کنید تا با آن نام در دیتابیس ذخیره شود, در این حالت مرتب سازی آن هم به صورت صعودی می‌باشد.
3. در حالت سوم شما ضمن وارد کردن نام ایندکس مرتب سازی آن را نیز وارد می‌کنید.

بعد از کلاس Attribute حالا نوبت به کلاس اصلی میرسد که به صورت زیر می‌باشد:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Metadata.Edm;
using System.Linq;
using System.Reflection;

namespace SampleUniqueIndex
{
    public static class DbContextExtention
    {
        private static BindingFlags PublicInstance = BindingFlags.Public | BindingFlags.Instance |
BindingFlags.FlattenHierarchy;

        public static void ExecuteUniqueIndexes(this DbContext context)
        {
            var tables = GetTables(context);
            var query = "";
            foreach (var dbSet in GetDbSets(context))
            {
                var entityType = dbSet.PropertyType.GetGenericArguments().First();
                var table = tables[entityType.Name];
                var currentIndexes = GetCurrentUniqueIndexes(context, table.TableName);
                foreach (var uniqueProp in GetUniqueProperties(context, entityType, table))
                {
                    var indexName = string.IsNullOrWhiteSpace(uniqueProp.IndexName) ?
                        "IX_Unique_" + uniqueProp.TableName + "_" + uniqueProp.FieldName :
                        uniqueProp.IndexName;

                    if (!currentIndexes.Contains(indexName))
                    {
                        query += "ALTER TABLE [" + table.TableSchema + "].[" + table.TableName + "] ADD
CONSTRAINT [" + indexName + "] UNIQUE ([" + uniqueProp.FieldName + "] " + uniqueProp.Order + "); ";
                    }
                    else
                    {
                        currentIndexes.Remove(indexName);
                    }
                }
                foreach (var index in currentIndexes)
                {
                    query += "ALTER TABLE [" + table.TableSchema + "].[" + table.TableName + "] DROP
CONSTRAINT " + index + "; ";
                }
            }

            if (query.Length > 0)
                context.Database.ExecuteSqlCommand(query);
        }

        private static List<string> GetCurrentUniqueIndexes(DbContext context, string tableName)
        {
            var sql = "SELECT CONSTRAINT_NAME FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS where
table_name = '"
                      + tableName + "' and CONSTRAINT_TYPE = 'UNIQUE'";
            var result = context.Database.SqlQuery<string>(sql).ToList();
            return result;
        }
        private static IEnumerable<PropertyDescriptor> GetDbSets(DbContext context)
        {
            foreach (PropertyDescriptor prop in TypeDescriptor.GetProperties(context))
            {
                var notMapped = prop.GetType().GetCustomAttributes(typeof(NotMappedAttribute),true);
                if (prop.PropertyType.Name == typeof(DbSet<>).Name && notMapped.Length == 0)
                    yield return prop;
            }
        }
        private static List<UniqueProperty> GetUniqueProperties(DbContext context, Type entity,
TableInfo tableInfo)
        {
            var indexedProperties = new List<UniqueProperty>();
            var properties = entity.GetProperties(PublicInstance);
            var tableName = tableInfo.TableName;
            foreach (var prop in properties)
            {
```

```
                if (!prop.PropertyType.IsValueType && prop.PropertyType != typeof(string)) continue;

                UniqueAttribute[] uniqueAttributes =
(UniqueAttribute[])prop.GetCustomAttributes(typeof(UniqueAttribute), true);
                NotMappedAttribute[] notMappedAttributes =
(NotMappedAttribute[])prop.GetCustomAttributes(typeof(NotMappedAttribute), true);
                if (uniqueAttributes.Length > 0 && notMappedAttributes.Length == 0)
                {
                    var fieldName = GetFieldName(context, entity, prop.Name);
                    if (fieldName != null)
                    {
                        indexedProperties.Add(new UniqueProperty
                        {
                            TableName = tableName,
                            IndexName = uniqueAttributes[0].IndexName,
                            FieldName = fieldName,
                            Order = uniqueAttributes[0].Order.ToString()
                        });
                    }
                }
            }
            return indexedProperties;
        }
        private static Dictionary<string, TableInfo> GetTables(DbContext context)
        {
            var tablesInfo = new Dictionary<string, TableInfo>();
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var tables = metadata.GetItemCollection(DataSpace.SSpace)
              .GetItems<EntityContainer>()
              .Single()
              .BaseEntitySets
              .OfType<EntitySet>()
              .Where(s => !s.MetadataProperties.Contains("Type")
                || s.MetadataProperties["Type"].ToString() == "Tables");
            foreach (var table in tables)
            {
                var tableName = table.MetadataProperties.Contains("Table")
                    && table.MetadataProperties["Table"].Value != null
                  ? table.MetadataProperties["Table"].Value.ToString()
                  : table.Name;
                var tableSchema = table.MetadataProperties["Schema"].Value.ToString();
                tablesInfo.Add(table.Name, new TableInfo
                {
                    EntityName = table.Name,
                    TableName = tableName,
                    TableSchema = tableSchema,
                });
            }

            return tablesInfo;
        }
        public static string GetFieldName(DbContext context, Type entityModel, string propertyName)
        {
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var osMembers = metadata.GetItem<EntityType>(entityModel.FullName,
DataSpace.OSpace).Properties;
            var ssMebers = metadata.GetItem<EntityType>("CodeFirstDatabaseSchema." + entityModel.Name,
DataSpace.SSpace).Properties;

            if (!osMembers.Contains(propertyName)) return null;

            var index = osMembers.IndexOf(osMembers[propertyName]);
            return ssMebers[index].Name;
        }

        internal class UniqueProperty
        {
            public string TableName { get; set; }
            public string FieldName { get; set; }
            public string IndexName { get; set; }
            public string Order { get; set; }
        }
        internal class TableInfo
        {
            public string EntityName { get; set; }
            public string TableName { get; set; }
            public string TableSchema { get; set; }
        }
    }
}
```

در کد بالا با استفاده از Extension Method برای کلاس DbContext یک متد با نام ExecuteUniqueIndexes ایجاد می‌کنیم تا برای ایجاد ایندکس‌ها در دیتابیس از آن استفاده کنیم.
روند اجرای کلاس بالا به صورت زیر می‌باشد:
در ابتدای متد ExecuteUniqueIndexes() :

```
public static void ExecuteUniqueIndexes(this DbContext context)
        {
            var tables = GetTables(context);
            ...
        }
```

با استفاده از متد GetTables() ما تمام جداول ساخته توسط دیتایس توسط DbContext را گرفنه:

```
private static Dictionary<string, TableInfo> GetTables(DbContext context)
        {
            var tablesInfo = new Dictionary<string, TableInfo>();
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var tables = metadata.GetItemCollection(DataSpace.SSpace)
              .GetItems<EntityContainer>()
              .Single()
              .BaseEntitySets
              .OfType<EntitySet>()
              .Where(s => !s.MetadataProperties.Contains("Type")
                 || s.MetadataProperties["Type"].ToString() == "Tables");
            foreach (var table in tables)
            {
                var tableName = table.MetadataProperties.Contains("Table")
                    && table.MetadataProperties["Table"].Value != null
                  ? table.MetadataProperties["Table"].Value.ToString()
                  : table.Name;
                var tableSchema = table.MetadataProperties["Schema"].Value.ToString();
                tablesInfo.Add(table.Name, new TableInfo
                {
                    EntityName = table.Name,
                    TableName = tableName,
                    TableSchema = tableSchema,
                });
            }

            return tablesInfo;
        }
```

با استفاده از این طریق چنانچه کاربر نام دیگری برای هر جدول در نظر بگیرد مشکلی ایجاد نمی‌شود و همینطور Schema جدول نیز گرفته می‌شود, سه مشخصه نام مدل و نام جدول و Schema جدول در کلاس TableInfo قرار داده می‌شود و در انتها تمام جداول در یک Collection قرار داده میشوند و به عنوان خروجی متد استفاده می‌شوند.
بعد از آنکه نام جداول متناظر با نام مدل آنها را در اختیار داریم نوبت به گرفتن تمام DbSetها در DbContext می‌باشد که با استفاده از متد GetDbSets() :

```
public static void ExecuteUniqueIndexes(this DbContext context)
        {
            var tables = GetTables(context);
            var query = "";
            foreach (var dbSet in GetDbSets(context))
            {
            ....
        }
```

در این متد چنانچه Property دارای Attribute NotMapped باشد در لیست خروجی متد قرار داده **نمی‌شود.**
سپس داخل چرخه DbSetها نوبت به گرفتن ایندکس‌های موجود با استفاده از متد GetCurrentUniqueIndexes() برای این مدل می‌باشد تا از ایجاد دوباره آن جلوگیری شود و البته اگر ایندکس هایی را در مدل تعریف نکرده باشید از دیتابیس حذف شوند.

```
public static void ExecuteUniqueIndexes(this DbContext context)
        {
```

```
            ...
            foreach (var dbSet in GetDbSets(context))
            {
                var entityType = dbSet.PropertyType.GetGenericArguments().First();
                var table = tables[entityType.Name];
                var currentIndexes = GetCurrentUniqueIndexes(context, table.TableName);
            }
        }
```

بعد از آن نوبت به گرفتن Propertyهای دارای Attribute Unique می‌باشد که این کار نیز با استفاده از متد GetUniqueProperties() انجام خواهد شد.

در متد GetUniqueProperties() چند شرط بررسی خواهد شد از جمله اینکه Property از نوع Value Type باشد و نه یک کلاس سپس Attribute NotMapped را نداشته باشد و بعد از آن می‌بایست نام متناظر با آن Property را در دیتابیس به دست بیاریم برای این کار از متد GetFieldName() استفاده می‌کنیم:

```
public static string GetFieldName(DbContext context, Type entityModel, string propertyName)
        {
            var metadata = ((IObjectContextAdapter)context).ObjectContext.MetadataWorkspace;
            var osMembers = metadata.GetItem<EntityType>(entityModel.FullName,
DataSpace.OSpace).Properties;
            var ssMebers = metadata.GetItem<EntityType>("CodeFirstDatabaseSchema." + entityModel.Name,
DataSpace.SSpace).Properties;

            if (!osMembers.Contains(propertyName)) return null;

            var index = osMembers.IndexOf(osMembers[propertyName]);
            return ssMebers[index].Name;
        }
```

برای این کار با استفاده از MetadataWorkspace در DbContext دو لیست SSpace و OSpace استفاده می‌کنیم که در ادامه در مورد این گونه لیست ها بیشتر توضیح می‌دهیم , سپس با استفاده از Memberهای این دو لیست و ایندکس‌های متناظر در این دو لیست نام متناظر با Property را در دیتابیس پیدا خواهیم کرد, **البته یک نکته مهم هست چنانچه برای فیلدهای دیتابیس OrderColumn قرار داده باشید دو لیست Memberها از نظر ایندکس متناظر متفاوت خواهند شد پس در نتیجه ایندکس به اشتباه برروی یک فیلد دیگر اعمال خواهد شد.**

لیست‌ها در MetadataWorkspace:

1. CSpace : این لیست شامل آبجکت‌های Conceptual از مدل‌های شما می‌باشد تا برای Mapping دیتابیس با مدل‌های شما مانند مبدلی این بین عمل کند.
2. OSpace : این لیست شامل آبجکت‌های مدل‌های شما می‌باشد.
3. SSpace : این لیست نیز شامل آبجکت‌های مربوط به دیتابیس از مدل‌های شما می‌باشد
4. CSSpace : این لیست شامل تنظیمات Mapping بین دو لیست SSpace و CSpace می‌باشد.
5. OCSpace : این لیست شامل تنظیمات Mapping بین دو لیست OSpace و CSpace می‌باشد.

روند Mapping مدل‌های شما از OSpace شروع شده و به SSpace ختم میشود که سه لیست دیگز شامل تنظیماتی برای این کار می‌باشند.

و حالا در متد اصلی ExecuteUniqueIndexes() ما کوئری مورد نیاز برای ساخت ایندکس‌ها را ساخته ایم.

حال برای استفاده از متدExecuteUniqueIndexes() می‌بایست در متد Seed آن را صدا بزنیم تا کار ساخت ایندکس‌ها شروع شود، مانند کد زیر:

```
protected override void Seed(myDbContext context)
        {
            //  This method will be called after migrating to the latest version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension method
            //  to avoid creating duplicate seed data. E.g.
            //
            //    context.People.AddOrUpdate(
            //      p => p.FullName,
            //      new Person { FullName = "Andrew Peters" },
            //      new Person { FullName = "Brice Lambson" },
```

```
            //      new Person { FullName = "Rowan Miller" }
            //    );
            //
            context.ExecuteUniqueIndexes();
        }
```

چند نکته برای ایجاد ایندکس منحصر به فرد وجود دارد که در زیر به آنها اشاره می‌کنیم:

1. فیلدهای متنی باید حداکثر تا 350 کاراکتر باشند تا ایندکس اعمال شود.

2. همانطور که بالاتر اشاره شد برای فیلدهای دیتابیس OrderColumn اعمال نکنید که علت آن در بالا توضیح داده شد

دانلود فایل پروژه:

Sample_UniqueIndex.zip

نظرات خوانندگان

نویسنده: rahimi
تاریخ: ۱۳۹۱/۰۹/۲۳ ۱۶:۳۲

سلام ممنون از آموزش‌های خوبتون
می‌خواستم خواهش کنم ازتون مثال هایی که توضیح دادید را فایلشو هم قرار بدید تا بتونیم استفاده کنیم
ممنون

نویسنده: پدرام جباری
تاریخ: ۱۳۹۱/۰۹/۲۴ ۹:۲۷

سلام
خواهش می‌کنم
فایل پروژه به انتهای آموزش اضافه شد.

نویسنده: آرش مصیر
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۶:۶

با تشکر از سایت خوبتون من چند ماه پیش به این مشکل بر خورده بودم و در متد Seed مربوط به Context مستقیما اسکریپت ساخت ایندکس رو گذاشته بودم حالا می‌خوام از روشی که گفتید استفاده کنم

نویسنده: اکبر
تاریخ: ۱۳۹۲/۰۷/۱۲ ۲۱:۱۰

با سلام.
وقتی از این اتریبیوت بر روی پراپرتی email استفاده میکنم، و چون مقدار این فیلد الزامی نیست، وقتی کاربر این فیلد را خالی بگذارد خطای زیر را دریافت میکنم.

```
Violation of UNIQUE KEY constraint 'IX_Unique_Members_Email'. Cannot insert duplicate key in object
'dbo.Members'
```

با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۱۲ ۲۱:۲۸

از چه دیتابیسی استفاده می‌کنید؟ اگر SQL Server است که تا قبل از نگارش 2008 آن چنین اجازه‌ای رو به شما نمی‌ده تا یک فیلد منحصربفرد نال پذیر داشته باشید. اگر 2008 به بعد است، باید ایندکس فیلتر شده برای اینکار تعریف کنید. مثلا:

```
create unique nonclustered index idx on dbo.DimCustomer(emailAddress)
where EmailAddress is not null;
```

اطلاعات بیشتر اینجا و اینجا
بر همین مبنا باید قسمت ADD CONSTRAINT متد ExecuteUniqueIndexes را در صورت نیاز بازنویسی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۷ ۲۳:۱۳

یک نکته‌ی تکمیلی

از EF 6.1 به بعد ، دیگر نیازی به این مطلب نیست. تعریف ایندکس به صورت توکار میسر شده‌است .

عنوان: افزودن خودکار کلاس‌های تنظیمات نگاشت‌ها در EF Code first
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۰ ۰:۲۱
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

اگر از روش Fluent-API برای تنظیم و افزودن نگاشت‌های کلاس‌ها استفاده کنیم، با زیاد شدن آن‌ها ممکن است در این بین، افزودن یکی فراموش شود یا کلا اضافه کردن دستی آن‌ها در متد OnModelCreating آنچنان جالب نیست. می‌شود این‌کار را به کمک Reflection ساده‌تر و خودکار کرد:

```
void loadEntityConfigurations(Assembly asm, DbModelBuilder modelBuilder, string nameSpace)
        {
            var configurations = asm.GetTypes()
                                    .Where(type => type.BaseType != null &&
                                           type.Namespace == nameSpace &&
                                           type.BaseType.IsGenericType &&
                                           type.BaseType.GetGenericTypeDefinition() ==
typeof(EntityTypeConfiguration<>))
                                    .ToList();

            configurations.ForEach(type =>
               {
                   dynamic instance = Activator.CreateInstance(type);
                   modelBuilder.Configurations.Add(instance);
               });
        }
```

در این متد، در یک اسمبلی مشخص و فضای نامی در آن، به دنبال کلاس‌هایی از نوع EntityTypeConfiguration خواهیم گشت. در ادامه این کلاس‌ها وهله سازی شده و به صورت خودکار به modelBuilder اضافه می‌شوند.

یک مثال کامل که بیانگر نحوه استفاده از متد فوق است:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Data.Entity.ModelConfiguration;
using System.Linq;
using System.Reflection;

namespace EFGeneral
{
    public class User
    {
        public int UserNumber { get; set; }
        public string Name { get; set; }
    }

    public class UserConfig : EntityTypeConfiguration<User>
    {
        public UserConfig()
        {
            this.HasKey(x => x.UserNumber);
            this.Property(x => x.Name).HasMaxLength(450).IsRequired();
        }
    }

    public class MyContext : DbContext
    {
        public DbSet<User> Users { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // modelBuilder.Configurations.Add(new UserConfig());

            var asm = Assembly.GetExecutingAssembly();
            loadEntityConfigurations(asm, modelBuilder, "EFGeneral");
        }

        void loadEntityConfigurations(Assembly asm, DbModelBuilder modelBuilder, string nameSpace)
        {
            var configurations = asm.GetTypes()
                                    .Where(type => type.BaseType != null &&
```

```
                                            type.Namespace == nameSpace &&
                                            type.BaseType.IsGenericType &&
                                            type.BaseType.GetGenericTypeDefinition() ==
typeof(EntityTypeConfiguration<>))
                                    .ToList();

            configurations.ForEach(type =>
               {
                   dynamic instance = Activator.CreateInstance(type);
                   modelBuilder.Configurations.Add(instance);
               });
        }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            context.Users.Add(new User { Name = "name-1" });
            context.Users.Add(new User { Name = "name-2" });
            context.Users.Add(new User { Name = "name-3" });
            base.Seed(context);
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var context = new MyContext())
            {
                var user1 = context.Users.Find(1);
                if (user1 != null)
                    Console.WriteLine(user1.Name);
            }
        }
    }
}
```

در این مثال، در متد OnModelCreating بجای اضافه کردن دستی تک تک تنظیمات تعریف شده، از متد loadEntityConfigurations جهت یافتن آن‌ها در اسمبلی جاری و فضای نام مشخصی به نام EFGeneral استفاده شده است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۳ ۱۳:۵۹

یک نکته تکمیلی

در EF 6 متد ذیل جهت پوشش این مطلب اضافه شده‌است:

```
modelBuilder.Configurations
      .AddFromAssembly(Assembly.GetExecutingAssembly())
```

نویسنده: سدا
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۳:۴۵

اگر نگاشت در مسیر DomainClasses.Mappings باشه چه تغییری باید ایجاد کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۳:۵۳

از AppDomain برای یافتن اسمبلی مدنظر استفاده کنید:

```
var enumerable = AppDomain.CurrentDomain.GetAssemblies()
        .SelectMany(assembly => assembly.GetTypes())
        .Select(type => type.Namespace)
        .Distinct()
        .Where(name => name != null &&
                       name == "DomainClasses.Mappings");
```

نویسنده: سدا
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۶:۵۶

در حالت اینکه از BaseEntity استفاده شه مشکلی نیست . ولی یه نوع وابستگی میشه... درسته؟
من در حالت BaseEntity جواب گرفتم. اما حالتی که فضای نام رو جستجو کنه خیلی منطقی‌تر و انعطاف پذیرتره.
ولی دقیقا شکل پیاده سازیش به مشکل برخوردم از Dbset خودکار که تو سایت هست استفاده کردم اما موفق نشدم.

امکان داره بگید زمانی که تمامی اسمبلی‌ها با دستور فوق در متغیر تعریف شده ذخیره شدن چطور در ModelBinder ست کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۷:۹

- کمی بالاتر عنوان شد: modelBuilder.Configurations.AddFromAssembly
- مطلب فوق در مورد کلاس‌های مشتق شده از EntityTypeConfiguration نوشته شد. این کلاس‌ها در EF همیشه به همین شکل هستند؛ مگر اینکه طراحی کل آن تغییر کند.
- در مطلب دیگری که به خودکار کردن تعاریف DbSetها اختصاص داشت، سطر ذیل را از آن مطابق نیاز خودتان، حذف کنید:

```
type.BaseType == typeof(BaseEntity)
```

بررسی فضای نام را هم دارد.

عنوان: خودکار کردن تعاریف DbSetها در EF Code first
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۷ ۲۱:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

پیشنیاز:

تعریف نوع جنریک به صورت متغیر

مطلبی را چندی قبل در مورد نحوه خودکار کردن افزودن کلاس‌های EntityTypeConfiguration به modelBuilder در این سایت مطالعه کردید . در مطلب جاری به خودکار سازی تعاریف مرتبط با DbSetها خواهیم پرداخت.

ابتدا مثال کامل زیر را درنظر بگیرید:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;
using System.Reflection;

namespace MyNamespace
{
    public abstract class BaseEntity
    {
        public int Id { set; get; }
        public string CreatedBy { set; get; }
    }

    public class User : BaseEntity
    {
        public string Name { get; set; }
    }

    public class MyContext : DbContext
    {
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            var asm = Assembly.GetExecutingAssembly();
            loadEntities(asm, modelBuilder, "MyNamespace");
        }

        void loadEntities(Assembly asm, DbModelBuilder modelBuilder, string nameSpace)
        {
            var entityTypes = asm.GetTypes()
                                    .Where(type => type.BaseType != null &&
                                           type.Namespace == nameSpace &&
                                           type.BaseType.IsAbstract &&
                                           type.BaseType == typeof(BaseEntity))
                                    .ToList();

            var entityMethod = typeof(DbModelBuilder).GetMethod("Entity");
            entityTypes.ForEach(type =>
            {
                entityMethod.MakeGenericMethod(type).Invoke(modelBuilder, new object[] { });
            });
        }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            context.Set<User>().Add(new User { Name = "name-1" });
            context.Set<User>().Add(new User { Name = "name-2" });
            context.Set<User>().Add(new User { Name = "name-3" });
            base.Seed(context);
        }
    }

    public static class Test
```

```
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var context = new MyContext())
            {
                var user1 = context.Set<User>().Find(1);
                if (user1 != null)
                    Console.WriteLine(user1.Name);
            }
        }
    }
}
```

توضیحات:

همانطور که ملاحظه می‌کنید در این مثال خبری از تعاریف DbSetها نیست. به کمک Reflection تمام مدل‌های برنامه که از نوع کلاس پایه BaseEntity هستند (روشی مرسوم جهت مدیریت خواص تکراری مدل‌ها) یافت شده (در متد loadEntities) و سپس نتیجه حاصل به صورت پویا به متد جنریک Entity ارسال می‌شود. حاصل، افزوده شدن خودکار کلاس‌های مورد نظر به سیستم EF است.

البته در این حالت چون دیگر کلاس‌های مدل‌ها در MyContext به صورت صریح تعریف نمی‌شوند، نحوه استفاده از آن‌ها را توسط متد Set، در متدهای RunTests و یا Seed، ملاحظه می‌کنید.

نظرات خوانندگان

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۸/۲۷ ۲۳:۴۴

البته کد

```
var asm = Assembly.GetExecutingAssembly();
```

در صورتی کار میکنه که مدل‌ها در همین پروژه باشند ولی اگر در یک پروژه جداگانه تعریف شده باشند باید از

```
var asm = Assembly.GetAssembly(typeof(DomainModels.BaseEntity));
```

استفاده کرد . که DomainModels.BaseEntity یکی از کلاس‌های موجود در اسمبلی مربوط به مدلها باید باشد .

نویسنده: ali mazinani
تاریخ: ۱۳۹۱/۰۸/۲۸ ۱۴:۳۹

سلام آیا این کار سربار اضافی نداره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۸ ۱۶:۱۴

نه. اینکار فقط یکبار در آغاز برنامه انجام می‌شود. نتیجه کار هم توسط EF کش خواهد شد.

نویسنده: hoseini
تاریخ: ۱۳۹۱/۱۲/۰۴ ۲۳:۲۴

سلام
این متد کار افزودن نگاشت‌های کلاس‌ها رو هم انجام میده ؟
و اگر بخواد انجام بده باید چه تغییراتی روی کد انجام بشه؟
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۴ ۲۳:۳۶

« افزودن خودکار کلاس‌های تنظیمات نگاشت‌ها در EF Code first »

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۵:۵۱

سلام
اگر مدلها در چندین پروژه بودن چطور؟
مثلا به صورت ماژول هر ماژول مدل خود را دارد

نویسنده: وحید نصیری

تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۷:۱۷

الف)
- تمام مدل‌های شما باید از کلاس مشخصی مثلا BaseEntity مشتق شوند.
- یا در تنظیمات برنامه مشخص کنید در حین Reflection چه فضای نامی باید جستجو شود.
- و یا مثلا مانند ASP.NET MVC اگر کلاسی نامش به عبارت خاصی ختم شد و همچنین از کلاس پایه خاصی نیز مشتق شده بود آنگاه بررسی شود.
ب)
- مرحله بعد اندکی ویرایش متد loadEntities است جهت خواندن مدل‌های واقع شده مثلا در یک فضای نام خاص یا مشتق شده از یک کلاس پایه خاص و سپس افزودن خودکار آن‌ها به modelBuilder همانند چیزی که در مثال فوق پیاده سازی شده.
در مثال بالا فقط یک اسمبلی جستجو می‌شود؛ اگر نیاز است تمام اسمبلی‌های بارگذاری شده در برنامه جستجو شوند، روش کار مراجعه به AppDomain است:

```
foreach (Assembly currentassembly in AppDomain.CurrentDomain.GetAssemblies())
{
   Type t = currentassembly.GetType("typeName", false, true);
    if (t != null) {return currentassembly.FullName;}
}
```

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۸:۱۱

ممنونم از جوابتون

AppDomain.CurrentDomain.GetAssemblisl اسمبلی‌ها را از پوشه Bin اضافه می‌نماید یا اینکه جستجویی در کل Libaryها انجام می‌دهد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۸:۲۰

خیر. چیزی را اضافه یا بارگذاری نمی‌کند. فقط در AppDomain برنامه، کل اسمبلی‌ها و ماژول‌های بارگذاری شده موجود را لیست می‌کند .

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۸:۲۳

سلام
من به این طریق Dbsetها رو اضافه میکنم و مشکلی پیش نمیاد . اما وقتی میخوام نگاشت‌ها را به وسیله توضیحات این لینک انجام بدم به مشکل میخورم
من ابتدا Dbsetها را با این کد شما اضافه میکنم سپس تابع اضافه کردن نگاشت‌ها را اجرا میکنم
اما به مشکل میخورم
ممنون میشم اگه راهنمایی نمایید به چه شکل باید هم موجودیت‌ها و هم نگاشت‌های آن‌ها را همزمان انجام داد
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۹:۱

عکس عمل کنید. ابتدا تنظیمات، بعد موجودیت‌ها اضافه شوند.
ضمنا اگر احیانا گذارتان به انجمنی افتاد این «به مشکل برخوردم» رو باید توضیح بدید. باید خطای حاصل رو ذکر کنید (علاوه بر روشی که طی شده).

نویسنده: مهدی فرهانی
تاریخ: ۱۳۹۲/۰۱/۰۹ ۱۳:۵۹

با سلام
چندوقتی هست که دارم از این روش برای برنامه هام استفاده میکنم و هیچ مشکلی نداشتم تا اینکه دیشب با یک مشکل عجیب برخورد کردم و مشکل این بود که زمان ساخت جداول به Property که به صورت NotMaped در کلاس Base تعریف کرده بودم ایراد گرفت.پس از بررسی که انجام دادم متوجه شدم که ترتیب Map کردن Entityها باعث این مشکل میشه.
http://entityframework.codeplex.com/workitem/481

این لینک مشکلی که گزارش شده، راه حلی که نوشه شده این که ترتیب Map کردن کلاس‌ها باید تغییر کند.این درحالی است که در این روش هیچ ترتیبی در نظر گرفته نمیشه و براساس خواندن کلاس‌ها از اسمبلی ساخته میشه.
برای رفع این مشکل اومدم و یک ویژگی ترتیب به کلاسهایی که میدونستم ترتیبشون مهم هست اضافه کردم و زمان خواندن کلاس‌ها از اسمبلی مورد نظر اونها مرتب کردم و مشکل حل شد.
آیا راه حل بهتری وجود داره یا نه ؟

```
var entityTypes = modelAssembly.GetTypes()
                                    .Where(type => type.Namespace != null
                                        && (type.Namespace.StartsWith(domainNamespace)
                                            && type.CustomAttributes.All(c => c.AttributeType != 
typeof(ComplexTypeAttribute))
                                            && !type.IsAbstract && !type.IsEnum)
                                        )
                                    .OrderBy(c => c.CustomAttributes.Any(
                                        x => x.AttributeType == typeof (EntityOrderAttribute)) ? 
c.GetCustomAttribute<EntityOrderAttribute>().OrderNumber : 100).ToList();
```

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۰ ۹:۹

این مساله در حالت معمولی تعریف DbSetها هم پیش میاد. یعنی الزاما محدود به روش گفته شده در این مطلب نیست. خودشون هم پذیرفتن که یک باگ است و در حال رفعش هستند.

نویسنده: پدرام جباری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۱:۲۷

با سلام

من برای پروژه ای که در حال انجام اون هستم چون تعداد جداول زیاد هست برای آنکه زمان نمونه ساختن هر context کاهش پیدا کنه، برای هر بخش از پروژه Context متفاوتی ایجاد کردم تا زمان نگاشت تنظیمات جداول کاهش پیدا کنه ، البته یک Context کلی هست که اون وظیفه ساخت دیتابیس رو داره (داخل EF 6 این امکان هست که یک دیتابیس از چند Context متفاوت ایجاد شده باشه ، ولی خوب برای این کار تو EF5 باید یک Context کلی داشت که مشکلی ایجاد نشه)
به نظر شما کار درستی هست ؟ اصلا تاثیری داره ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۱:۵۳

- بحث جاری در مورد EF 5 هست. کل مباحثی که در سایت مطرح شده در مورد تا قبل از EF 6 است . (هرچند هدف اصلی EF6 از بحث چند Context ایی پوشش دادن ایجاد جداول مختلف به ازای Schemaهای مختلف است. پیشتر dbo بیشتر مد نظر بود (پشتیبانی از تک schema به ازای یک دیتابیس). الان پوشش چندین Schema با هم در طی چندین Context مختلف در یک بانک اطلاعاتی)
- تاثیری نداره. نگاشت‌ها فقط یکبار در آغاز کار برنامه انجام می‌شوند و بعد کش خواهند شد. هر بار وهله سازی context به

معنای انجام چند باره نگاشت‌ها و یافتن و برقراری آن‌ها نیست. اتفاقا این وهله سازی‌های ثانویه پس از آغاز برنامه، فوق العاده هم سریع هستند.
خلاصه اینکه مباحث مطرح شده در مطلب جاری فقط در آغاز برنامه اجرا می‌شوند و نه به ازای هر بار وهله سازی تک Context برنامه.
- در مورد اینکه چرا باید یک کلاس Context در برنامه داشت اینجا توضیح دادم . بحث الگوی واحد کار مهم‌ترین آن‌ها است.

نویسنده: پدرام جباری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۳:۴۳

ممنون بابت جوابتون، واقعا نمی‌دونم چرا همیشه فکر می‌کردم به این صورت نیست و هر بار بخش نگاشت‌ها انجام میشه، تقریبا بیشتر مقاله‌های مربوط به افزایش سرعت رو در EF رو در وبسایت مطالعه کردم و کل روند انجام کار مفهوم شد برام، شاید بیشتر به خاطر عدم اطمینانم به EF بود که همچین فکری کردم.
واقعا ممنون بابت مطالبتون که در وبسایت قرار دادید خیلی به من کمک کرده از همه نظر.

نویسنده: بهنام حقی
تاریخ: ۱۳۹۳/۰۱/۲۸ ۱۲:۳۶

سلام
من یه مشکلی دارم توی این قسمت.
توی پروژه از خودکار سازی نگاشت‌ها و همین بخش استفاده کردم که در نهایت پیغام خطای زیر رو میده:

```
The entity type User is not part of the model for the current context.
```

قسمت نگاشت‌ها همانند همین که تو سایت گفتین استفاده کردم. پروژه شامل سه بخش Domain و Model و Console هست برای تست این دو بخش. قسمت خودکار سازی نگاشت‌ها بدون خود کارسازی تعاریف DbSetها به درستی کار میکنه، وقتی این بخش رو پیاده میکنم پیغام خطای بالا رو میده تو متد Seed واسه دیتاهای پیش فرض User.

```
namespace Test.Domain
{
    public abstract class BaseEntity
    {
        public int Id { set; get; }
    }
}

namespace Test.Domain.Entities
{
    public class User : BaseEntity
    {
        public int UserNumber { get; set; }
        public string Name { get; set; }
    }
}

namespace Test.Domain.Mappings
{
    public class UserMap : EntityTypeConfiguration<User>
    {
        public UserMap()
        {
            this.HasKey(x => x.UserNumber);
            this.Property(x => x.Name).HasMaxLength(450).IsRequired();
        }
    }
}
```

```
loadEntities(asm, modelBuilder, "Test.Domain");
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۸ ۱۲:۵۱

درسته. چون کلاس User در فضای نام Test.Domain قرار ندارد.
بررسی انجام شده به صورت type.Namespace == nameSpace است ولی حالت مدنظر شما type.Namespace. **StartsWith** nameSpace است.

نویسنده: بهنام حقی
تاریخ: ۱۳۹۳/۰۱/۲۸ ۱۳:۱۱

خیلی ممنون. برای اینکه همه اسمبلی هارو بگرده، و یه اسمبلی خاص رو بهش معرفی نکنیم، چطور از این تابع استفاده کنم:

```
foreach (Assembly asm in AppDomain.CurrentDomain.GetAssemblies())
{
  //...
}
```

تابع زیر رو به ازای هر فضای نام که میده باید فراخونی کنم؟

```
loadEntities(asm, modelBuilder, "نام فضا");
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۸ ۱۳:۲۲

بله. ولی این حالت مقرون به صرفه نیست. چون تعداد اسمبلی‌های بارگذاری شده در یک App domain زیاد است و شامل موارد پایه و سیستمی دات نت هم هست (با هزاران فضای نام). در این حالت این جستجو زمان زیادی را به خود اختصاص خواهد داد. بهتر است لیست یک سری اسمبلی مشخص را درنظر داشته باشید تا اینکه کل سیستم را جستجو کنید.

نویسنده: سدا
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۷:۱۵

اگر بخوایم خودکار Dbsetها تعریف شه و Models و Mapping در پروژه ای دیگه باشه، متود Seed که نیاز مستقیم به Context و Dbsetها داره. یعنی بر فرض بخوایم یک User به Context.Users اضافه کنیم امکان پذیر نیست درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۲۱ ۱۷:۲۲

از متد Set استفاده کنید. در مثال فوق در متد protected override void Seed نمونه‌اش ارائه شده:

```
context.Set<User>().Add(new User { Name = "name-1" });
```

عنوان: بررسی خطای cycles or multiple cascade paths و یا cyclical reference در EF Code first
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۸ ۱۳:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

ابتدا مثال کامل این قسمت را با شرح زیر درنظر بگیرید؛ در اینجا هر کاربر، یک کارتابل می‌تواند داشته باشد (رابطه یک به صفر یا یک) و تعدادی سند منتسب به او (رابطه یک به چند). همچنین روابط بین کارتابل و اسناد نیز چند به چند است:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Data.Entity.ModelConfiguration;

namespace EF_General.Models.Ex18
{
    public class UserProfile
    {
        public int UserProfileId { set; get; }
        public string UserName { set; get; }

        [ForeignKey("CartableId")]
        public virtual Cartable Cartable { set; get; } // one-to-zero-or-one
        public int? CartableId { set; get; }

        public virtual ICollection<Doc> Docs { set; get; } // one-to-many
    }

    public class Doc
    {
        public int DocId { set; get; }
        public string Title { set; get; }
        public string Body { set; get; }

        [ForeignKey("UserProfileId")]
        public virtual UserProfile UserProfile { set; get; }
        public int UserProfileId { set; get; }

        public virtual ICollection<Cartable> Cartables { set; get; } // many-to-many
    }

    public class Cartable
    {
        public int CartableId { set; get; }

        [ForeignKey("UserProfileId")]
        public virtual UserProfile UserProfile { set; get; }
        public int UserProfileId { set; get; }

        public virtual ICollection<Doc> Docs { set; get; } // many-to-many
    }

    public class UserProfileMap : EntityTypeConfiguration<UserProfile>
    {
        public UserProfileMap()
        {
            this.HasOptional(x => x.Cartable)
                .WithRequired(x => x.UserProfile)
                .WillCascadeOnDelete();
        }
    }

    public class MyContext : DbContext
    {
        public DbSet<UserProfile> UserProfiles { get; set; }
        public DbSet<Doc> Docs { get; set; }
        public DbSet<Cartable> Cartables { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new UserProfileMap());
            base.OnModelCreating(modelBuilder);
        }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
```

```
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var context = new MyContext())
            {
                var user = context.UserProfiles.Find(1);
                if (user != null)
                    Console.WriteLine(user.UserName);
            }
        }
    }
}
```

اگر این مثال را اجرا کنیم، به خطای ذیل برخواهیم خورد:

```
Introducing FOREIGN KEY constraint 'FK_DocCartables_Cartables_Cartable_CartableId'
on table 'DocCartables' may cause cycles or multiple cascade paths. Specify
ON DELETE NO ACTION or ON UPDATE NO ACTION, or modify other FOREIGN KEY constraints.
Could not create constraint. See previous errors.
```

علت اینجا است که EF به صورت پیش فرض ویژگی cascade delete را برای حالات many-to-many و یا کلیدهای خارجی غیرنال پذیر اعمال می‌کند.
این دو مورد در کلاس‌های Doc و Cartable با هم وجود دارند که در نهایت سبب بروز circular cascade delete (حذف آبشاری حلقوی) می‌شوند و بیشتر مشکل SQL Server است تا EF؛ از این لحاظ که SQL Server در این حالت نمی‌تواند در مورد نحوه حذف خودکار رکوردهای وابسته درست تصمیم‌گیری و عمل کند. برای رفع این مشکل تنها کافی است کلید خارجی تعریف شده در دو کلاس Doc و کارتابل را nullable تعریف کرد تا cascade delete اضافی پیش فرض را لغو کند:

```
public int? UserProfileId { set; get; }
```

راه دیگر، استفاده از تنظیمات Fluent و تنظیم WillCascadeOnDelete به false است که به صورت پیش فرض در حالات ذکر شده (روابط چند به چند و یا کلید خارجی غیرنال پذیر)، true است.

شبیه به همین خطا نیز زمانی رخ خواهد داد که در یک کلاس حداقل دو کلید خارجی تعریف شده باشند:

```
The referential relationship will result in a cyclical reference that is not allowed. [ Constraint name
=  ]
```

در اینجا نیز با نال پذیر تعریف کردن این کلیدهای خارجی، خطای cyclical reference برطرف خواهد شد.

نظرات خوانندگان

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۷/۰۸ ۱۶:۲۲

سلام اقای نصیری . من دقیقا همین مشکل و خطا رو دارم با این تفاوت که نمیخوام CasCade delete رو غیر فعال کنم . در واقع من یه کلاس دارم که یکی از properties هاش یه لیستی از خود همین کلاس هست . یعنی به صورت تو در تو با خودش رابطه داره . مثله یه tree View . حالا من میخوام با حذف پدر تمام فرزندان و فزندان فرزندان و ... تا پایین همشون دیلیت بشن . یعنی همون on delete cascade خودمون . ولی خب اینجا میشه همونی که گفتین . حذف ابشاری حلقوی . این کار بخوام بکنم باید چیکار کنم . ؟ ممنون از لطفتون . منتظر پاسخم...

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۰۸ ۱۷:۳۴

مراجعه کنید به آخرین نظرات مطلب « مباحث تکمیلی مدل‌های خود ارجاع دهنده در EF Code first »

نویسنده: دانش پژوه
تاریخ: ۱۳۹۳/۱۰/۰۸ ۱۱:۱۶

با سلام

من با چنین مشکلی بر خوردم و به این آدرس رسیدم که از روش زیر استفاده می‌کرد:

```
modelBuilder.Entity<Employee>()
                    .HasRequired(e => e.SecondTeam)
                    .WithMany(t => t.SecondEmployees)
                    .HasForeignKey(e => e.FirstTeamId)
                    .WillCascadeOnDelete(false);
```

که متاسفانه متوجه نشدم. اگه زحمتی نبود در مورد این روش هم کمی توضیح بدید.

ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۸ ۱۲:۵

در متن عنوان شد «راه دیگر، استفاده از تنظیمات Fluent و تنظیم WillCascadeOnDelete به **false** است که به صورت پیش فرض در حالات ذکر شده (روابط چند به چند و یا کلید خارجی غیرنال پذیر)، true است. » « از این لحاظ که SQL Server در این حالت (**true** بودن حذف آبشاری) نمی‌تواند در مورد نحوه حذف خودکار رکوردهای وابسته درست تصمیم‌گیری و عمل کند »

عنوان: **بالا بردن سرعت بارگذاری اولیه EF Code first با تعداد مدل‌های زیاد**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۹ ۱۹:۸
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

EF Code first هربار در حین آغاز اجرای برنامه و اولین کوئری که به بانک اطلاعاتی ارسال می‌کند، کار تشخیص روابط بین کلاس‌ها و همچنین نگاشت آن‌ها را به بانک اطلاعاتی، انجام می‌دهد. این مورد شاید با تعداد کم کلاس‌ها آنچنان به نظر نرسد، اما اگر تعداد کلاس‌های شما به بالای 200 عدد رسید، زمان آغاز برنامه آزار دهنده خواهد شد. راه حلی برای این مساله وجود دارد به نام ایجاد Viewهای متناظر با نگاشت‌ها و سپس کامپایل آن به عنوان جزئی از برنامه، که در ادامه نحوه انجام این‌کار را مرور خواهیم کرد.

بررسی ساختار pre-generated views

برای کامپایل نگاشت‌های EF در خود برنامه (بجای تولید پویای هربار آن‌ها)، ابتدا باید فایل edmx متناظر با مدل‌ها و روابط بین آن‌ها تشکیل شود:

```
var ms = new MemoryStream();
    using (var writer = XmlWriter.Create(ms))
    {
        EdmxWriter.WriteEdmx(new Context(), writer);
    }
```

پس از اینکه edmx تشکیل شد، باید از ساختار فشرده آن سه جزء زیر را استخراج کرد:

الف) storageModels : ssdl

ب) conceptualModels : csdl

ج) mappings : msl

اینکار را به صورت زیر می‌توان انجام داد:

```
var xDoc = XDocument.Load(ms);

    var ssdl = xDoc.Descendants("{http://schemas.microsoft.com/ado/2009/02/edm/ssdl}Schema").Single();
    var csdl = xDoc.Descendants("{http://schemas.microsoft.com/ado/2008/09/edm}Schema").Single();
    var msl =
xDoc.Descendants("{http://schemas.microsoft.com/ado/2008/09/mapping/cs}Mapping").Single();
```

پس از آن باید محتوای این سه جزء را توسط متد Save هر کدام، در فایل‌های xml ایی ذخیره کرد و توسط ابزاری به نام EdmGen.exe که جزئی از ویژوال استودیو است، فایل Context.Views.cs را تولید، به برنامه اضافه و سپس کامپایل کرد:

```
EdmGen.exe /mode:ViewGeneration /incsdl:Context.csdl  /inmsl:Context.msl /inssdl:Context.ssdl
/outviews:Context.Views.cs
```

بهتر است این پروسه هر بار که قرار است ارائه نهایی برنامه صورت گیرد، انجام شود.

علاوه بر این‌ها اگر علاقمند باشید که کار فایل EdmGen را شبیه سازی کنید، کلاس زیر این‌کار را انجام داده و قادر است خروجی vb یا cs متناظری را نیز تولید کند:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Design;
using System.Data.Entity.Infrastructure;
using System.Data.Mapping;
using System.Data.Metadata.Edm;
using System.IO;
using System.Linq;
```

```
using System.Xml;
using System.Xml.Linq;

namespace EfUtils
{
    public static class PreGeneratedViewsWriter
    {
        public static void CreatePreGeneratedViewsFile(
                this DbContext contextInstance,
                LanguageOption language = LanguageOption.GenerateCSharpCode,
                string viewsFile = "Context.Views.cs",
                string edmxFile = "context.edmx",
                string ssdlFile = "context.ssdl.xml",
                string csdlFile = "context.csdl.xml",
                string mslFile = "context.msl.xml")
        {
            using (var contextViewsMemoryStream = new MemoryStream())
            {
                using (var edmxMemoryStream = new MemoryStream())
                {
                    var edmx = createEdmx(contextInstance, edmxFile, edmxMemoryStream);
                    var mappingItemCollection = createMappingItemCollection(ssdlFile, csdlFile,
mslFile, edmx);
                    generateViews(language, viewsFile, contextViewsMemoryStream,
mappingItemCollection);
                }
            }
        }

        private static void generateViews(LanguageOption language, string viewsFile, MemoryStream
contextViewsMemoryStream, StorageMappingItemCollection mappingItemCollection)
        {
            var viewGenerator = new EntityViewGenerator // It's defined in
System.Data.Entity.Design.dll
            {
                LanguageOption = language
            };
            using (var streamWriter = new StreamWriter(contextViewsMemoryStream))
            {
                var errors = viewGenerator.GenerateViews(mappingItemCollection, streamWriter).ToList();

                if (errors.Any())
                    throw new InvalidOperationException(errors.First().Message);

                contextViewsMemoryStream.Position = 0;
                using (var reader = new StreamReader(contextViewsMemoryStream))
                {
                    var codeData = reader.ReadToEnd();
                    File.WriteAllText(viewsFile, codeData);
                }
            }
        }

        private static StorageMappingItemCollection createMappingItemCollection(string ssdlFile, string
csdlFile, string mslFile, XDocument edmx)
        {
            var ssdl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2009/02/edm/ssdl}Schema").Single();
            ssdl.Save(ssdlFile);
            var storeItemCollection = new StoreItemCollection(new[] { ssdl.CreateReader() });

            var csdl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2008/09/edm}Schema").Single();
            csdl.Save(csdlFile);
            var edmItemCollection = new EdmItemCollection(new[] { csdl.CreateReader() });

            var msl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2008/09/mapping/cs}Mapping").Single();
            msl.Save(mslFile);

            var mappingItemCollection = new StorageMappingItemCollection(edmItemCollection,
storeItemCollection, new[] { msl.CreateReader() });
            return mappingItemCollection;
        }

        private static XDocument createEdmx(DbContext contextInstance, string edmxFile, MemoryStream
edmxMemoryStream)
        {
            var settings = new XmlWriterSettings { Indent = true };
            using (var writer = XmlWriter.Create(edmxMemoryStream, settings))
            {
```

```
                EdmxWriter.WriteEdmx(contextInstance, writer);
            }
            File.WriteAllBytes(edmxFile, edmxMemoryStream.ToArray());

            edmxMemoryStream.Position = 0;
            var edmx = XDocument.Load(edmxMemoryStream);
            return edmx;
        }
    }
}
```

در اینجا همان مراحلی که عنوان شد، تکرار می‌شود. فایل edmx متناظر با وهله‌ای از DbContext برنامه، تولید شده و سه جزء آن استخراج می‌شوند. سپس این موارد به EntityViewGenerator موجود در اسمبلی System.Data.Entity.Design.dll ارسال شده و کد نهایی متناظر قابل کامپایل در برنامه تولید می‌گردد.

پس از تولید فایل Context.Views.cs یا Context.Views.vb، آن‌را به پروژه اضافه کنید.

اینبار نحوه استفاده از آن باید به صورت زیر باشد:

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

از این جهت که تمام اطلاعات لازم جهت آغاز کار، در فایل تولیدی Context.Views وجود دارد و اکنون جزئی از فایل اجرایی برنامه است و نیازی به تکرار ساخت مجدد پویای آن نیست.

مرجع:

Entity Framework Code First View Generation Templates On Visual Studio Code Gallery

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۷/۰۹ ۲۰:۳۸

در یکی از پروژه هایی که بجای تعداد 200 جدول رابطه زیادی هم داشت مجبور به استفاده از این روش شدم، البته سبک Model First

نویسنده: Petek
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۱:۴۶

با سلام
آیا امکان این هست که بشه از این در روش DataBase First استفاده کرد . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۲:۰

بله. همین روال رو می‌تونید توسط افزونه « Entity Framework Power Tools » هم انجام بدید (گزینه Generate Views را به منوی کلیک راست بر روی Entity Data Model/*.EDMX اضافه می‌کند).

نویسنده: Petek
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۷:۵۰

با سلام
ممنوم از راهنماییتون خیلی عالیه .
با استفاده از این کار یه view در کنار Model ام ایجاد شد و سرعت بارگذاری برای این اولین بار رو از 10 ثانیه به کمتر از 2 ثانیه تقلیل داد آیا چیز دیگه ای نیاز نیست و نیز من برای ایجاد دیتابیسم با استفاده از Model در اولین واکشی به این صورت عمل می‌کنم آیا مشکلی پیش نمیاد در استفاده از این روش . با تشکر

```
if (!db_Context.DatabaseExists())
      db_Context.CreateDatabase();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۸:۵۵

- خیر. به تنظیم دیگری نیاز ندارد. این کلاس‌ها به صورت خودکار تشخیص داده شده و استفاده می‌شوند. البته به ازای هربار تغییر مدل‌ها نیاز است مجددا تولید شوند.
- اگر روش شما db first است که عنوان کردید، بررسی فوق (ایجاد بانک اطلاعاتی) کار اضافی است. اگر روش code first است، باز هم نیازی نیست چون در حالت خودکار migrations اینکار را انجام می‌دهد.
در کل بهتر است تمام جوانب را بررسی و آزمایش کنید.
کاری که در اینجا انجام می‌شود ایجاد یک cached metadata کامپایل شده است بجای تولید پویای هربار آن (تفاوت مهم و اصلی با روش‌های متداول).

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۱/۲۸ ۰:۵۲

سلام و خسته نباشید
سوالی که من دارم اینکه بعد از ساختن فایل Context.Views.cs چطور باید ازش استفاده کرد . من این فایلو ساختم و کنار Context.cs تو پروژهام گذاشتم . ولی فرق خاصی احساس نمی‌کنم . میشه بیشتر راهنمایی کنید

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۱/۲۸ ۰:۵۸

اگر فرقی احساس نکردید منتظر نگارش بعدی EF باشید. این مورد رو برطرف کردن:

«significantly improved warm up time (view generation), especially for large models»

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۲۷ ۱:۶

به روز شده این اطلاعات برای EF 6
اطلاعات بیشتر

نویسنده: سوین
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۱:۵۰

آیا بعد از ایجاد View برای بالا بردن لود اولیه ، باز هم migration کار میکنه یا نه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۲:۱۱

بله. هم روش دستی migrations و هم روش خودکار در اینجا کار می‌کنند. اینکه در انتهای بحث عنوان شده مقدار را null قرار بدید، برای رسیدن به حداکثر سرعت بارگذاری برنامه است. در این حالت می‌توانید از روش دستی برای انجام migrations استفاده کنید. این نکته در انتهای مطلب پنجم سری EF سایت بحث شده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۰:۵۵

اگر علاقمند باشید که مدیریت تولید این Viewها را خودکار کنید می‌توانید از پروژه Interactive Pre Generated Views استفاده نمائید:

پروژه: Interactive Pre Generated Views for Entity Framework 6
بسته نیوگت
نحوه استفاده: Using Pre-Generated Views Without Having To Pre-Generate Views

نویسنده: امین
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۸:۱۵

باسلام.این کد را در کجا و چطوری در مدل code first استفاده بکنیم؟

```
using (var ctx = new MyContext())
{
    InteractiveViews
        .SetViewCacheFactory(
            ctx,
            new FileViewCacheFactory(@"C:\MyViews.xml"));
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۸:۲۶

در آغاز برنامه (مثلا در application_start برنامه‌های وب و یا ابتدای متد Main برنامه‌های دسکتاپ): « وادار کردن EF Code

« [به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه first](#)

نویسنده: دانش پژوه
تاریخ: ۱۳۹۳/۰۹/۲۹ ۱۴:۲۴

با سلام
میشه خواهش کنم یک نمونه پروژه برای استفاده از این روش بزارید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۲۹ ۱۴:۳۳

نیازی به مثال آنچنانی ندارد. ابتدا بسته‌ی نیوگت این پروژه را نصب کنید:

```
PM> Install-Package EFInteractiveViews
```

بعد یکبار در ابتدای برنامه در اولین کوئری، متد InteractiveViews.SetViewCacheFactory آن‌را فراخوانی کنید. البته بهتر است آن‌را درون یک سینگلتون thread safe قرار دهید. بار اولی که فایل xml آن ایجاد می‌شود زمان خواهد برد. بار دوم اجرای برنامه سریع است.

```
private static bool _isPreGeneratedViewCacheSet;

private void InitializationPreGeneratedViews()
{
   if (_isPreGeneratedViewCacheSet) return;

   var precompiledViewsFilePath = new FileInfo(Assembly.GetExecutingAssembly().Location).DirectoryName
+ @"\EF6PrecompiledViews.xml";
   InteractiveViews.SetViewCacheFactory(this, new FileViewCacheFactory(precompiledViewsFilePath));
   _isPreGeneratedViewCacheSet = true;
}
```

نویسنده: دانش پژوه
تاریخ: ۱۳۹۳/۱۰/۰۲ ۱۰:۳۶

با سلام و ممنون از جوابتون
روشی رو گفتید رفتم گفتم اینجا بزارم هم دیگران استفاده کنند و اگه هم اشتباه کردم بفرمایید اصلاح کنم.
پکیج رو تو پروژه ای که کلاس context هست نصب کردم و تابع زیر رو

```
private static bool _isPreGeneratedViewCacheSet;

private void InitializationPreGeneratedViews()
{
   if (_isPreGeneratedViewCacheSet) return;

   var precompiledViewsFilePath = new FileInfo(Assembly.GetExecutingAssembly().Location).DirectoryName
+ @"\EF6PrecompiledViews.xml";
   InteractiveViews.SetViewCacheFactory(this, new FileViewCacheFactory(precompiledViewsFilePath));
   _isPreGeneratedViewCacheSet = true;
}
```

توی کلاس context گذاشتم بعد از اجرای یک فایل Xml در مسیر
C:\Users\Hadi\AppData\Local\Temp\Temporary ASP.NET Files\root\2781dacc\5d62fdaf\assembly\dl3\142eef19\00077ffc_731ed001

میسازه، البته من بصورت دستی این تابع رو یک بار اجرا کردم و بعد غیرفعالش کردم.
بعد این تابع رو در application_start نوشتم:

```
InteractiveViews
```

```
                .SetViewCacheFactory(ctx, new FileViewCacheFactory(new
FileInfo(Assembly.GetExecutingAssembly().Location).DirectoryName + @"\EF6PrecompiledViews.xml"));
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۲ ۱۰:۳۹

- این فایل باید به ازای هربار تغییر در مدل‌ها دوباره ساخته شود. بنابراین تولید دوباره آن نباید غیرفعال شود.
- Assembly.GetExecutingAssembly().Location برای برنامه‌های دسکتاپ مفید است (یعنی مسیر فایل اجرایی). در برنامه‌های وب، به این مسیر عموما دسترسی ندارید. به همین جهت بهتر است از HttpRuntime.AppDomainAppPath استفاده کنید.

عنوان: **Entity Framework و InnerException**
نویسنده: ایمان محمدی
تاریخ: ۱۳۹۱/۰۵/۲۰ ۱۳:۵۴
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, Exception, Tips

در Entity Framework بیشتر استثناها تودرتو هستند و ما باید تمام استثناها رو بررسی کنیم تا به پیغام اصلی خطا برسیم. با استفاده از تکه کد زیر به راحتی می‌تونیم استثناها رو پیمایش کنیم و متن خطا را مشخص کنیم.

```
catch (Exception ex)
                {
                    StringBuilder errorMsg = new StringBuilder();
                    for (Exception current = ex; current != null; current = current.InnerException)
                    {
                        if (errorMsg.Length > 0)
                            errorMsg.Append("\n");

                        errorMsg.Append(current.Message.Replace("See the inner exception for details.",
string.Empty));
                    }
                    // log
                    errorMsg.ToString();
                }
```

برای استفاده در قسمت‌های مختلف برنامه یک متد الحاقی مانند زیر تعریف می‌کنیم:

```
public static string ExceptionToString(this Exception ex)
        {
            StringBuilder errorMsg = new StringBuilder();
            for (Exception current = ex; current != null; current = current.InnerException)
            {
                if (errorMsg.Length > 0)
                    errorMsg.Append("\n");
                errorMsg.Append(current.Message.
                    Replace("See the inner exception for details.", string.Empty));
            }
            return errorMsg.ToString();
        }
```

```
catch (Exception ex)
                {
                    // log
                    ex.ExceptionToString();
                }
```

نظرات خوانندگان

نویسنده: f.beigirad
تاریخ: ۱۳۹۲/۰۵/۰۴ ۱۵:۱۳

من که نتونستم از کد بالا استفاده کنم.

لینک رو ببینید : http://barnamenevis.org/showthread.php?410767

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۰۴ ۱۸:۳۹

نتونستید یعنی چکار کردید دقیقا؟ در خطای شما هم InnerException -> UpdateException بوده مثل توضیحات بالا. یعنی اول UpdateException صادر میشه، بعد باید محتویات InnerException اون رو بررسی کرد تا به خطای اصلی رسید مثل کدهای فوق.

عنوان: **استثناهایی که باید حین استفاده از EF Code first بررسی شوند**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۳۱ ۱۹:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

سه نوع استثنای مهم ممکن است حین ذخیره سازی تغییرات در EF code first رخ دهند که بررسی جزئیات آن‌ها می‌تواند راهنمای خوبی برای کاربر و همچنین برنامه نویس در عیب یابی سیستم باشد. این استثناءها باید به صورت مستقل و جداگانه بررسی شوند ونه اینکه از حالت عمومی catch Exception استفاده شود.
این سه نوع استثناء شامل موارد DbUpdateConcurrencyException، DbEntityValidationException و DbUpdateException هستند که به صورت خلاصه به شکل زیر باید تعریف شوند:

```
try
{
    context.SaveChanges();
}
catch (DbEntityValidationException validationException)
{
   //...
}
catch (DbUpdateConcurrencyException concurrencyException)
{
   //...
}
catch (DbUpdateException updateException)
{
   //...
}
```

توضیحات تکمیلی

در حالت DbEntityValidationException به جزئیات خطاهای حاصل از اعتبار سنجی اطلاعات خواهیم رسید. برای مثال اگر قرار است طول فیلدی 30 حرف باشد و کاربر 40 حرف را وارد کرده است، نام خاصیت و همچنین پیغام خطای درنظر گرفته شده را دقیقا در اینجا می‌توان دریافت کرد و به نحو مقتضی به کاربر نمایش داد:

```
catch (DbEntityValidationException validationException)
{
    foreach (var error in validationException.EntityValidationErrors)
    {
        var entry = error.Entry;
        foreach (var err in error.ValidationErrors)
        {
            Debug.WriteLine(err.PropertyName + " " + err.ErrorMessage);
        }
    }
}
```

نوع استثنای DbUpdateConcurrencyException به مسایل همزمانی و به روز رسانی یک رکورد توسط دو یا چند کاربر در شبکه مرتبط می‌شود که در قسمت سوم سری EF code first با معرفی ویژگی‌های ConcurrencyCheck و Timestamp در مورد آن بحث شد. در اینجا به کلیه موجودیت‌های تداخل دار توسط خاصیت concurrencyException.Entries خواهیم رسید و همچنین به کمک متد GetDatabaseValues می‌توان موارد جدید ثبت شده مرتبط با این موجودیت تداخل دار را از بانک اطلاعاتی نیز دریافت کرد:

```
catch (DbUpdateConcurrencyException concurrencyException)
{
    //بررسی مورد اول
    var dbEntityEntry = concurrencyException.Entries.First();
    var dbPropertyValues = dbEntityEntry.GetDatabaseValues();
```

```
}
```

و یا کلا ممکن است حین به روز رسانی بانک اطلاعاتی مشکلی رخ داده باشد که در اینجا عموما پیغام حاصل را باید در InnerException تولیدی یافت و همچنین در اینجا لیست موجودیت‌های مشکل دار نیز قابل دریافت و بررسی هستند:

```
catch (DbUpdateException updateException)
{
    if (updateException.InnerException != null)
        Debug.WriteLine(updateException.InnerException.Message);

    foreach (var entry in updateException.Entries)
    {
        Debug.WriteLine(entry.Entity);
    }
}
```

بنابراین بررسی catch exception کلی در EF Code first مناسب نبوده و نیاز است بیشتر به جزئیات ذکر شده، وارد و دقیق شد.

یک نکته:

بهتر است یک کلاس پایه عمومی مشتق شده از DbContext را ایجاد و متد SaveChanges آن را تحریف کرد. سپس سه حالت فوق را به آن اعمال نمود. اکنون می‌توان از این کلاس پایه بارها استفاده کرد بدون اینکه نیازی به تکرار کدهای آن در هرجایی که قرار است از متد SaveChanges استفاده شود، باشد. شبیه به این کار را در قسمت 14 سری EF code first مشاهده نموده‌اید.

نظرات خوانندگان

نویسنده: علی قمشلویی
تاریخ: ۱۳۹۱/۰۳/۳۱ ۲۲:۸

با سلام و تشکر مثل همیشه عالی

نویسنده: Mona
تاریخ: ۱۳۹۱/۰۳/۳۱ ۲۳:۱۶

یک خواهش، بدلیل اینکه سطوح علمی مراجعین متفاوت است، پیشنهاد میکنم در صورت امکان مثل codeproject یک فایل Sample هم برای پست‌ها آپلود شود.
ممنون

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۲:۳۷

آیا می‌توان به جای نمایش یک پیام انگلیسی هنگام وقوع خطای اعتبارسنجی مثلا از

```
[Required(ErrorMessage = "لطفا نام نویسنده را مشخص نمائید")]
```

استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۲:۳۹

بله. اگر از ASP.NET MVC استفاده کنید به صورت خودکار تبدیل به پیغام‌های اعتبار سنجی سمت کاربر هم می‌شود. ولی در کل اثر بررسی سمت سرور هم دارد و همین پیغام‌ها توسط DbEntityValidationException قابل دریافت هستند .

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۲:۵۹

من یک پیام خطا برای یکی از Propertyها تعیین کردم.وقتی می‌خوام به طریق زیر پیام رو دریافت کنم بازم همون خطای انگلیسی رو مشاهده می‌کنم!؟

```
catch (DbEntityValidationException validationException)
                {
                    var errors = validationException.EntityValidationErrors.First();
                    foreach (var propertyError in errors.ValidationErrors)
                    {
                        Console.WriteLine(propertyError.PropertyName + "----" +
propertyError.ErrorMessage);
                    }
                }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۳

شما از متد First اینجا استفاده کردید. به عبارتی فقط اولین مورد گزارش داده شده رو دارید بررسی می‌کنید و از بقیه موارد صرفنظر کردید.

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۵

درسته چونکه میخواستم فقط همین خطا رو که عمدی ایجاد کردم ببینم ولی نشد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۱۴

کد کاملی که من در مطلب فوق نوشتم چنین چیزی رو نشون می‌ده:

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۳۵

```
[MaxLength(2), Required(ErrorMessage = "طول فیلد بیش از حد مجاز است")]
public string ProductName { get; set; }
```

من از MySQL استفاده کردم. میتونه مشکل از این باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۴۰

خیر. علت این است که به ازای تک تک ویژگی‌هایی که تعریف می‌کنید باید ErrorMessage مقدار دهی شود. مثلا در اینجا MaxLength تعریف شده دارای پیغام خطا نیست و آن‌را از پیغام خطای Required دریافت نمی‌کند.

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۴۴

بله همینطوره.باید به این شکل می‌نوشتم.

```
[MaxLength(2, ErrorMessage = "طول فیلد بیش از حد مجاز است")]
```

خیلی ممنون آقای نصیری.

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۶ ۱۹:۳۸

بهتر است یک کلاس پایه عمومی مشتق شده از DbContext را ایجاد و متد SaveChanges آن را تحریف کرد. سپس سه حالت فوق را به آن اعمال نمود. اکنون می‌توان از این کلاس پایه بارها استفاده کرد بدون اینکه نیازی به تکرار کدهای آن در هرجایی که قرار است از متد SaveChanges استفاده شود، باشد.
چطور میشه از این استثناها در override متد savechange استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۶ ۲۱:۲۹

به این شکل کلی

```
try
{
   return base.SaveChanges();
}
catch (DbEntityValidationException validationException)
{
   //...
}
//...
```

نویسنده: محمدرضا
تاریخ: ۱۳۹۱/۰۴/۰۶ ۲۱:۴۴

منظورم اینه که متد savechange فقط مقدار int برمیگردونه.اگر استثنایی رخ داد فقط می‌تونیم یه عدد خاص برگردونیم که نشانه‌ی وقوع استثنا هست.در این صورت چطور می‌تونیم پیام خطاهای اعتبارسنجی رو کنترل کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۶ ۲۱:۵۵

اینکه شما چه کاری با این اطلاعات انجام می‌دید به طراحی سیستم خودتون مرتبط است. یکی یک پیغام popup نمایش می‌ده یکی یک messagebox یکی هم فقط این‌ها رو لاگ می‌کنه برای بررسی بعدی و یک پیغام کلی به کاربر نمایش می‌ده که مشکلی رخ داده.
من این‌ها رو لاگ می‌کنم و پیغام کلی نمایش می‌دم.

نویسنده: حسيني
تاریخ: ۱۳۹۱/۰۸/۰۹ ۲۲:۲۳

بسم الله الرحمن الرحیم
با سلام
چه جوری میشه بعد از درآوردن اعتبارسنجی‌های مربوط به تک تک پراپرتی‌ها پیام اونها رو در جلوی تکس باکس مربوطه نوشت.
راهی که به ذهن بنده میرسه به شکل زیر است:

```
ICollection<DbValidationError> ValidationProperty =
                                context.Entry(product).Property(p => p.Name).GetValidationErrors();


                            if (ValidationProperty.Count() > 0)
                            {

                                    DbValidationError Error=ValidationProperty.First();
                                    errorProvider1.SetError(txtName, Error.ErrorMessage);
                            }
```

خواستم ببینم راه بهترو سریعتری وجود داره؟

با سپاس فراوان

شب خوش

نویسنده: سارا کیانی
تاریخ: ۱۳۹۳/۰۱/۲۷ ۱۵:۱۳

سلام

چنانچه قصد داشته باشیم در Context خودمان استثنای Connect به سرویس دهنده رو هم در نظر گرفته تا در صورت قطع ارتباط، پیغام خاصی به کاربر نمایش دهیم با کدام Exception در Code First تشخیص داده خواهد شد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۷ ۱۵:۳۰

مانند دوران ADO.NET است:

```
catch (System.Data.SqlClient.SqlException ex)
{
            foreach (SqlError error in ex.Errors)
            {
                switch (error.Number)
                {
                    case 1205:
                        System.Diagnostics.Debug.WriteLine("SQL Error: Deadlock condition.");
                        return true;

                    case -2:
                        System.Diagnostics.Debug.WriteLine("SQL Error: Timeout expired.");
                        return true;

                    case -1:
                        System.Diagnostics.Debug.WriteLine("SQL Error: Timeout expired.");
                        return true;
                }
            }
```

ضمنا یک سری مباحث به نام اتصال بهبودپذیر و مقاوم به EF 6 در این زمینه اضافه شده:

Connection Resiliency Spec

+

نحوه راه اندازی مجدد یک دیتابیس اس کیوال سرور پس از پر شدن هارد دیسک

عنوان: **اعتبارسنجی در Entity framework Code first قسمت اول**
نویسنده: ایمان محمدی
تاریخ: ۱۳۹۱/۰۵/۱۳ ۱:۳
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Validation

در سری مباحث آموزشی EntityFramework وحیدی نصیری عزیز بصورت مختصر با اعتبار سنجی داده‌ها آشنا شدیم، در این آموزش سه قسمتی سعی می‌کنیم شناخت بیشتری از اعتبار سنجی داده در EF بدست بیاریم.
در EF CodeFirst بصورت پیش فرض پس از فراخوانی متد SaveChanges() اعتبار سنجی داده‌ها انجام می‌پذیرد؛ در صورتیکه که اعتبار سنجی با موفقیت انجام نشود با استثنای DbEntityValidationException روبرو می‌شویم که در اینجا از خاصیت EntityValidationErrors جهت اعلام خطاهای اعتبارسنجی استفاده می‌شود. EntityValidationErrors مجموعه‌ای از خطاهای مربوط به هر موجودیت می‌باشد و هر EntityValidationErrors شامل یک خاصیت ValidationErrors که خود مجموعه‌ای از خطاهای مربوط به propertyها می‌باشد. در تصویر زیر به خوبی می‌تونید این قضیه رو مشاهده کنید.

برای درک بهتر موضوع به ساختار کلاس‌های اعتبارسنجی در تصویر بعدی دقت کنید.

```
namespace System.Data.Entity.Validation
{
    public class DbEntityValidationException : DataException
    {
        // ...
        public IEnumerable<DbEntityValidationResult> EntityValidationErrors { get; }
    }

    public class DbEntityValidationResult
    {
        // ...
        public DbEntityEntry Entry { get; }
        public bool IsValid { get; }
        public ICollection<DbValidationError> ValidationErrors { get; }
    }

    public class DbValidationError
    {
        // ...

        public string ErrorMessage { get; }
        public string PropertyName { get; }
    }
}
```

اعتبار سنجی Context (مجموعه ای ازموجودیت ها)

اعتبار سنجی موجودیت (مجموعه ای از پروپرتی ها)

اعتبار سنجی یک پروپرتی

اعتبار سنجی در چه قسمت هایی اتفاق می‌افتد:

```
1.Property

2.Entity

3.Context
```

الف - Property :

در بخش سوم آموزش EF Code First با متادیتای مورد استفاده در EF و طرز استفاده از آنها آشنا شدیم.

```
[Required(ErrorMessage = "لطفا نام نویسنده را مشخص نمائید")]
        public string AuthorName { set; get; }

        [StringLength(100,MinimumLength=3,ErrorMessage="حداقل سه حرف و حداکثر 100 حرف وارد نمایید.")]
        public string AuthorName { set; get; }
```

همانطور که در ادامه می‌بینید برای اعتبار سنجی فقط به متادیتاهای واقع در DataAnnotations.dll نیاز داریم. بقیه متادیتاها مربوط به نگاشت روابط موجودیتها و نحوه ذخیره اون در دیتابیس می‌باشد.

```
Validation Attributes
AssemblySystem.ComponentModel.DataAnnotations.dll
NamespaceSystem.ComponentModel.DataAnnotations
StringLength
RegularExpression
DataType
Required
Range
CustomValidation
```

```
Mapping Attributes
AssemblyEntityFramework.dll
NamespaceSystem.ComponentModel.DataAnnotations
Key
Column,Table
ComplexType
ConCurrency
TimeStamp
DatabaseGenerated
ForeignKey
InverseProperty
MaxLength
MinLength
NotMapped
```

ب- Entity :

برای اعتبار سنجی یک موجودیت باید اینترفیس IValidatableObject پیاده سازی شود:

```
public class Blog : IValidatableObject
    {

        public int blogID { set; get; }

        [Required(ErrorMessage = "لطفا عنوان وبلاگ را مشخص نمائید")]
        public string Title { set; get; }
        [Required(ErrorMessage = "لطفا نام نویسنده را مشخص نمائید")]
        [StringLength(100,MinimumLength=3,ErrorMessage="حداقل سه حرف و حداکثر 100 حرف وارد نمایید.")]
        public string AuthorName { set; get; }


        public IEnumerable<ValidationResult> Validate(ValidationContext ValidationContext)
        {
             if (this.AuthorName =="بدون نام")
                yield return new ValidationResult
                    ("این نام برای نام نویسنده مجاز نمی‌باشد.", new[] { "AuthorName"});

            if (this.AuthorName == this.Title)
                yield return new ValidationResult
                    ("نام نویسنده وبلاگ و عنوان وبلاگ نمی‌تواند همسان باشد.!", new[] { "AuthorName",
"Title" });
        }
    }
```

نکته: در بررسی هم نام بودن نام نویسنده و نام وبلاگ هردو خاصیت ("AuthorName", "Title") رو درج کردیم اینکار باعث ایجاد دو خطای اعتبارسنجی می‌شود.

پ- Context :

برای اعتبار سنجی در سطح Context باید متد ()ValidateEntity واقع در کلاس DbContext را تحریف کنیم. این قسمت در بخش دوم مقاله کامل شرح داده خواهد شد.

```
protected override DbEntityValidationResult ValidateEntity(DbEntityEntry entityEntry,
System.Collections.Generic.IDictionary<object, object> items)
        {
            return base.ValidateEntity(entityEntry, items);
        }
```

نحوه فراخوانی اعتبار سنجی ها:

```
// اعتبار سنجی یک خاصیت
ICollection<DbValidationError> ValidationProperty = Context.Entry(Blog).Property(p =>
p.AuthorName).GetValidationErrors();

// اعتبار سنجی یک موجودیت
DbEntityValidationResult ValidationEntity = Context.Entry(Blog).GetValidationResult();

// اعتبار سنجی همه موجودیت‌ها
 IEnumerable<DbEntityValidationResult> ValidationContext = Context.GetValidationErrors();
```

نکته : در اعتبار سنجی Context بصورت پیش فرض فقط موجودیت‌های جدید و یا تغییر یافته اعتبار سنجی می‌شوند.

```
EntityState.Added ||  EntityState.Modified
```

ترتیب فراخوانی اعتبار سنجی‌ها :

ابتدا اعتبار سنجی روی Property انجام می‌گیرد در صورتی که خطایی وجود نداشته باشد اعتبار سنجی مرحله بعد یعنی موجودیت‌ها بررسی می‌شود. اگر در مرحله اعتبار سنجی خاصیت‌ها خطایی وجود داشته باشد اعتبار سنجی موجودیت انجام نمی‌گیرد. ترتیب اعتبار سنجی در مرحله Context بستگی به نحوه پیاده سازی ما دارد که در بخش دوم آموزش شرح داده خواهد شد.

سوال:

اعتبار سنجی چند زبانی رو چگونه تعریف کنیم؟

متد GetValidationErrors() رو در الگوی Repository , UOW چگونه پیاده سازی کنیم؟

آیا اعتبار سنجی در کنار موجودیت‌ها از نظر معماری چند لایه کار درستی می‌باشد؟

با پاسخ دادن به سوالات بالا در قالب نظر و یا مقاله به تکمیل موضوع کمک کنید و فضای آموزشی سایت رو رونق ببخشید.

نظرات خوانندگان

نویسنده: daneshjoo
تاریخ: ۱۳۹۲/۰۲/۰۹ ۲۱:۵۱

سلام؛ من دارم با wpf کار می‌کنم و همین طور که می‌دونید در این تکنولوژی اعتبار سنجی خوبی داره حالا سوال من اینه که چطور می‌تونم اعتبار سنجی EF 5 رو با اعتبار سنجی WPF تلفیق کنم و چیز جامع و یکپارچه ای ازش در بیاد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۹ ۲۲:۱۰

مراجعه کنید به قسمت پنجم سری MVVM که در مورد نحوه استفاده از data annotations برای اعتبار سنجی در WPF مطلب داره (محل دریافت دوم کل سری).

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۸:۴۷

من از ValidationHelper که شما قرار دادید در کلاس زیر استفاده کردم و baseentity از کلاس زیر مشتق شده تا تمام موجودیت‌ها اینترفیس IDataErrorInfo رو برای wpf پیاده کرده باشند.

```
public abstract class DataErrorInfo :ObservableObject, IDataErrorInfo
    {
        [Browsable(false)]
        public string Error
        {
            get
            {
                var errors = ValidationHelper.GetErrors(this);
                return string.Join(Environment.NewLine, errors);
            }
        }

        public string this[string columnName]
        {
            get
            {
                var errors = ValidationHelper.ValidateProperty(this, columnName);
                return string.Join(Environment.NewLine, errors);
            }
        }
    }
```

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۲/۲۲ ۰:۵۲

ValidationHelper مورد نیاز جهت کلاس DataErrorInfo :

```
public class ValidationHelper
    {
        public static IList<ValidationResult> GetErrors(object instance)
        {
            var res = new List<ValidationResult>();
            Validator.TryValidateObject(instance,
                new ValidationContext(instance, null, null), res, true);
            return res;
        }


        public static IList<ValidationResult> ValidateProperty(object value, string propertyName)
        {
```

```
            if (string.IsNullOrEmpty(propertyName))
                throw new ArgumentException("Invalid property name", propertyName);

            var propertyValue = value.GetType().GetProperty(propertyName).GetValue(value, null);

            var results = new List<ValidationResult>();
            var context = new ValidationContext(value, null, null) { MemberName = propertyName };
            Validator.TryValidateProperty(propertyValue, context, results);
            return results;
        }
    }
```

نویسنده: ایمان
تاریخ: ۱۳۹۲/۱۱/۱۴ ۱۲:۲۹

سلام خسته نباشید

قسمت دوم این مقاله رو ننوشتین تو سایت؟

خیلی خوبه و من واقعا بهش احتیاج دارم چون دارم الگوی Repositor,Uow رو پیاده سازی میکنم و تو اعتبار سنجی هاش به مشکل خوردم و نمیدونم تو یه بیزینس بزرگ چجوری باید اون رو پیاده سازی کرد که به مشکل نخوره

نویسنده: ابوالفضل موسوی
تاریخ: ۱۳۹۳/۰۴/۱۹ ۱۶:۱۸

سلام . یه سوال داشتم . توی codefirst می‌خوام فیلدهای جدول تولید شده نالیبل باشند و از اعتبار سنجی سمت کلاینت و سرور(MVC) هم استفاده کنم [Requierd] . اگه امکانش هست راهنمای کنید ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۱۹ ۱۶:۳۵

از ViewModel باید استفاده و اعتبارسنجی رو به ViewModel اعمال کنید.

نویسنده: ابوالفضل موسوی
تاریخ: ۱۳۹۳/۰۴/۲۰ ۱۲:۴۹

این کاری که شما گفتین دوباره کاریه! و فیلدهای که من دارم و همچنین جداول خیلی زیادن. من تونستم NULL بودن و اعتبار سنجی سمت سرور رو انجام بدم؛ با کدهایی که زیر قرار میدم . اما چطوری با جی کوری ایجکس باید این ولیدیشنو سمن کلاینت نیز فرا خوامی بکنم ؟

```
[AttributeUsage(AttributeTargets.Field | AttributeTargets.Property, AllowMultiple = false, Inherited =
true)]
    public class RequierdExAttribute : ValidationAttribute
    {
        public RequierdExAttribute(string ErrorMessage)
            : base()
        {
            this.ErrorMessage = ErrorMessage;
        }

        public override bool IsValid(object value)
        {
            if (value == null) return false;
```

```
        return true;
    }
```

حالا بجایِ Requierd روی فیلدها از RequierdEx استفاده میکنم که فیلد مورد نظر در دیتا بیس نال پذیر ساخته میشه . اما مشکل اعتبار سنجی سمت کلاینته؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۲۰ ۱۳:۰

- کاری که می‌خوای، منطقا زیر سؤاله. هم قرار نال پذیر باشه. هم کاربر باید اجبارا پرش کنه! یعنی چی اینکار؟!

- در مورد ویژگی‌های اعتبار سنجی سفارشی و مدیریت کدهای سمت کلاینت اون‌ها مطلب در سایت موجوده:

طراحی ValidationAttribute دلخواه و هماهنگ سازی آن با ASP.NET MVC

MVC 13 قسمت اعتبارسنجی سفارشی

نویسنده: ابوالفضل موسوی
تاریخ: ۱۳۹۳/۰۴/۲۰ ۱۵:۴۰

تا حالا شده که شما بخوای یه فیلد از جدولو در یه مقطعی پر کنی بعد فیلدهای بعدی رو در جاهای دیگه پر کنید ؟ خوب وقتی مقدار فیلدها نال پذیر نباشه نمیشه در مرحله ای اول اون فیلدو ذخیره کرد ! حالا چون ولیدیشن‌های MVC رو هم نیاز دارم پس باید requird هم باشه فیلد ها... تا در مراحل بعدی بگم کدام فیلدها ورودشون اجباریه ! منطقش درسته ! این کاری که می‌گم برای ذخیره سازی چند جدول به صورت همزمانه ! که چند جدول با هم ارتباط هم دارن ! توی database first کار میکنه اما توی code first به دلیل این مشکل کار نمی‌کرد ! من فقط می‌خوام وقتی داده‌ی در جدول اولی ذخیره شد ایدی اون در جدول دومی ذخیره بشه به عنوان کلید خارجی بعد اطلاعات دیگه بعدا پر بشه ! همین ! مرسی از لینک‌های که قرارا دادین برسی کردم لینک دوم کاری که من انجام دادمو اورده سمت کلاینت . و تقریبا مشکلم حل شد . تشکر

عنوان: EF Code First #14
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۹ ۱۷:۱۷:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

ردیابی تغییرات در EF Code first

EF از DbContext برای ذخیره اطلاعات مرتبط با تغییرات موجودیت‌های تحت کنترل خود کمک می‌گیرد. این نوع اطلاعات توسط Change Tracker API جهت بررسی وضعیت فعلی یک شیء، مقادیر اصلی و مقادیر تغییر کرده آن در دسترس هستند. همچنین در اینجا امکان بارگذاری مجدد اطلاعات موجودیت‌ها از بانک اطلاعاتی جهت اطمینان از به روز بودن آن‌ها تدارک دیده شده است.
ساده‌ترین روش دستیابی به این اطلاعات، استفاده از متد context.Entry می‌باشد که یک وهله از موجودیتی خاص را دریافت کرده و سپس به کمک خاصیت State خروجی آن، وضعیت‌هایی مانند Unchanged یا Modified را می‌توان به دست آورد. علاوه بر آن خروجی متد context.Entry، دارای خواصی مانند CurrentValues و OriginalValues نیز می‌باشد. OriginalValues شامل مقادیر خواص موجودیت درست در لحظه اولین بارگذاری در DbContext برنامه است. CurrentValues مقادیر جاری و تغییر یافته موجودیت را باز می‌گرداند. به علاوه این خروجی امکان فراخوانی متد GetDatabaseValues را جهت بدست آوردن مقادیر جدید ذخیره شده در بانک اطلاعاتی نیز ارائه می‌دهد. ممکن است در این بین، خارج از Context جاری، اطلاعات بانک اطلاعاتی توسط کاربر دیگری تغییر کرده باشد. به کمک GetDatabaseValues می‌توان به این نوع اطلاعات نیز دست یافت.
حداقل چهار کاربرد عملی جالب را از اطلاعات موجود در Change Tracker API می‌توان مثال زد که در ادامه به بررسی آن‌ها خواهیم پرداخت.

کلاس‌های مدل مثال جاری

در اینجا یک رابطه many-to-one بین جدول هزینه‌های اقلام خریداری شده یک شخص و جدول فروشندگان تعریف شده است:

```
using System;

namespace EF_Sample09.DomainClasses
{
    public abstract class BaseEntity
    {
        public int Id { get; set; }

        public DateTime CreatedOn { set; get; }
        public string CreatedBy { set; get; }

        public DateTime ModifiedOn { set; get; }
        public string ModifiedBy { set; get; }
    }
}
```

```
using System;

namespace EF_Sample09.DomainClasses
{
    public class Bill : BaseEntity
    {
        public decimal Amount { set; get; }
        public string Description { get; set; }

        public virtual Payee Payee { get; set; }
    }
}
```

```
using System.Collections.Generic;

namespace EF_Sample09.DomainClasses
{
    public class Payee : BaseEntity
    {
        public string Name { get; set; }

        public virtual ICollection<Bill> Bills { set; get; }
    }
}
```

به علاوه همانطور که ملاحظه می‌کنید، این کلاس‌ها از یک abstract class به نام BaseEntity مشتق شده‌اند. هدف از این کلاس پایه تنها تامین یک سری خواص تکراری در کلاس‌های برنامه است و هدف از آن، مباحث ارث بری مانند TPH، TPT و TPC نیست. به همین جهت برای اینکه این کلاس پایه تبدیل به یک جدول مجزا و یا سبب یکی شدن تمام کلاس‌ها در یک جدول نشود، تنها کافی است آن‌را به عنوان DbSet معرفی نکنیم و یا می‌توان از متد Ignore نیز استفاده کرد:

```
using System.Data.Entity;
using EF_Sample09.DomainClasses;

namespace EF_Sample09.DataLayer.Context
{
    public class Sample09Context : MyDbContextBase
    {
        public DbSet<Bill> Bills { set; get; }
        public DbSet<Payee> Payees { set; get; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Ignore<BaseEntity>();

            base.OnModelCreating(modelBuilder);
        }
    }
}
```

الف) به روز رسانی اطلاعات Context در صورتیکه از متد context.Database.ExecuteSqlCommand مستقیما استفاده شود

در قسمت قبل با متد context.Database.ExecuteSqlCommand برای اجرای مستقیم عبارات SQL بر روی بانک اطلاعاتی آشنا شدیم. اگر این متد در نیمه کار یک Context فراخوانی شود، به معنای کنار گذاشتن Change Tracker API می‌باشد؛ زیرا اکنون در سمت بانک اطلاعاتی اتفاقاتی رخ داده‌اند که هنوز در Context جاری کلاینت منعکس نشده‌اند:

```
using System;
using System.Data.Entity;
using EF_Sample09.DataLayer.Context;
using EF_Sample09.DomainClasses;

namespace EF_Sample09
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample09Context,
Configuration>());

            using (var db = new Sample09Context())
            {
                var payee = new Payee { Name = "فروشگاه سر کوچه" };
```

```
                var bill = new Bill { Amount = 4900, Description = "یک سطل ماست", Payee = payee };
                db.Bills.Add(bill);

                db.SaveChanges();
            }

            using (var db = new Sample09Context())
            {
                var bill1 = db.Bills.Find(1);
                bill1.Description = "ماست";

                db.Database.ExecuteSqlCommand("Update Bills set Description=N'سطل ماست' where
id=1");
                Console.WriteLine(bill1.Description);

                db.Entry(bill1).Reload(); //Refreshing an Entity from the Database
                Console.WriteLine(bill1.Description);

                db.SaveChanges();
            }
        }
    }
}
```

در این مثال ابتدا دو رکورد به بانک اطلاعاتی اضافه می‌شوند. سپس توسط متد db.Bills.Find، اولین رکورد جدول Bills بازگشت داده می‌شود. در ادامه، خاصیت توضیحات آن به روز شده و سپس با استفاده از متد db.Database.ExecuteSqlCommand نیز بار دیگر خاصیت توضیحات اولین رکورد به روز خواهد شد.

اکنون اگر مقدار bill1.Description را بررسی کنیم، هنوز دارای مقدار پیش از فراخوانی db.Database.ExecuteSqlCommand می‌باشد، زیرا تغییرات سمت بانک اطلاعاتی هنوز به Context مورد استفاده منعکس نشده است.

در اینجا برای هماهنگی کلاینت با بانک اطلاعاتی، کافی است متد Reload را بر روی موجودیت مورد نظر فراخوانی کنیم.

ب) یکسان سازی ی و ک اطلاعات رشته‌ای دریافتی پیش از ذخیره سازی در بانک اطلاعاتی

یکی از الزامات برنامه‌های فارسی، یکسان سازی ی و ک دریافتی از کاربر است. برای این منظور باید پیش از فراخوانی متد SaveChanges نهایی، مقادیر رشته‌ای کلیه موجودیت‌ها را یافته و به روز رسانی کرد:

```
using System;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Reflection;
using EF_Sample09.DataLayer.Toolkit;
using EF_Sample09.DomainClasses;

namespace EF_Sample09.DataLayer.Context
{
    public class MyDbContextBase : DbContext
    {
        public void RejectChanges()
        {
            foreach (var entry in this.ChangeTracker.Entries())
            {
                switch (entry.State)
                {
                    case EntityState.Modified:
                        entry.State = EntityState.Unchanged;
                        break;

                    case EntityState.Added:
                        entry.State = EntityState.Detached;
                        break;
                }
            }
        }

        public override int SaveChanges()
        {
```

```
            applyCorrectYeKe();
            auditFields();
            return base.SaveChanges();
        }

        private void applyCorrectYeKe()
        {
            //پیدا کردن موجودیت‌های تغییر کرده
            var changedEntities = this.ChangeTracker
                                      .Entries()
                                      .Where(x => x.State == EntityState.Added || x.State ==
EntityState.Modified);

            foreach (var item in changedEntities)
            {
                if (item.Entity == null) continue;

                //یافتن خواص قابل تنظیم و رشته‌ای این موجودیت‌ها
                var propertyInfos = item.Entity.GetType().GetProperties(
                    BindingFlags.Public | BindingFlags.Instance
                    ).Where(p => p.CanRead && p.CanWrite && p.PropertyType == typeof(string));

                var pr = new PropertyReflector();

                //اعمال یکپارچگی نهایی
                foreach (var propertyInfo in propertyInfos)
                {
                    var propName = propertyInfo.Name;
                    var val = pr.GetValue(item.Entity, propName);
                    if (val != null)
                    {
                        var newVal = val.ToString().Replace("ی" ,"ی").Replace("ک" ,"ک");
                        if (newVal == val.ToString()) continue;
                        pr.SetValue(item.Entity, propName, newVal);
                    }
                }
            }
        }

        private void auditFields()
        {
            // var auditUser = User.Identity.Name; // in web apps
            var auditDate = DateTime.Now;
            foreach (var entry in this.ChangeTracker.Entries<BaseEntity>())
            {
                // Note: You must add a reference to assembly : System.Data.Entity
                switch (entry.State)
                {
                    case EntityState.Added:
                        entry.Entity.CreatedOn = auditDate;
                        entry.Entity.ModifiedOn = auditDate;
                        entry.Entity.CreatedBy = "auditUser";
                        entry.Entity.ModifiedBy = "auditUser";
                        break;

                    case EntityState.Modified:
                        entry.Entity.ModifiedOn = auditDate;
                        entry.Entity.ModifiedBy = "auditUser";
                        break;
                }
            }
        }
    }
}
```

اگر به کلاس Context مثال جاری که در ابتدای بحث معرفی شد دقت کرده باشید به این نحو تعریف شده است (بجای DbContext از MyDbContextBase مشتق شده):

```
public class Sample09Context : MyDbContextBase
```

علت هم این است که یک سری کد تکراری را که می‌توان در تمام Contextها قرار داد، بهتر است در یک کلاس پایه تعریف کرده و سپس از آن ارث بری کرد.

تعاریف کامل کلاس MyDbContextBase را در کدهای فوق ملاحظه می‌کنید.

در اینجا کار با تحریف متد SaveChanges شروع می‌شود. سپس در متد applyCorrectYeKe کلیه موجودیت‌های تحت نظر ChangeTracker که تغییر کرده باشند یا به آن اضافه شده باشند، یافت شده و سپس خواص رشته‌ای آن‌ها جهت یکسانی سازی ی و ک، بررسی می‌شوند.

ج) ساده‌تر سازی به روز رسانی فیلدهای بازبینی یک رکورد مانند DateCreated، DateLastUpdated و امثال آن بر اساس وضعیت جاری یک موجودیت

در کلاس MyDbContextBase فوق، کار متد auditFields، مقدار دهی خودکار خواص تکراری تاریخ ایجاد، تاریخ به روز رسانی، شخص ایجاد کننده و شخص تغییر دهنده یک رکورد است. به کمک ChangeTracker می‌توان به موجودیت‌هایی از نوع کلاس پایه BaseEntity دست یافت. در اینجا اگر entry.State آن‌ها مساوی EntityState.Added بود، هر چهار خاصیت یاد شده به روز می‌شوند. اگر حالت موجودیت جاری، EntityState.Modified بود، تنها خواص مرتبط با تغییرات رکورد به روز خواهند شد.
به این ترتیب دیگر نیازی نیست تا در حین ثبت یا ویرایش اطلاعات برنامه نگران این چهار خاصیت باشیم؛ زیرا به صورت خودکار مقدار دهی خواهند شد.

د) پیاده سازی قابلیت لغو تغییرات در برنامه

علاوه بر این‌ها در کلاس MyDbContextBase، متد RejectChanges نیز تعریف شده است تا بتوان در صورت نیاز، حالت موجودیت‌های تغییر کرده یا اضافه شده را به حالت پیش از عملیات، بازگرداند.

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۴/۱۲ ۱۰:۲۵

بعضی مواقع ممکنه نیاز باشه بررسی کنیم آیا موجودیت‌های ما تغییر داشته اند یا خیر (برای مثال در صورتی که تغییرات ذخیره نشده در سیستم وجود دارد ؛ پیغامی مبنی بر ذخیره یا عدم ذخیره کردن تغییرات نمایش دهیم) .برای اینکار من از متد زیر در Context استفاده میکنم:

```
public bool HasChanges()
        {
            foreach (var entry in this.ChangeTracker.Entries())
            {
                if (entry.State == EntityState.Modified || entry.State = EntityState.Added)
                {
                    return true;
                }
            }
            return false;
        }
```

در کد بالا State موجودیت‌ها بررسی می‌شود.اگر با یکی از دو مقدار Modified و یا Added برابر باشد مقدار true و در غیر این صورت false برمیگرداند.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۶/۱۲ ۹:۵۹

با سلام .
1) کلاس PropertyReflector درکدام Namespace قرار دارد ؟
2) چرا برای متد SaveChanges حالت EntryState.Deleted درنظر گرفته نشده است؟
با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۲ ۱۰:۳۰

- یک کلاس سفارشی است که در سورس‌های این سری قرار دارد (^).
- چون زمانیکه رکوردی به صورت فیزیکی حذف شد، نیازی به اصلاح ی و ک آن نیست (وجود خارجی ندارد که اهمیت داشته باشد).

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۸:۵

سلام آقای نصیری،
تو عمل ما از Unit of Work استفاده میکنیم،حالا در اینصورت نحوه‌ی استفاده از این روش چجوره؟
1- باید اینترفیس IUnitOfWork رو توی کلاس MyDbContextBase پیاده سازی کنیم و پیاده سازی متد SaveChanges اینترفیس IUnitOfWork توی کلاس MyDbContextBase باید به شکل زیر باشه؟

```
applyCorrectYeKe();
auditFields();
return base.SaveChanges();
```

2- یا باید اینترفیس IUnitOfWork رو توی کلاس به ارث رسیده از MyDbContextBase یعنی Sample09Context پیاده سازی کرد و MyDbContextBase باید بی خبر از وجود اینترفیس IUnitOfWork و پیاده سازی هاش باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۸:۱۶

ارث بری رو به صورت «is a» معرفی می‌کنند و می‌خونند. یعنی هر دو حالتی که نام بردید یکی است و فرقی نمی‌کنه.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۸:۲۴

گیج شدم آقای نصیری،میشه یکم موضوع رو بازش کنید؟
پس اینجوری بگیم،به نظر شما کدوم استانداردتره(بهتره)؟1 یا 2؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۸:۴۰

هدف الگوی واحد کار این است که یک سری اعمال مرتبط با موجودیت‌های مختلف در یک تراکنش انجام شوند. در اینجا هم این مورد (مثلا یکی کردن ی و ک) در طی تراکنش جاری انجام خواهد شد. ضمن اینکه زمانیکه با IUnitOfWork کار می‌کنید هیچ وقت به صورت مستقیم با کلاس‌های مشتق شده از DbContext کار نخواهید کرد. وهله سازی و dispose این‌ها کار مثلا StructureMap و امثال آن خواهد بود.
MyDbContextBase فقط برای مدیریت کدهای تکراری بین برنامه‌های مختلف ایجاد شده است. اصلا می‌تواند وجود خارجی هم نداشته باشد. اگر کل سیستم شما یک پروژه است و از این کدها نمی‌خواهید در پروژه دیگری استفاده کنید، یک کلاس مشتق شده از DbContext کفایت می‌کند. بنابراین اگر قصد استفاده مجدد از کدهای زیرساخت پروژه جاری خودتون رو در پروژه‌های دیگر دارید، می‌تونید به غنی سازی MyDbContextBase بیشتر بپردازید.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۸:۵۰

منم گیرما D:
پس روش 2 استانداردتره؟درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۱

روش اول بهتره. در کلاس Context نهایی و اصلی فقط باید تعاریف DbSet و حداکثر تحریف متد OnModelCreating وجود داشته باشد. مابقی کد تکراری است (و پایه انجام پروژه‌های دیگر) که می‌شود به MyDbContextBase انتقالشون داد. کل هدف این کلاس پایه، مدیریت کدهای تکراری بین پروژه‌های مختلف است.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۱۹:۶

یه دنیا ممنون.

نویسنده: daneshjoo
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱:۳۸

مهندس عزیز , من در برنامم در یه فرم برا خروج باید entity رو چک کنم که اگه کاربر درخواست درج یه رکورد رو زده باشه و بعدش حداقل یک پارامتر رو پر کرده و دکمه ذخیره رو نزده براش یه پیغام بیاد که شما مایل به درج رکورد هستید یانه.

جطوری می‌تونم این حالت رو چک کنم؟

برا ویرایش مشکلی نیست , طبق گفته خودتون state رو چک می‌کنم اگه modifed بود پیغام مایل به ذخیره تغرات رو می‌دم اما برا درج نمی‌دونم چه شرطی رو بنویسم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۸:۴۴

حالت Added هم دارد:

```
//پیدا کردن موجودیت‌های تغییر کرده
var changedEntities = this.ChangeTracker
                        .Entries()
                        .Where(x => x.State == EntityState.Added || x.State ==
EntityState.Modified);
```

نویسنده: daneshjoo
تاریخ: ۱۳۹۲/۰۲/۱۱ ۰:۲۳

ممنون

این کد شما زمانی هست که من رکورد رو ذخیره کرده باشم و لی من حالتی رو می‌خوام که یک شی از جدول رو در برنامه ایجاد کردم و بعدش می‌خوام تست کنم که کاربر در ستون هاش مقداری وارد کرده یا نه که اگه حتی یک مقدار وارد کرده بود پیغام << آیا می‌خواهید شخص مورد نظر اضافه شود >> را بدم که اگه تایید کرد من اونو ضافه کنم .

من وقتی با کد : contex.entry(table1).state وضعیت شی table1 رو که ایجاد کردم رو قبل از هر کاری چک می‌کنم این کد مقدار deteced رو می‌ده و وقتی که ستونهای شی table1 رومقدار دهی می‌کنم مقدار deteced رو باز میده و وقتی این شی رو با استفاده از متد savecheng در دیتابیس ذخیره می‌کنم بعد state رو چک می‌کنم مقدار unchenged رو بهم می‌ده

لطفا در این خصوص کمک کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۱ ۰:۳۹

- آزمایش کردی یکبار؟ (من این رو در یک برنامه WPF استفاده کردم؛ با یک Context در سطح ViewModel که کار تحت نظر قرار دادن اطلاعات رو داره. حتی دکمه undo هم میشه طراحی کرد با استفاده از متد RejectChanges و در WPF با سیستم Binding خوبی که داره بلافاصله UI به صورت خودکار به روز میشه)
- Added مربوط به زمانی است که اطلاعات به سیستم ردیابی (context در اینجا) اضافه شده و نه به بانک اطلاعاتی. Modified مربوط به حالتی است که اطلاعات تحت نظر سیستم ردیابی مثلا یک خاصیت آن تغییر کرده است؛ پیش از ذخیره سازی در بانک اطلاعاتی. EF بر همین اساس هست که تشخیص می‌ده چه کوئری را باید صادر کند برای ذخیره یا به روز رسانی نهایی اطلاعات.

نویسنده: daneshjoo
تاریخ: ۱۳۹۲/۰۲/۱۷ ۲۲:۴

آقا وحید عزیز حرف شما درست بود و من تقریبا اشتباه فهمیده بودم .

من در برنامم اول میام یک شی از تیبلم رو می‌سازم :

```
var t=new db.table1();
```

که اگه بیای state اونو بگیری بهت detached نشون میده
و اگه بیای اونو به مدل اضافه کنی :

```
db.table1.add(t);
```

که اگه بیای state اونو بگیری بهت added نشون میده
حالا سوال من اینه که اگه من بخوام قبل از اینکه شی رو add کنم بخوام فهمم که مقداری به ستونها اضافه شده باید چکار کنم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۷ ۲۲:۳۵

- بدون Add شدن یک شیء که Context از وجود آن اطلاعاتی نخواهد داشت. صرفا یک شیء معمولی تشکیل شده در حافظه است.
+ لطفا بحث و پاسخ‌های داده شده را یکبار بررسی کنید. امکان کوئری گرفتن از DbContext برای درک اینکه چه چیزی به آن پیش از درج در بانک اطلاعاتی کم یا زیاد شده، موجود است؛ که با مثال در مطلب جاری عنوان شده

```
var changedEntries = con.ChangeTracker.Entries().Where(x => x.State == EntityState.Added || x.State ==
EntityState.Modified).ToList();
if (changedEntries.Any())
{
   // یعنی یک سری مدخل ثبت نشده داریم که الان لیستش رو هم داریم
}
```

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۸:۴۷

بهترین راه برای مدیریت تغییرات در یک سناریوی Disconnected چیه؟
منظورم اینه که اگر از کلاسهای POCO استفاده کنیم و بخواهیم تغییرات سمت کلاینت (WinForm)رو روی Entityهای DBContext اعمال کنیم مناسب‌ترین راه چیه؟ (هم کشف تغییرات سمت کلاینت و هم اعمال اونها سمت سرور)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۹:۷

برای اتصال به Context یک سری روش مانند Attach و غیره هست که در بحث « استفاده از کلیدهای خارجی در EF » مطرح (قسمت وارد کردن یک شیء به سیستم Tracking) و مثال زده شده.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۹:۴۶

ممنون؛ ولی منظورم اینه که چجوری میشه این تغییرات سمت کلاینت رو بصورت خودکار تشخیص داد و سمت سرور اعمال کرد؟ یعنی ممکنه یک سری از POCOها در سمت کلاینت ایجاد شده باشند یک سری دیگه ویرایش و یک سری دیگه حذف. در اینصورت چجوری میشه این تغییرات را تشخیص داد و مدیریت کرد؟ و ما از قبل نمیدونم سمت کلاینت دقیقا کدوم یک از عملیات CRUD اتفاق افتاده.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۴ ۲۲:۸

در سیستم‌های Disconnected، یعنی زمانی که ارتباط دائم بین Context و Entityها وجود ندارد (مثل سیستم‌های مبتنی بر WCF و SOA) باید از Entity Self Tracking استفاده کنید که برای اولین بار در .Net4 و VS2010 معرفی شد و این امکان رو به شما می‌ده تمام تغییرات موجود در Entity + وضعیت Entity مثل Added و Deleted و Modified را به سمت سرور ارسال کنید.
هر تغییری رو که در خواص یک کلاس اعمال کنید مقدار جدید و مقدار قدیم به علاوه نام Property در خود مدل ، Track می‌شوند و تمام این اطلاعات همراه Entity به سرور ارسال شده و در سمت سرور هم یک Extension Method به نام ApplyChanged برای

ObjectContext وجود داره که با توجه به تغییرات و State هر Entity داده‌ها رو ذخیره می‌کنه.
در ضمن شما از طریق دو متد StopTracking و StartTracking می‌تونید تمام تغییرات Entity رو استارت یا متوقف کنید.
فقط نکته مهم اینه که استفاده از این روش کمی هزینه بر است (چون هر Entity تمام تغییرات خود را در 2 Dictionary به نام‌های OriginalValueCollection و CurrentValueCollection ذخیره میکنه در نتیجه هنگام انتقال داده‌ها باید حواستون به حجم داده‌های ارسالی هم باشه.)
در ضمن در این حالت دیگه Lazy Loading ساپورت نمیشه و فقط می‌تونید از Include استفاده کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۴ ۲۳:۰

مدیریت Context (یا همان سیستم ردیابی خودکار به بیانی دیگر) در برنامه‌های ویندوزی «معمولی» مانند WPF یا WinForms یا سرویس‌های ویندوز NT و ...، یا در سطح فرم است (با آغاز فرم، Context، وهله سازی می‌شود و با بسته شدن آن خاتمه خواهد یافت) یا در طی یک عملیات کوتاه مانند کلیک بر روی یک دکمه فعال و سپس Dispose می‌شود. یکی از این دو حالت رو بسته به سناریویی که دارید می‌تونید دنبال کنید.
شما شیء Context رو در سطح فرم تعریف می‌کنید (چون می‌تونید؛ چون برنامه‌های ویندوزی متفاوت‌اند از برنامه‌های وب بدون حالت که در آن‌ها پس از نمایش صفحه، کلیه اشیاء تخریب می‌شوند)، حالا هر شیءایی که اضافه بشه، به Context جاری اضافه شده، حذف بشه از این مرجع حذف شده یا اگر ویرایش شود باز هم به صورت خودکار، تحت نظر Context تعریف شده در سطح فرم است. نهایتا با فراخوانی یک SaveChanges تمام این تغییرات بدون نیاز به محاسبه خاصی در کدهای ما، توسط EF اعمال می‌شوند. سیستم Tracking به صورت خودکار، کوئری‌های update، insert و delete رو محاسبه و اجرا می‌کنه. نیازی به مدیریت خاصی بجز تعریف Context در سطح فرم نداره.
به صورت خلاصه مرسوم نیست در مثلا WinForms «متداول»، منقطع از Context کار کرد، چون اساسا می‌شود به سادگی، تا زمانیکه یک فرم در حال نمایش است، Context و سیستم ردیابی خودکار آن‌را زنده نگه داشت و از آن استفاده کرد.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۴ ۲۳:۱۹

درسته جناب نصیری ولی در صورتی که پروژه به صورت SOA باشه مثل WCF (خواه Win App باشد خواه Web App) دیگه Context در سطح فرم معنی پیدا نمی‌کنه . در این حالت اصلا Context سمت کلاینت وجود ندارد که بتونیم ردیابی خودکار اشیا را به عهده اون بزاریم. سمت کلاینت فقط مدل برنامه است به علاوه یک ChannelFactory برای ارتباط با سرور .در این حالت خود مدل‌های برنامه باید توانایی Track کردن رو داشته باشند و Context سمت برنامه با استفاده از این اطلاعات Track شده توسط Entity عملیات CRUD را رو دیتابیس اجرا می‌کنه.
در این حالت می‌تونیم N تا Entity رو که هر کدوم یک State مشخص دارند مثل Addedd , Deleted , modified توسط متد ApplyChanged که برای ObjectContext تعریف شده به ObjectStateManager اضافه کنیم و در نهایت با دستور SaveChanged اطلاعات به صورت نهایی روی دیتابیس اعمال می‌شوند.
توضیحات تکمیلی (^)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۵ ۰:۴

- بحث Self tracking entities در حالت database first است و در Code first پشتیبانی نمی‌شود و احتمالا هم نخواهد شد .
در اینجا رسما پایین صفحه قید شده که دیگر از STE با EF 5.0 برای حالت‌های N-Tier استفاده نکنید.
- در EF Code first توصیه می‌شود که برای کار با WCF از WCF Data Services و یا RIA Services استفاده کنید. هر دوی این‌ها برای کار با EF Code first به روز شدن اخیرا. هر دوی این‌ها change tracking سمت کاربر رو هم پشتیبانی می‌کنند.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۱۰/۰۷ ۱۲:۱۹

جناب نصیری ممنون از جوابتون، با روشی که میفرمایید(استفاده از DbContext در سطح فرم)چنانچه مدل من به شکل زیر باشه :

و صفحه ویرایش سفارش من به شکل زیر:

و کاربر پس از زدن دکمه ویرایش قادر به انجام کارهای زیر باشه:

ویرایش شماره سفارش

ویرایش یک OrderLine

حذف یک OrderLine

ویرایش یک OrderLine

اضافه کردن یک OrderLine

و سپس بخواد دکمه ذخیره رو بزنه؛ برای اینکه کل تغییرات کاربر رو ذخیره کنم کدوم یک از روشهای زیر رو بایستی استفاده کنم؟

eventهای مناسبی رو پیدا کنم و به محض رخ دادن اونها بر اساس اونها تصمیم بگیرم که entity مورد نظر بایستی در DbContext(اضافه/حذف و یا ویرایش) بشه؟

تا زمانی که کاربر دکمه ذخیره رو نزده، کاری با DbContext نداشته باشم و وقتی کاربر دکمه ذخیره رو زد، گراف(سفارش و آیتمهای سفارش) رو به DbContext بدم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۷ ۱۳:۸

در WPF مفهومی وجود دارد به نام انقیاد دو طرفه (two way binding). زمانیکه کاربر UI را به روز می‌کند، خود به خود (بدون نیاز به کدنویسی اضافه‌تری، منهای تنظیمات اولیه آن)، اشیاء یک لیست به روز می‌شوند و برعکس. در این بین EF Code first با استفاده از خاصیت Local آن توانایی اتصال به یک چنین سیستمی را دارد و در اینجا عملا یکپارچگی کاملی رخ داده و نیازی نیست کار اضافه‌تری انجام دهید. Context از تمام تغییرات شما مطلع است. فقط کافی است SaveChanges فراخوانی شود تا کلیه تغییرات انجام شده و تحت نظر آن به صورت یکجا در بانک اطلاعاتی ثبت شوند. این خاصیت Local در WinForms هم قابل استفاده است.
برای مطالعه بیشتر:

Databinding with WPF

Databinding with WinForms

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۱۰/۰۷ ۱۳:۴۲

ممنون ولی در مثال Databinding with WinForms ، در متد OnLoad فرم این دستور استفاده شده:

```
this.categoryBindingSource.DataSource = _context.Categories.Local.ToBindingList();
```

از خواص DbContext در لایه UI استفاده شده، این کار درست به نظر نمیاد، نظر شما چیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۷ ۱۳:۴۶

در WinForms مگر اینکه از الگوی MVP استفاده شود جهت جداسازی لایه‌ها، وگرنه روش متداول آن همان مثالی است که مایکروسافت ارائه داده.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۱۰/۰۷ ۱۷:۳۱

با توجه به اینکه طول عمر یک DbContext در وب معادل طول عمر یک درخواست است(EF Code First #12)، سناریو فوق در وب چطور میتواند پیاده شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۷ ۱۷:۳۹

به پیاده سازی پروژه IRIS مراجعه کنید؛ برای مشاهده یک نمونه واقعی استفاده از آن در وب.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۱۰/۲۵ ۹:۵۳

ممنون از راهنماییتون، پروژه رو نگاه کردم، شبیه به سناریو فوق در بخش ویرایش یک پست وجود داره که در اونجا رابطه یک به چند بین **Post** و **DownLoadLink** هست.
اما روشی که برای ویرایش در اونجا استفاده شده به این صورت هست که ابتدا در در اکشن متد **EditPost** کنترلر **PostController** ناحیه ادمین، همه **DownLoadLink** های پستی که در حال ویرایش آن هستیم، پاک میشوند و سپس در متد **EditPost** مربوط به **PostService**، دانلودلینکهای **EditPostModel** به جای آنها مینشینند، در واقع در صورت ویرایش دانلودلینکهای یک پست، ویرایش واقعی انجام نمیشود، بلکه این کار با حذف(sql Delete) کلیه دانلودلینکهای آن پست از DB و درج مجدد دانلودلینکهای تغییر یافته و نیافته (sql Insert) ، شبیه سازی میشود. درست است که نتیجه کار با ویرایش واقعی(sql Update) تفاوت نمیکند اما به نظر من ویرایش با این روش سه مشکل زیر را دارد:
حتی اگر هنگام ویرایش یک پست هیچ تغییری در دانلودلینکها داده نشود باز هم حذف تمامی آنها و درج مجدد آنها صورت خواهد گرفت.

پایین آمدن کارایی وقتی که تعداد رکوردهای طرف چندِ رابطهٔ یک به چند زیاد باشد(در اینجا دانلودلینکها).
در برخی موارد مثل مورد زیر که طرف چند رابطه (**OrderLine**) دارای ارتباطاتی باشد، حذف فیزیکی و درج مجدد آن به هنگام ویرایش **Order** دردسرساز خواهد شد:

آیا راهی برای رفع این موارد وجود دارد؟ به عبارت دیگر آیا راهی وجود دارد که به جای حذف فیزیکی رکوردها و درج تغییرات(Delete,Insert)؛ فقط تغییرات را اعمال کنیم(Update)؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۵ ۱۱:۱

به چه مشکلی برخوردید زمانیکه رکوردهای OrderLine را مستقیما واکشی و ویرایش کردید؟ Id هر Order در OrderLineهای مرتبط وجود دارد. بنابراین یک کوئری بگیرید بر این اساس. نتیجه‌ی این کوئری الان تحت نظر سیستم Tracking است. در این بین آن‌ها را ویرایش کرده و سپس SaveChanges در آخر کار.
احتمالا علت حذف لینک‌ها در آن پروژه این بوده: من یک سری لینک دارم. اما زمان ارسال به سرور نمی‌دانم کدام یکی جدید است و کدام یکی دارد ویرایش می‌شود. برای حل این مساله باید id هر رکورد را در یک فیلد مخفی کنار لینک قرار داد؛ یا حتی در یک ویژگی data-*. بعد هنگام ارسال به سرور، بر اساس این idها می‌شود تصمیم گیری کرد. اگر رکوردی جدید است و می‌شود به صورت پویا ردیفی را به لیست اضافه کرد، این id وجود ندارد. اگر قدیمی است، id آن دقیقا مشخص است و به سرور ارسال می‌شود. ضمن اینکه با داشتن این id حتی دیگر نیازی به واکشی رکورد متناظر آن از دیتابیس نخواهد بود. می‌شود به کمک روش علامتگذاری یک شیء به صورت EntityState.Modified، آن‌را وارد سیستم Tracking کرد. در این مورد در مطلب « کار با کلیدهای اصلی و خارجی در EF Code first » بیشتر بحث شده و نکته‌ی مهم آن، کار کردن با Id یک شیء است در ارتباطات و تعریف صریح آن توسط ویژگی ForeignKey. همچنین مطلب « رفتار متصل و غیر متصل در EF چیست؟ » نیز مفید است.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۱۰/۲۵ ۱۳:۱۷

پستها را خواندم، درست است؛ با این روش که شما فرمودید، میتوان رکوردهای جدید از قدیم را تشخیص داد(با استفاده از id) ولی موردی که باقی می‌ماند این است که سمت کلاینت ممکن است برخی از OrderLineها ویرایش شوند و برخی نشوند و ما دقیقا نمیدانیم کدامها ویرایش شده اند و کدامها نشده اند، تا در سرور state آنها را بصورت Unchanged و یا Modified قرار دهیم.
آیا روشی برای تشخیص این مورد وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۵ ۱۳:۵۹

با استفاده از فناوری‌های SPA اینکار ممکن است. دقیقا می‌توان در سمت کلاینت تشخیص داد که چه فیلدهایی تغییر کرده‌اند و صرفا آن‌ها را به سرور ارسال کرد. یک مثال AngularJS آن در اینجا و یا اینکار با jQuery هم میسر است: یک مثال

نویسنده: سهیل
تاریخ: ۱۳۹۲/۱۱/۱۰ ۲۰:۵۹

با سلام. اگر بخواهیم همه پروپرتی‌های از جنس رشته رو قبل از ذخیره در دیتابیس Trim کنیم میشه همونجایی که "ی" و "ک" رو عوض میکنیم Trim رو هم انجام بدیم؟ آیا این کار درست هستش؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۰ ۲۱:۵۸

- بله. در همانجا قابل تنظیم است.
- بستگی دارد به پروژه و نوع کاربرد. اگر جهت مقاصد امنیتی یا نمایشی است، این فضاهای خالی نیاز است و معنا دار.

ضمنا اگر از SQL Server استفاده می‌کنید و نوع داده‌ی مورد استفاده nvarchar است و همچنین ansi_padding set به off تنظیم شده، این trim خودکار خواهد بود (البته در حالت پیش فرض، ansi_padding خاموش نیست).

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۱ ۲۲:۱۷

سلام

من از این روش استفاده کردم اما متاسفانه در زمان بروزرسانی و استفاده از متدی مانند AddOrUpdate با خطا مواجه می‌شود بررسی کردم در زمان ثبت داده چون دو فیلد CreatedOn, ModifidOn مقداردهی میشه و مشکلی نیست. اما در زمان بروز رسانی چون فقط ModifidOn مقدار میگیره و فیلد تاریخ ایجاد مقدار پیش فرض میگیره و در زمان بروز رسانی با خطای زیر مواجه میشم. نظر شما چیه؟

```
The conversion of a datetime2 data type to a datetime data type resulted in an out-of-range value.
The statement has been terminated.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۱ ۲۳:۲۰

متد AddOrUpdate مطابق توصیه تیم EF فقط برای متد Seed طراحی شده‌است و از آن در برنامه استفاده نکنید (چون برخلاف تصور، تمام خواص را به روز رسانی می‌کند و اگر در این بین اطلاعاتی مقدار دهی نشود، با نال جایگزین خواهد شد که علت بروز خطای فوق است). هدف اصلی آن هم صرفا عدم ثبت اطلاعات تکراری در حین فراخوانی متد Seed است. به همین جهت آن‌را در فضای نام System.Data.Entity.Migrations قرار داده‌اند. اطلاعات بیشتر

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۲ ۰:۱۱

بله کاملا درسته
منم در زمان Seed فراخوانی میکنم اما همین خطا در هر بار اجرای برنامه رخ میده (غیر از دفعه اول که دیتابیس میسازه)، در بقیه موارد هر بار مدلم تغییر کنه این خطا رخ میده.
در صورتی که فیلد کلید مقداردهی نشه داده تکراری ثبت میشه در هرباز اجرا اگر هم فیلد کلید مقداردهی بشه (به صورت دستی) خطای فوق الذکر رخ میده

نویسنده: صابر فتح الهی

تاریخ: ۱۳۹۳/۱۰/۰۲ ۲۰:۴۸

مثلا من دستور زیر بنویسم خطا دارم

```
context.MemberSettings.AddOrUpdate(new MemberSetting { Id = 1, AllowableFileTypeUpload =
".jpg;.jpeg;.bmp;.png;.gif;.tif", RowCount = 10, MaxFileSize = 512 });
```

اما در صورتی که فیلد Id حذف کنم خطا رخ نمیده اما در عوض داده تکراری ثبت میشه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۲ ۲۱:۴

از متد Any قبل از ثبت در متد Seed استفاده کنید:

```
if (!context.MemberSettings.Any())
{
 // ... add new MemberSettings
}
```

عنوان: ردیابی تغییرات در Entity Framework، بخش اول
نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

همان طور که می‌دانید، Entity Framework تغییراتی را که بر روی اشیا انجام می‌دهید، ردیابی می‌کند. بدیهی است که EF از طریق ردیابی این تغییرات است که می‌تواند تغییرات انجام شده را شناسایی کند و آن‌ها را در مواقع مورد نیاز مانند ذخیره‌ی تغییرات (DbContext.SaveChanges)، بر روی پایگاه داده اعمال کند. شما می‌توانید به اطلاعات این ردیاب تغییر و اعمال مرتبط به آن از طریق ویژگی DbContext.ChangeTracker دسترسی پیدا کنید.
در این مقاله بیشتر سعی به بررسی مفاهیم ردیابی و روش هایی که EF برای ردیابی تغییرات استفاده می‌کند، بسنده می‌کنم و بررسی APIهای مختلف آن را به مقاله ای دیگر موکول می‌کنم.
به طور کلی EF از دو روش برای ردیابی تغییرات رخ داده شده در اشیا استفاده می‌کند:
1) ردیابی تغییر عکس فوری! (Snapshot change tracking)
2) پروکسی‌های ردیابی تغییر (Change tracking proxies)

ردیابی تغییر عکس فوری

به نظر من، اسم مناسبی برای این روش انتخاب کرده اند و دقیقا بیان گر کاری است که EF انجام می‌دهد. در حالت عادی کلاس‌های دامین ما یا همان کلاس‌های POCO، هیچ منطق و کدی را برای مطلع ساختن EF از تغییراتی که در آن‌ها رخ می‌دهد پیاده سازی نکرده اند. چون هیچ راهی برای EF، برای مطلع شدن از تغییرات رخ داده وجود ندارد، EF راه جالبی را بر می‌گزیند. EF هر گاه شیئی را می‌بیند از مقادیر ویژگی‌های آن یک عکس فوری می‌گیرد! و آن‌ها را در حافظه ذخیره می‌کند.این عمل هنگامی که یک شی از پرس و جو (query) حاصل می‌شود، و یا شیئی را به DbSet اضافه می‌کنیم رخ می‌دهد.
زمانی که EF می‌خواهد بفهمد که چه تغییراتی رخ داده است، مقادیر کنونی موجود در کلیه اشیا را اسکن می‌کند و با مقادیری که در عکس فوری ذخیره کرده است مقایسه می‌کند و متوجه تغییرات رخ داده می‌شود. این فرآیند اسکن کردن کلیه اشیا زمانی رخ می‌دهد که متد DetectChanges ویژگی DbSet.ChangeTracker صدا زده شود.

پروکسی‌های ردیابی تغییر

پروکسی‌های ردیابی تغییر، مکانیزم دیگری برای ردیابی تغییرات EF است و به EF این اجازه را می‌دهد تا از تغییرات رخ داده، مطلع شود.
اگر به یاد داشته باشید در مباحث Lazy loading نیز از واژه پروکسی‌های پویا استفاده شد. پروکسی‌های ردیابی تغییر نیز با استفاده از همان مکانیزم کار می‌کنند و علاوه بر فراهم کردن Lazy loading، این امکان را می‌دهند تا تغییرات را به Context انتقال دهند.
برای استفاده از پروکسی‌های ردیابی تغییر، شما باید ساختار کلاس‌های خود را به گونه ای تغییر دهید، تا EF بتواند در زمان اجرا، نوع پویایی را که هریک، از کلاس‌های POCO شما مشتق می‌شوند ایجاد کند، و تک تک ویژگی‌های آن‌ها را تحریف (override) کند.
این نوع پویا که به عنوان پروکسی پویا نیز شناخته می‌شود، منطقی را در ویژگی‌های تحریف شده شامل می‌شود، تا EF را از تغییرات صورت گرفته در ویژگی هایش مطلع سازد.
برای بیان ادامه‌ی مطلب، من مدل یک دفترچه تلفن ساده را به شرح زیر در نظر گرفتم که روابط مهم و اساسی در آن در نظر گرفته شده است.

```
namespace EntitySample1.DomainClasses
{
    public class Person
    {
        public int Id { get; set; }
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public DateTime BirthDate { get; set; }
        public virtual PersonInfo PersonInfo { get; set; }
        public virtual ICollection<PhoneNumber> PhoneNumbers { get; set; }
        public virtual ICollection<Address> Addresses { get; set; }
    }
}
```

```
namespace EntitySample1.DomainClasses
{
    public class PersonInfo
    {
        public int Id { get; set; }
        public string Note { get; set; }
        public string Major { get; set; }
    }
}
```

```
namespace EntitySample1.DomainClasses
{
    public enum PhoneType
    {
        Home,
        Mobile,
        Work
    }

    public class PhoneNumber
    {
        public int Id { get; set; }
        public string Number { get; set; }
        public PhoneType PhoneType { get; set; }
        public virtual Person Person { get; set; }
    }
}
```

```
namespace EntitySample1.DomainClasses
{
    public class Address
    {
        public int Id { get; set; }
        public string City { get; set; }
        public string Street { get; set; }
        public virtual ICollection<Person> Persons { get; set; }
    }
}
```

طبق کلاس‌های فوق ، ما تعدادی شخص ، اطلاعات شخص ، شماره تلفن و آدرس داریم. رابطه‌ی بین شخص و اطلاعات آن شخص یک به یک، شخص و آدرس چند به چند و شخص با شماره تلفن یک به چند است. همچنین به این نکته توجه داشته باشید که کلیه کلاس‌های فوق به صورت public تعریف، و کلیه خواص راهبری (navigation properties) به صورت virtual تعریف شده اند. دلیل این کار هم این است که این دو مورد، جز الزامات، برای فعال سازی Lazy loading هستند.

تعریف کلاس context نیز به شکل زیر است:

```
namespace EntitySample1.DataLayer
{
    public class PhoneBookDbContext : DbContext
    {
        public DbSet<Person> Persons { get; set; }
        public DbSet<PhoneNumber> PhoneNumbers { get; set; }
        public DbSet<Address> Addresses { get; set; }

    }
}
```

استفاده از ردیابی تغییر عکس فوری

ردیابی تغییر عکس فوری، وابسته به این است که EF بفهمد، چه زمانی تغییرات رخ داده است. رفتار پیش فرض DbContext API ، این هست که به صورت خودکار بازرسی لازم را در نتیجه‌ی رخدادهای DbContext انجام دهد. DetectChanges تنها اطلاعات مدیریت حالت context، که وظیفه‌ی انعکاس تغییرات صورت گرفته به پایگاه داده را دارد، به روز نمی‌کند، بلکه اصلاح رابطه(ralationship) ترکیبی از خواص راهبری مرجع ، مجموعه ای و کلیدهای خارجی را انجام می‌دهد. این خیلی مهم خواهد بود که درک روشنی داشته باشیم از این که چگونه و چه زمانی تغییرات تشخیص داده می‌شوند،چه چیزی باید از آن انتظار داشته باشیم و چگونه کنترلش کنیم.

چه زمانی تشخیص خودکار تغییرات اجرا می‌شود؟

متد DetectChanges کلاس ObjectContext، از EF نسخه‌ی 4 به عنوان بخشی از الگوی ردیابی تغییر عکس فوری اشیای POCO ،در دسترس بوده است. تفاوتی که در مورد DataContext.ChangeTracker.DetectChanges(در حقیقت ObjectContext.DetectChanges فراخوانی می‌شود) وجود دارد این است که، رویدادهای خیلی بیشتری وجود دارند که به صورت خودکار DetectChanges را فراخوانی می‌کنند.

لیستی از متدهایی که باعث انجام عمل تشخیص تغییرات (DetectChanges)، می‌شوند را در ادامه مشاهده می‌کنید:

- DbSet.Add
- DbSet.Find
- DbSet.Remove
- DbSet.Local
- DbSet.SaveChanges
- فراخوانی Linq Query از DbSet
- DbSet.Attach
- DbContext.GetValidationErrors
- DbContext.Entry
- DbChangeTracker.Entries

کنترل زمان فراخوانی DetectChanges

بیشترین زمانی که EF احتیاج به فهمیدن تغییرات دارد، در زمان SaveChanges است، اما حالت‌های زیاد دیگری نیز هست. برای مثال، اگر ما از ردیاب تغییرات، درخواست وضعیت فعلی یک شی را بکنیم،EF احتیاج به اسکن کردن و بررسی تغییرات رخ داده را دارد. همچنین وضعیتی را در نظر بگیرید که شما از پایگاه داده یک شماره تلفن را واکشی می‌کنید و سپس آن را به مجموعه شماره تلفن‌های یک شخص جدید اضافه می‌کنید.آن شماره تلفن اکنون تغییر کرده است، چرا که انتساب آن به یک شخص جدید،خاصیت PersonId آن را تغییر داده است. ولی EF برای اینکه بفهمد تغییر رخ داده است(یا حتی نداده است) ، احتیاج به اسکن کردن همه‌ی اشیا Person دارد.

بیشتر عملیاتی که بر روی DbContext API انجام می‌دهید، موجب فراخوانی DetectChanges می‌شود. در بیشتر موارد DetectChanges به اندازه کافی سریع هست تا باعث ایجاد مشکل کارایی نشود. با این حال ممکن است ، شما تعداد خیلی زیادی اشیا در حافظه داشته باشید، و یا تعداد زیادی عملیات در DbContext ، در مدت خیلی کوتاهی انجام دهید، رفتار تشخیص خودکار تغییرات ممکن است، باعث نگرانی‌های کارایی شود. خوشبختانه گزینه ای برای خاموش کردن رفتار تشخیص خودکار تغییرات وجود دارد و هر زمانی که می‌دانید لازم است، می‌توانید آن را به صورت دستی فراخوانی کنید.

EF بر مبنای این فرض ساخته شده است که شما ، در صورتی که در فراخوانی آخرین API، موجودیتی تغییر پیدا کرده است، قبل از فراخوانی API جدید، باید DetectChanges صدا زده شود. این شامل فراخوانی DetectChanges، قبل از اجرای هر query نیز می‌شود.اگر این عمل ناموفق یا نابجا انجام شود،ممکن است عواقب غیر منتظره ای در بر داشته باشد. DbContext انجام این وظیفه را بر عهده گرفته است و به همین دلیل به طور پیش فرض تشخیص تغییرات خودکار آن فعال است.

نکته: تشخیص اینکه چه زمانی احتیاج به فراخوانی DetectChanges است،آن طور که ساده و بدیهی به نظر می‌آید نیست. تیم EF شدیدا توصیه کرده اند که فقط، وقتی با مشکلات عدم کارایی روبرو شدید، تشخیص تغییرات را به حالت دستی در بیاورید.همچنین توصیه شده که در چنین مواقعی، تشخیص خودکار تغییرات را فقط برای قسمتی از کد که با کارایی پایین مواجه شدید خاموش کنید و پس از اینکه اجرای آن قسمت از کد تمام شد،دوباره آن را روشن کنید.

برای خاموش یا روشن کردن تشخیص خودکار تغییرات، باید متغیر بولین DbContext.Configuration.AutoDetectChangesEnabled را تنظیم کنید.
در مثال زیر، ما در متد ManualDetectChanges، تشخیص خودکار تغییرات را خاموش کرده ایم و تاثیرات آن را بررسی کرده ایم.

```
private static void ManualDetectChanges()
        {
            using (var context = new PhoneBookDbContext())
            {
                context.Configuration.AutoDetectChangesEnabled = false; // turn off Auto Detect Changes

                var p1 = context.Persons.Single(p => p.FirstName == "joe");

                p1.LastName = "Brown";

                Console.WriteLine("Before DetectChanges: {0}", context.Entry(p1).State);

                context.ChangeTracker.DetectChanges(); // call detect changes manually

                Console.WriteLine("After DetectChanges: {0}", context.Entry(p1).State);
            }
        }
```

در کدهای بالا ابتدا تشخیص خودکار تغییرات را خاموش کرده ایم و سپس یک شخص با نام joe را از دیتابیس فراخواندیم و سپس نام خانوادگی آن را به Brown تغییر دادیم. سپس در خط بعد، وضعیت فعلی موجودیت p1 را از context جاری پرسیدیم. در خط بعدی، DetectChanges را به صورت دستی صدا زده ایم و دوباره همان پروسه را برای به دست آوردن وضیعت شی p1، انجام داده ایم. همان طور که می‌بینید ، برای به دست آوردن وضعیت فعلی شی مورد نظر از متد Entry متعلق به ChangeTracker API استفاده می‌کنیم، که در آینده مفصل در مورد آن بحث خواهد شد. اگر شما متد Main را با صدا زدن ManualDetectChanges ویرایش کنید ، خروجی زیر را مشاهده خواهید کرد:

```
Before DetectChanges: Unchanged
After DetectChanges: Modified
```

همان طور که انتظار می‌رفت، به دلیل خاموش کردن تشخیص خودکار تغییرات، context قادر به تشخیص تغییرات صورت گرفته در شی p1 نیست، تا زمانی که متد DetectChanges را به صورت دستی صدا بزنیم. دلیل این که در دفعه اول، ما نتیجه‌ی غلطی مشاهده می‌کنیم، این است که ما قانون را نقض کرده ایم و قبل از صدا زدن هر API ، متد DetectChanges را صدا نزده ایم.
خوشبختانه چون ما در اینجا وضعیت یک شی را بررسی کردیم، با عوارض جانبی آن روبرو نشدیم.

نکته : به این نکته توجه داشته باشید که متد Entry به صورت خودکار، DetectChanges را فراخوانی می‌کند. برای اینکه دانسته بخواهیم این رفتار را غیر فعال کنیم، باید AutoDetectChangesEnabled را غیر فعال کنیم.
در مثال فوق ،خاموش کردن تشخیص خودکار تغییرات، برای ما مزیتی به همراه نداشت و حتی ممکن بود برای ما دردسر ساز شود. ولی حالتی را در نظر بگیرید که ما یک سری API را فراخوانی می‌کنیم ،بدون این که در این بین ،در حالت اشیا تغییری ایجاد کنیم.در نتیجه می‌توانیم از فراخوانی‌های بی جهت DetectChanges جلوگیری کنیم.

در متد AddMultiplePersons مثال بعدی، این کار را نشان داده ام:

```
private static void AddMultiplePerson()
        {
            using (var context = new PhoneBookDbContext())
            {
                context.Configuration.AutoDetectChangesEnabled = false;

                context.Persons.Add(new Person
                    {
                        FirstName = "brad",
                        LastName = "watson",
                        BirthDate = new DateTime(1990, 6, 8)
                    });

                context.Persons.Add(new Person
                {
                    FirstName = "david",
                    LastName = "brown",
                    BirthDate = new DateTime(1990, 6, 8)
                });

                context.Persons.Add(new Person
                {
                    FirstName = "will",
                    LastName = "smith",
                    BirthDate = new DateTime(1990, 6, 8)
                });

                context.SaveChanges();

            }
        }
```

در مثال بالا ما از فراخوانی چهار DetectChanges غیر ضروری که شامل DbSet.Add و SaveChanges می‌شود، جلوگیری کرده ایم.

استفاده از DetectChanges برای فراخوانی اصلاح رابطه

DetectChanges همچنین مسئولیت انجام اصلاح رابطه ، برای هر رابطه ای که تشخیص دهد تغییر کرده است را دارد.اگر شما بعضی از روابط را تغییر دادید و مایل بودید تا همه‌ی خواص راهبری و خواص کلید خارجی را منطبق کنید، DetectChanges این کار را برای شما انجام می‌دهد. این قابلیت می‌تواند برای سناریوهای data-binding که در آن ممکن است در رابط کاربری(UI) یکی از خواص راهبری (یا حتی یک کلید خارجی) تغییر کند، و شما بخواهید که خواص دیگری این رابطه به روز شوند و تغییرات را نشان دهند، مفید واقع شود.

متد DetectRelationshipChanges در مثال زیر از DetectChanges برای انجام اصلاح رابطه استفاده می‌کند.

```
private static void DetectRelationshipChanges()
        {
            using (var context = new PhoneBookDbContext())
            {

                var phone1 = context.PhoneNumbers.Single(x => x.Number == "09351234567");

                var person1 = context.Persons.Single(x => x.FirstName == "will");

                person1.PhoneNumbers.Add(phone1);

                Console.WriteLine("Before DetectChanges: {0}", phone1.Person.FirstName);

                context.ChangeTracker.DetectChanges(); // ralationships fix-up

                Console.WriteLine("After DetectChanges: {0}", phone1.Person.FirstName);
            }
        }
```

در اینجا ابتدا ما شماره تلفنی را از دیتابیس لود می‌کنیم. سپس شخص دیگری را نیز با نام will از دیتابیس می‌خوانیم. قصد داریم

شماره تلفن خوانده شده را به این شخص نسبت دهیم و مجموعه شماره تلفن‌های وی اضافه کنیم و ما این کار را با افزودن phone1 به مجموعه شماره تلفن‌های person1 انجام داده ایم. چون ما از اشیای POCO استفاده کرده ایم،EF نمی‌فهمد که ما این تغییر را ایجاد کرده ایم و در نتیجه کلید خارجی PersonId شی phone1 را اصلاح نمی‌کند. ما می‌توانیم تا زمانی صبر کنیم تا متدی مثل SaveChanges، متد DetectChanges را فراخوانی کند،ولی اگر بخواهیم این عمل در همان لحظه انجام شود، می‌توانیم DetectChanges را دستی صدا بزنیم.

اگر ما متد Main را با اضافه کردن فرخوانی DetectRealtionShipsChanges تغییر بدهیم و آن را اجرا کنیم، نتیجه زیر را مشاهده می‌کنید:

```
Before DetectChanges: david
After DetectChanges: will
```

تا قبل از فراخوانی تشخیص تغییرات(DetectChanegs)، هنوز phone1 منتسب به شخص قدیمی(david) بوده، ولی پس از فراخوانی DetectChanges ، اصلاح رابطه رخ داده و همه چیز با یکدیگر منطبق می‌شوند.

فعال سازی و کار با پروکسی‌های ردیابی تغییر

اگر پروفایلر کارایی شما، فراخوانی‌های بیش از اندازه DetectChnages را به عنوان یک مشکل شناسایی کند، و یا شما ترجیح می‌دهید که اصلاح رابطه به صورت بلادرنگ صورت گیرد ، ردیابی تغییر پروکسی‌های پویا، به عنوان گزینه ای دیگر مطرح می‌شود.فقط با چند تغییر کوچک در کلاس‌های POCO، EF قادر به ساخت پروکسی‌های پویا خواهد بود.پروکسی‌های ردیابی تغییر به EF اجازه ردیابی تغییرات در همان لحظه ای که ما تغییری در اشیای خود می‌دهیم را می‌دهند و همچنین امکان انجام اصلاح رابطه را در هر زمانی که تغییرات روابط را تشخیص دهد، دارد.
برای اینکه پروکسی ردیابی تغییر بتواند ساخته شود، باید قوانین زیر رعایت شود:

- کلاس باید public باشد و seald نباشد.
- همه‌ی خواص(properties) باید virtual تعریف شوند.
- همه‌ی خواص باید getter و setter با سطح دسترسی public داشته باشند.
- همه‌ی خواص راهبری مجموعه ای باید نوعشان، از نوع ICollection<T> تعریف شوند.

کلاس Person مثال خود را به گونه ای بازنویسی کرده ایم که تمام قوانین فوق را پیاده سازی کرده باشد.

نکته: توجه داشته باشید که ما دیگر در داخل سازنده کلاس ،کدی نمی‌نویسیم و منطقی که باعث نمونه سازی اولیه خواص راهبری می‌شدند، را پیاده سازی نمی‌کنیم. این پروکسی ردیاب تغییر، همه‌ی خواص راهبری مجموعه ای را تحریف کرده و ار نوع مجموعه ای مخصوص خود(EntityCollection<T>) استفاده می‌کند. این نوع مجموعه ای، هر تغییری که در این مجموعه صورت می‌گیرد را زیر نظر گرفته و به ردیاب تغییر گزارش می‌دهد. اگر تلاش کنید تا نوع دیگری مانند List<T> که معمولا در سازنده کلاس از آن استفاده می‌کردیم را به آن انتساب دهیم، پروکسی، استثنایی را پرتاب می‌کند.

```
namespace EntitySample1.DomainClasses
{
    public class Person
    {
        public virtual int Id { get; set; }
        public virtual string FirstName { get; set; }
        public virtual string LastName { get; set; }
        public virtual DateTime BirthDate { get; set; }
        public virtual PersonInfo PersonInfo { get; set; }
        public virtual ICollection<PhoneNumber> PhoneNumbers { get; set; }
        public virtual ICollection<Address> Addresses { get; set; }
    }
}
```

همان طور که در مباحث مربوط به Lazy loading نیز مشاهده کردید،EF زمانی پروکسی‌های پویا را برای یک کلاس ایجاد می‌کند که یک یا چند خاصیت راهبری آن با virtual علامت گذاری شده باشند.آن پروکسی‌ها که از کلاس مورد نظر، مشتق شده اند، به خواص راهبری virtual امکان می‌دهند تا به صورت lazy لود شوند.پروکسی‌های ردیابی تغییر نیز به همان شکل در زمان اجرا ایجاد می‌شوند، با این تفاوت که این پروکسی ها، امکانات بیشتری دارند.

با این که احتیاجات رسیدن به پروکسی‌های ردیابی تغییر خیلی ساده هستند، اما ساده‌تر از آن ها، فراموش کردن یکی از آن هاست.حتی از این هم ساده‌تر می‌شود که در آینده تغییری در آن کلاس‌ها ایجاد کنید و ناخواسته یکی از آن قوانین را نقض کنید.به این خاطر، فکر خوبیست که یک آزمون واحد نیز اضافه کنیم تا مطمئن شویم که EF توانسته، پروکسی ردیابی تغییر را ایجاد کند یا نه.

در مثال زیر یک متد نوشته شده که این مورد را مورد آزمایش قرار می‌دهد. همچنین فراموش نکنید که فضای نام System.Data.Object.DataClasses را به usingهای خود اضافه کنید.

```
private static void TestForChangeTrackingProxy()
        {
            using (var context = new PhoneBookDbContext())
            {
                var person = context.Persons.First();
                var isProxy = person is IEntityWithChangeTracker;
                Console.WriteLine("person is a proxy: {0}", isProxy);
            }
        }
```

زمانی که EF ، پروکسی پویا برای ردیابی تغییر ایجاد می‌کند، اینترفیس IentityWithChangeTrackerرا پیاده سازی خواهد کرد.متد تست در مثال بالا، نمونه ای از Person را با دریافت آن از دیتابیس ایجاد می‌کند و سپس آن را با اینترفیس ذکر شده چک می‌کند تا مطمئن شود که Person ، توسط پروکسی ردیابی تغییر احاطه شده است. این نکته را نیز به یاد داشته باشید که چک کردن این که EF ، کلاس پروکسی ای که از کلاس ما مشتق شده است ایجاد کرده است یا نه،کفایت نمی‌کند، چرا که پروکسی‌های Lazy Loading نیز چنین کاری انجام می‌دهند. در حقیقت آن چیزی که سبب می‌شود EF به تغییرات صورت گرفته به صورت بلادرنگ گوش دهد،حضور IEntityWithChangeTracker است.

اکنون متد ManualDetectChanges را که کمی بالاتر بررسی کرده ایم را در نظر بگیرید و کد context.ChangeTracker.DetectChanges آن را حذف کنید و بار دیگر آن را فرا بخوانید و نتیجه را مشاهده کنید:

```
Before DetectChanges: Modified
After DetectChanges: Modified
```

این دفعه،EF از تغییرات صورت گرفته آگاه است،حال چه DetectChanges فراخوانده شود یا نشود.

اکنون متد DetectRelationshipChanges را ویرایش کرده و برنامه را اجرا کنید:

```
Before DetectChanges: will
After DetectChanges: will
```

این بار می‌بینیم که EF، تغییر رابطه را تشخیص داده و اصلاح رابطه را بدون فراخوانی DetectChanges انجام داده است.

نکته: زمانی که شما از پروکسی‌های ردیابی تغییر استفاده می‌کنید،احتیاجی به غیرفعال کردن تشخیص خودکار تغییرات نیست. DetectChanges برای همه اشیایی که تغییرات را به صورت بلادرنگ گزارش می‌دهند،فرآیند تشخیص تغییرات را انجام نمی‌دهد. بنابراین فعال سازی پروکسی‌های ردیابی تغییر،برای رسیدن به مزایای کارایی بالا در هنگام عدم استفاده از DetectChanges کافی است. در حقیقت زمانی که EF، یک پروکسی ردیابی پیدا می‌کند، از مقادیر خاصیت ها، عکس فوری نمی‌گیرد. همچنین DetectChanges این را نیز می‌داند که نباید تغییرات موجودیت هایی که عکسی از مقادیر اصلی آنها ندارد را اسکن کند.

تذکر: اگر شما موجودیت هایی داشته باشید که شامل انواع پیچیده(Complex Types) می‌شوند،EF هنوز هم از ردیابی تغییر عکس فوری، برای خواص موجود در نوع پیچیده استفاده می‌کند، و از این جهت لازم است کهEF، برای نمونه‌ی نوع پیچیده، پروکسی ایجاد نمی‌کند.شما هنوز هم، تشخیص خودکار تغییرات خواصی که مستقیما درون آن موجودیت(Entity) تعریف شده اند را دارید، ولی تغییرات رخ داده درون نوع پیچیده، فقط از طریق DetectChanges قابل تشخیص است.

چگونگی اطمینان از اینکه نمونه‌های جدید ، پروکسی‌ها را دریافت خواهند کرد

EF به صورت خودکار برای نتایج حاصل از کوئری هایی که شما اجرا می‌کنید، پروکسی‌ها را ایجاد می‌کند. با این حال اگر شما فقط از سازنده‌ی کلاس POCO خود برای ایجاد نمونه‌ی جدید استفاده کنید،دیگر پروکسی‌ها ایجاد نخواهند شد.بدین منظور برای دریافت پروکسی ها، شما باید از متد DbSet.Create برای دریافت نمونه‌های جدید آن موجودیت استفاده کنید.
نکته: اگر شما، پروکسی‌های ردیابی تغییر را برای موجودیتی از مدلتان فعال کرده باشید،هنوز هم می‌توانید،نمونه‌های فاقد پروکسی آن موجودیت را ایجاد و بیافزایید.خوشبختانه EF با موجودیت‌های پروکسی و غیر پروکسی در همان مجموعه(set) کار می‌کند.شما باید آگاه باشید که ردیابی خودکار تغییرات و یا اصلاح رابطه، برای نمونه هایی که پروکسی هایی ردیابی تغییر نیستند، قابل استفاده نیستند.داشتن مخلوطی از نمونه‌های پروکسی و غیر پروکسی در همان مجموعه، می‌تواند گیج کننده باشد.بنابر این عموما توصیه می‌شود که برای ایجاد نمونه‌های جدید از DbSet.Create استفاده کنید، تا همه‌ی موجودیت‌های موجود در مجموعه، پروکسی‌های ردیابی تغییر باشند.
متد CreateNewProxies را به برنامه‌ی خود اضافه کرده و آن را اجرا کنید.

```
private static void CreateNewProxies()
        {
            using (var context = new PhoneBookDbContext())
            {
                var phoneNumber = new PhoneNumber { Number = "987" };

                var davidPersonProxy = context.Persons.Create();
                davidPersonProxy.FirstName = "david";
                davidPersonProxy.PhoneNumbers.Add(phoneNumber);

                Console.WriteLine(phoneNumber.Person.FirstName);
            }
        }
```

خروجی مثال فوق david خواهد بود.همان طور که می‌بینید با استفاده از context.Persons.Create، نمونه‌ی ساخته شده، دیگر شی POCO نیست، بلکه davidPersonProxy، از جنس پروکسی ردیابی تغییر است و تغییرات آن به طور خودکار ردیابی شده و رابطه آن نیز به صورت خودکار اصلاح می‌شود.در اینجا نیز با افزودن phoneNumber به شماره تلفن‌های davidPersonProxy، به طور خودکار رابطه‌ی بین phoneNumber و davidPersonPeroxy بر قرار شده است. همان طور که می‌دانید این عملیات بدون استفاده از پروکسی‌های ردیابی تغییرات امکان پذیر نیست و موجب بروز خطا می‌شود.

ایجاد نمونه‌های پروکسی برای انواع مشتق شده

اورلود جنریک دیگری برای DbSet.Create وجود دارد که برای نمونه سازی کلاس‌های مشتق شده در مجموعه ما استفاده می‌شود .برای مثال، فراخوانی Create بر روی مجموعه‌ی Persons ،نمونه ای از کلاس Person را بر می‌گرداند.ولی ممکن است کلاس هایی در مجموعه‌ی Persons وجود داشته باشند، که از آن مشتق شده باشند، مانند Student. برای دریافت نمونه‌ی پروکسی Student، از اورلود جنریک Create استفاده می‌کنیم.

```
var newStudent = context.Persons.Create<Student>();
```

واکشی موجودیت‌ها بدون ردیابی تغییرات

تا به این جای کار باید متوجه شده باشید که ردیابی تغییرات، فرآیندی ساده و بدیهی نیست و مقداری سربار در کار است. در بعضی از بخش‌های برنامه تان، احتمالا داده‌ها را به صورت فقط خواندنی در اختیار کاربران قرار می‌دهید و چون اطلاعات هیچ وقت تغییر نمی‌کنند، شما می‌خواهید که سربار ناشی از ردیابی تغییرات را حذف کنید.
خوشبختانه EF شامل متد AsNoTracking است که می‌توان از آن برای اجرای کوئری‌های بدون ردیابی استفاده کرد.یک کوئری بدون ردیابی، یک کوئری ساده هست که نتایج آن توسط context برای تشخیص تغییرات ردیابی نخواهد شد.

متد PrintPersonsWithoutChangeTracking را به برنامه اضافه کنید و آن را اجرا کنید:

```
private static void PrintPersonsWithoutChangeTracking()
        {
            using (var context = new PhoneBookDbContext())
            {
                var persons = context.Persons.AsNoTracking().ToList();

                foreach (var person in persons)
                {
                    Console.WriteLine(person.FirstName);
                }

            }
        }
```

در مثال بالا از متد AsNoTracking برای گرفتن کوئری فاقد ردیابی استفاده کردیم تا محتویات مجموعه Persons را دریافت کنیم. در نهایت با یک حلقه‌ی foreach، نتایج را بر روی کنسول به نمایش در آوردیم.به دلیل اینکه، این یک کوئری بدون ردیابی هست، context دیگر تغییراتی که روی Persons رخ می‌دهد را ردیابی نمی‌کند.در نتیجه اگر شما یکی از خواص یکی از Persons را تغییردهید و SaveChanges را صدا بزنید، تغییرات به دیتابیس ارسال نمی‌شوند.

نکته: واکشی داده‌ها بدون ردیابی تغییرات،معمولا وقتی باعث افزایش قابل توجه کارایی می‌شود که بخواهیم تعداد خیلی زیادی داده را به صورت فقط خواندنی نمایش دهیم. اگر برنامه‌ی شما داده ای را تغییر می‌دهد و می‌خواهد آن را ذخیره کند، باید از AsNoTracking استفاده نکنید.

AsNoTracking یک متد الحاقی است، که در IQueryable<T> تعریف شده است، در نتیجه شما می‌توانید از آن، در کوئری‌های LINQ نیز استفاده کنید. شما می‌توانید از AsNoTracking، در انتهای DbSet ،در خط from کوئری استفاده کنید.

```
var query = from p in context.Persons.AsNoTracking()
                            where p.FirstName == "joe"
                            select p;
```

شما همچنین از AsNoTracking می‌توانید برای تبدیل یک کوئری LINQ موجود، به یک کوئری فاقد ردیابی استفاده کنید. این نکته را به یاد داشته باشید که فقط AsNoTracking بر روی کوئری، فرانخوانده شده است، بلکه متغیر query را با نتیجه‌ی حاصل از فراخوانی AsNoTracking بازنویسی(override) کرده است و این، از این جهت لازم است که AsNoTracking ،تغییری در کوئری ای که بر روی آن فراخوانده شده نمی‌دهد، بلکه یک کوئری جدید بر می‌گرداند.

```
var query = from p in context.Persons
                            where p.FirstName == "joe"
                            select p;
                query = query.AsNoTracking();
```

نکته: به دلیل اینکه AsNoTracking یک متد الحاقی است، شما احتیاج به افزودن فضای نام System.Data.Entity به فضاهای نام خود دارید.

منبع: ترجمه ای آزاد از کتاب Programming Entity Framework: DbContext

نظرات خوانندگان

نویسنده: امیر خلیلی
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۳:۳۸

یعنی با یکی از 2 روش گفته شده در بالا میتوان دیتابیس را مانیتور کرد و از تغییرات ایجاد شده در دیتابیس در همان لحظه با خبر شد ؟ مثلا ثبت نام یک کاربر جدید , یا ارسال یک نظر جدید و همان لحظه نمایش یک پیغام در صفحه ادمین؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۳:۴۶

خیر. در ORMها کلا ردیابی منظور ردیابی تغییرات انجام شده در اشیایی است که در حال کار با آن‌ها هستیم آن هم در طی یک Context موجود. مثلا در یک Context باز شده و فعال، یک شیء اضافه می‌شود. دو خاصیت شیء ایی دیگر ویرایش می‌شوند. دو شیء دیگر نیز حذف خواهند شد. اینجا است که ORM باید بتواند این موارد و تغییرات را ردیابی کرده و سپس SQL صحیح و بهینه‌ای را جهت اعمال بر روی بانک اطلاعاتی تولید کند.

نویسنده: امیر خلیلی
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۳:۵۳

خیلی ممنون از جوابتون
اگه امکان داره لطف بفرمایین با یک راهنمایی کوچک که برای مانیتور دیتابیس و اون هدفی که در بالا گفتم از چه روشی میتوان استفاده کرد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۳:۵۹

دوره « معرفی SignalR و ارتباطات بلادرنگ » در سایت می‌تونه شروع خوبی باشه.

نویسنده: سوین
تاریخ: ۱۳۹۲/۰۹/۰۹ ۰:۴۴

با سلام
من قبلا در EF 4 برای ذخیره اطلاعات با استفاده از دیتاگرید در WPF App می‌اومدم یه Context ایجاد می‌کردم و اطلاعات رو از جدول به صورت IQueryable به ItemSource دیتاگراید بایند می‌کردم و بعد از تغییر اطلاعات در انتها با یه SaveChange تغییرات رو تو دیتابیس ذخیره می‌شد اما الان در EF 6 این خطا رو می‌ده

```
Data binding directly to a store query (DbSet, DbQuery, DbSqlQuery) is not supported. Instead populate a DbSet with data, for example by calling Load on the DbSet, and then bind to local data. For WPF bind to DbSet.Local. For WinForms bind to DbSet.Local.ToBindingList().
```

ممنون میشم راهنماییم کنید .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۹ ۰:۵۸

استفاده از خاصیت Local در Entity Framework
مدیریت تغییرات گریدی از اطلاعات به کمک استفاده از الگوی واحد کار مشترک بین ViewModel و لایه سرویس

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۲/۰۸ ۱۷:۸

با این تفاسیر الان ما باید خاصیت‌های کلاس هامون، مثلا Id رو هم به صورت virtual تعریف کنیم؟ یا لزومی نداره؟

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۱۲/۰۹ ۸:۳۳

به شخصه من این کار را انجام میدهم. ولی یادم هست که در یک پروژه و در یک سناریوی خاص Entity framework یک استثنا صادر می‌کرد که با جست و جو در اینترنت، یکی از اعضای توسعه دهنده‌ی تیم Entity framework گفته بود که در این سناریو، Entity framework توانایی کار با تمام اعضای virtual را ندارد.

البته این موضوع به به نسخه‌ی 4.3 بر میگرده و احتمالش هست که اشکالش در نسخه‌های بعد رفع شده باشد.

از نظر شخصی خودم در پروژه هاتون به خصوص پروژه‌های ویندوزی به عنوان یک best practice همه‌ی اعضا را virtual تعریف کنید مگر اینکه به مشکل بر بخورید.

عنوان: چند نکته کاربردی درباره Entity Framework
نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۴:۵۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, EF Code First

1) رفتار متصل و غیر متصل در EF چیست؟

اولین نکته ای که به ذهنم می‌رسه اینه که برای استفاده از EF حتما باید درک صحیحی از رفتارها و قابلیت‌های اون داشته باشیم. نحوه استفاده ازEF رو به دو رفتار متصل و غیر متصل تقسیم می‌کنیم.

حالت پیش فرضEF بر مبنای رفتار متصل می‌باشد. در این حالت شما یک موجودیت رو از دیتابیس فرا می‌خونید EF این موجودیت رو ردگیری می‌کنه اگه تغییری در اون مشاهده کنه بر روی اون برچسب "تغییر داده شد" می‌زنه و حتی اونقدر هوشمند هست که وقتی تغییرات رو ذخیره می‌کنید کوئری آپدیت رو فقط براساس فیلدهای تغییر یافته اجرا کنه. یا مثلا در صورتی که شما بخواهید به یک خاصیت رابطه ای دسترسی پیدا کنید اگر قبلا لود نشده باشه در همون لحظه از دیتابیس فراخوانی میشه، البته این رفتارها هزینه بر خواهد بود و در تعداد زیاد موجودیت‌ها میتونه کارایی رو به شدت پایین بیاره.

رفتار متصل شاید در ویندوز اپلیکیشن کاربرد داشته باشه ولی در حالت وب اپلیکیشن کاربردی نداره چون با هر در خواستی به سرور همه چیز از نو ساخته میشه و پس از پاسخ به درخواست همه چی از بین میره. پس DbContext همیشه از بین می‌ره و ما برحسب نیاز، در درخواست‌های جدید به سرور ، دوباره DbContext رو می‌سازیم. پس از ساخته شدن DbContext باید موجودیت مورد استفاده رو به اون معرفی کنیم و وضعیت اون موجودیت رو هم مشخص کنیم.(جدید ، تغییر یافته، حذف ، بدون تغییر) در این حالت سیستم ردگیری تغییرات بی استفاده است و ما فقط در حال هدر دادن منابع سیستم هستیم.

در حالت متصل ما باید همیشه از یک DbContext استفاده کنیم و همه موجودیت‌ها در آخر باید تحت نظر این DbContext باشند در یک برنامه واقعی کار خیلی سخت و پیچیده ای است. مثلا بعضی وقت‌ها مجبور هستیم از موجودیت هایی که قبلا در حافظه برنامه بوده اند استفاده کنیم اگر این موجودیت در حافظه DbContext جاری وجود نداشته باشه با معرفی کردن اون از طریق متد attach کار ادامه پیدا می‌کنه ولی اگر قبلا موجودیتی در سیستم ردگیری DbContext با همین شناسه وجود داشته باشد با خطای زیر مواجه می‌شویم.

```
An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot
track multiple objects with the same key
```

این خطا مفهوم ساده و مشخصی داره ، دو شی با یک شناسه نمی‌توانند در یک DbContext وجود داشته باشند. معمولا در این حالت ما بااین اشیا تکراری کاری نداریم و فقط به شناسه اون شی برای نشان دادن روابط نیاز داریم و از دیگر خاصیت‌های اون جهت نمایش به کاربر استفاده می‌کنیم ولی متاسفانه DbContext نمی‌دونه چی تو سر ما می‌گذره و فقط حرف خودشو می‌زنه! البته اگه خواستید با DbContext بر سر این موضوع گفتگو کنید از کدهای زیر استفاده کنید:

```
T attachedEntity = set.Find(entity.Id);
var attachedEntry = dbContext.Entry(attachedEntity);
attachedEntry.CurrentValues.SetValues(entity);
```

خوب با توجه به صحبت‌های بالا اگر بخواهیم از رفتار غیر متصل استفاده کنیم باید تنظیمات زیر رو به متد سازنده DbContext اضافه کنیم. از اینجا به بعد همه چیز رو خودمون در اختیار می‌گیریم و ما مشخص می‌کنیم که کدوم موجودیت باید چه وضعیتی داشته باشه (افزودن ، بروز رسانی، حذف)و اینکه چه موقع روابط خودش را با دیگر موجودیتها فراخوانی کنه.

```
public DbContext()
        {
            this.Configuration.ProxyCreationEnabled = false;
            this.Configuration.LazyLoadingEnabled = false;
            this.Configuration.AutoDetectChangesEnabled = false;
         }
```

2) تعیین وضعیت یک موجودیت و راوبط آن در EF چگونه است؟

با کد زیر می‌تونیم وضعیت یک موجدیت رو مشخص کنیم ، با اجرای هر یک از دستورات زیر موجودیت تحت نظر DbContext

قرار می‌گیره یعنی عمل attach نیز صورت گرفته است :

```
dbContext.Entry(entity).State = EntityState.Unchanged ;
dbContext.Entry(entity).State = EntityState.Added ; //or  Dbset.Add(entity)
dbContext.Entry(entity).State = EntityState.Modified ;
dbContext.Entry(entity).State = EntityState.Deleted ; // or  Dbset.Remove(entity)
```

با اجرای این کد موجودیت از سیستم ردگیری DbContext خارج می‌شه.

```
dbContext.Entry(entity).State = EntityState.Detached;
```

در موجودیت‌های ساده با دستورات بالا نحوه ذخیره سازی را مشخص می‌کنیم در وضعیتی که با موجودیت‌های رابطه ای سروکار داریم باید به نکات زیر توجه کنیم.

در نظر بگیرید یک گروه از قبل وجود دارد و ما مشتری جدیدی می‌سازیم در این حالت انتظار داریم که فقط یک مشتری جدید ذخیره شده باشد:

```
// group id=19  Name="General"
var customer = new Customer();
customer.Group = group;
customer.Name = "mohammadi";
dbContext.Entry(customer).State = EntityState.Added;
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Added
var groupstate = dbContext.Entry(group);// groupstate=EntityState.Added
```

اگه از روش بالا استفاده کنید می‌بینید گروه General جدیدی به همراه مشتری در دیتابیس ساخته می‌شود.نکته مهمی که اینجا وجود داره اینه که DbContext به id موجودیت گروه توجهی نداره ، برای جلو گیری از این مشکل باید قبل از معرفی موجودیت‌های جدید رابطه هایی که از قبل وجود دارند را به صورت بدون تغییر attach کنیم و بعد وضعیت جدید موجودیت رو اعمال کنیم.

```
// group id=19  Name="General"
var customer = new Customer();
customer.Group = group;
 customer.Name = "mohammadi";
dbContext.Entry(group).State = EntityState.Unchanged;
dbContext.Entry(customer).State = EntityState.Added;
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Added
 var groupstate = dbContext.Entry(group);// groupstate=EntityState.Unchanged
```

در مجموع بهتره که موجودیت ریشه رو attach کنیم و بعد با توجه به نیاز تغییرات رو اعمال کنیم.

```
// group id=19  Name="General"
var customer = new Customer();
 customer.Group = group;
customer.Name = "mohammadi";
dbContext.Entry(customer).State = EntityState.Unchanged;
dbContext.Entry(customer).State = EntityState.Added;
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Added
var groupstate = dbContext.Entry(group);//// groupstate=EntityState.Unchanged
```

3) AsNoTracking و Include دو ابزار مهم در رفتار غیر متصل:

درصورتیکه ما تغییراتی روی داده‌ها نداشته باشیم و یا از روش‌های غیر متصل از موجودیت‌ها استفاده کنیم با استفاده از متد AsNoTracking() در زمان و حافظه سیستم صرف جویی می‌کنیم در این حالت موجودیت‌های فراخوانی شده از دیتابیس در سیستم

ردگیری DbContext قرار نمی‌گیرند و اگر وضعیت آنها را بررسی کنیم در وضعیت Detached قرار دارند.

```
var customer  = dbContext.Customers.FirstOrDefault();
 var customerAsNoTracking  = dbContext.Customers.AsNoTracking().FirstOrDefault();
var customerstate = dbContext.Entry(customer).State;// customerstate=EntityState.Unchanged
var customerstateAsNoTracking  = dbContext.Entry(customerAsNoTracking).State;//
customerstate=EntityState.Detached
```

نحوه بررسی کردن موجودیت‌های موجود در سیستم ردگیری DbContext :

```
var Entries = dbContext.ChangeTracker.Entries();
var AddedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Added);
var ModifiedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Modified);
var UnchangedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Unchanged);
var DeletedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Deleted);
var DetachedEntries = dbContext.ChangeTracker.Entries().Where(entityEntry =>
entityEntry.State==EntityState.Detached);//* not working !
```

* در نظر داشته باشید وضعیت Detached وجود خارجی ندارد و به حالتی گفته می‌شود که DbContext در سیستم رد گیری خود اطلاعی از موجودیت مورد نظر نداشته باشد.

وقتی که سیستم فراخوانی خودکار رابطه‌ها خاموش باشد باید موقع فراخوانی موجودیت‌ها روابط مورد نیاز را هم با دستور Include در سیستم فراخوانی کنیم.

```
var CustomersWithGroup = dbContext.Customers.AsNoTracking().Include("Group").ToList();
 var CustomerFull =
dbContext.Customers.AsNoTracking().Include("Group").Include("Bills").Include("Bills.BillDetails").ToLis
t();
```

4) از متد AddOrUpdate در در فضای نام System.Data.Entity.Migrations استفاده نکنیم، چرا؟

در صورتی که از فیلد RowVersion و کنترل مسایل همزمانی استفاده کرده باشیم هر وقتی متد AddOrUpdate رو فراخوانی کنیم، تغییر اطلاعات توسط دیگر کاربران نادیده گرفته می‌شود. با توجه به این که متد AddOrUpdate برای عملیات Migrations در نظر گرفته شده است، این رفتار کاملا طبیعی است. برای حل این مشکل می‌تونیم این متد رو با بررسی شناسه به سادگی پیاده سازی کنیم:

```
public virtual void AddOrUpdate(T entity)
        {
            if (entity.Id == 0)
                Add(entity);
            else
                Update(entity);

        }
```

5) اگر بخواهیم موجودیت‌های رابطه ای در دیتا گرید ویو (ویندوز فرم) نشون بدیم باید چه کار کنیم؟

گرید ویو در ویندوز فرم قادر به نشون دادن فیلدهای رابطه ای نیست برای حل این مشکل می‌تونیم یک نوع ستون جدید برای گرید ویو تعریف کنیم و برای نشون دادن فیلدهای رابطه ای از این نوع ستون استفاده کنیم:

```
public class DataGridViewChildRelationTextBoxCell : DataGridViewTextBoxCell
    {
        protected override object GetValue(int rowIndex)
        {
            try
            {
```

```
                var bs = (BindingSource)DataGridView.DataSource;
                var cl = (DataGridViewChildRelationTextBoxColumn)DataGridView.Columns[ColumnIndex];
                return getChildValue(bs.List[rowIndex], cl.DataPropertyName).ToString();
            }
            catch (Exception)
            {
                return "";
            }
        }

        private object getChildValue(object dataSource, string childMember)
        {
            int nextPoint = childMember.IndexOf('.');
            if (nextPoint == -1) return
dataSource.GetType().GetProperty(childMember).GetValue(dataSource, null);
            string proName = childMember.Substring(0, nextPoint);
            object newDs = dataSource.GetType().GetProperty(proName).GetValue(dataSource, null);
            return getChildValue(newDs, childMember.Substring(nextPoint + 1));
        }
    }

    public class DataGridViewChildRelationTextBoxColumn : DataGridViewTextBoxColumn
    {
        public string DataMember { get; set; }

        public DataGridViewChildRelationTextBoxColumn()
        {
            CellTemplate = new DataGridViewChildRelationTextBoxCell();
        }
    }
```

نحوه استفاده را در ادامه می‌بینید. این روش توسط ویزارد گریدویو هم قابل استفاده است. موقع Add کردن Column نوع اون رو روی DataGridViewChildRelationTextBoxColumn تنظیم کنید.

```
GroupNameColumn= new DataGridViewChildRelationTextBoxColumn(); //from your class
GroupNameColumn.HeaderText = "گروه مشتری";
GroupNameColumn.DataPropertyName = "Group.Name"; //EF  Property: Customer.Group.Name
GroupNameColumn.Visible = true;
GroupNameColumn.Width = 300;
DataGridView.Columns.Add(GroupNameColumn);
```

نظرات خوانندگان

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۶:۹

سلام ... مطلب بسیار کاربردی بود ...
در برنامه‌های وب با استفاده از ابزارهایی مثه Stucturemap میتوان DbContext رو به httpContext محدود کرد که فک میکنم باعث رفتار متصل میشه !

با توجه به این مقاله (قسمت **تعیین طول عمر اشیاء در StructureMap**)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۳

در حالت Detached (مثل ایجاد یک شیء CLR ساده)
در متد Updateایی که نوشتید، قسمت Find حتما اتفاق می‌افته. چون Tracking خاموش هست (مطابق تنظیماتی که عنوان کردید)، بنابراین Find چیزی رو از کشی که وجود نداره نمی‌تونه دریافت کنه و میره سراغ دیتابیس. ماخذ :

```
The Find method on DbSet uses the primary key value to attempt to find an entity tracked by the
context.
If the entity is not found in the context then a query will be sent to the database to find the entity
there.
Null is returned if the entity is not found in the context or in the database.
```

حالا تصور کنید که در یک حلقه می‌خواهید 100 آیتم رو ویرایش کنید. یعنی 100 بار رفت و برگشت خواهید داشت با این متد Update سفارشی که ارائه دادید. البته منهای کوئری‌های آپدیت متناظر. این 100 تا کوئری فقط Find است.
قسمت Find متد Update شما در حالت detached اضافی است. یعنی اگر می‌دونید که این Id در دیتابیس وجود داره نیازی به Findاش نیست. فقط State اون رو تغییر بدید کار می‌کنه .

در حالت نه آنچنان Detached ! (دریافت یک لیست از Context ایی که ردیابی نداره)
با خاموش کردن Tracking حتما نیاز خواهید داشت تا متد context.ChangeTracker.DetectChanges رو هم پیش از ذخیره سازی یک لیست دریافت شده از بانک اطلاعاتی فراخوانی کنید. وگرنه چون این اطلاعات ردیابی نمی‌شوند، هر تغییری در آن‌ها، وضعیت Unchanged رو خواهد داشت و نه Detached. بنابراین SaveChanges عمل نمی‌کنه؛ مگر اینکه DetectChanges فراخوانی بشه.

سؤال: این سربار که می‌گن چقدر هست؟ ارزشش رو داره که راسا خاموشش کنیم؟ یا بهتره فقط برای گزارشگیری این کار رو انجام بدیم؟
یک آزمایش:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Diagnostics;
using System.Linq;

namespace EF_General.Models.Ex21
{
    public abstract class BaseEntity
    {
        public int Id { set; get; }
    }

    public class Factor : BaseEntity
    {
        public int TotalPrice { set; get; }
    }
```

```
    public class MyContext : DbContext
    {
        public DbSet<Factor> Factors { get; set; }

        public MyContext() { }
        public MyContext(bool withTracking)
        {
            if (withTracking)
                return;

            this.Configuration.ProxyCreationEnabled = false;
            this.Configuration.LazyLoadingEnabled = false;
            this.Configuration.AutoDetectChangesEnabled = false;
        }

        public void CustomUpdate<T>(T entity) where T : BaseEntity
        {
            if (entity == null)
                throw new ArgumentException("Cannot add a null entity.");


            var entry = this.Entry<T>(entity);
            if (entry.State != EntityState.Detached)
                return;

            /*var set = this.Set<T>(); // این‌ها اضافی است
            //متد فایند اگر اینجا باشه حتما به بانک اطلاعاتی رجوع می‌کنه در حالت منقطع از زمینه و در یک حلقه
به روز رسانی کارآیی مطلوبی نخواهد داشت
            T attachedEntity = set.Find(entity.Id);
            if (attachedEntity != null)
            {
                var attachedEntry = this.Entry(attachedEntity);
                attachedEntry.CurrentValues.SetValues(entity);
            }
            else
            {*/
            entry.State = EntityState.Modified;
            //}
        }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            if (!context.Factors.Any())
            {
                for (int i = 0; i < 20; i++)
                {
                    context.Factors.Add(new Factor { TotalPrice = i });
                }
            }
            base.Seed(context);
        }
    }

    public class Performance
    {
        public TimeSpan ListDisabledTracking { set; get; }
        public TimeSpan ListNormal { set; get; }
        public TimeSpan DetachedEntityDisabledTracking { set; get; }
        public TimeSpan DetachedEntityNormal { set; get; }
    }

    public static class Test
    {
        public static void RunTests()
        {
            startDb();

            var results = new List<Performance>();
            var runs = 20;
            for (int i = 0; i < runs; i++)
            {
```

```
                Console.WriteLine("\nRun {0}", i + 1);

                var tsListDisabledTracking = PerformanceHelper.RunActionMeasurePerformance(() =>
updateListTotalPriceDisabledTracking());
                var tsListNormal = PerformanceHelper.RunActionMeasurePerformance(() =>
updateListTotalPriceNormal());
                var tsDetachedEntityDisabledTracking = PerformanceHelper.RunActionMeasurePerformance(()
=> updateDetachedEntityTotalPriceDisabledTracking());
                var tsDetachedEntityNormal = PerformanceHelper.RunActionMeasurePerformance(() =>
updateDetachedEntityTotalPriceNormal());
                results.Add(new Performance
                {
                    ListDisabledTracking = tsListDisabledTracking,
                    ListNormal = tsListNormal,
                    DetachedEntityDisabledTracking = tsDetachedEntityDisabledTracking,
                    DetachedEntityNormal = tsDetachedEntityNormal
                });
            }

            var detachedEntityDisabledTrackingAvg = results.Average(x =>
x.DetachedEntityDisabledTracking.TotalMilliseconds);
            Console.WriteLine("detachedEntityDisabledTrackingAvg: {0} ms.",
detachedEntityDisabledTrackingAvg);

            var detachedEntityNormalAvg = results.Average(x =>
x.DetachedEntityNormal.TotalMilliseconds);
            Console.WriteLine("detachedEntityNormalAvg: {0} ms.", detachedEntityNormalAvg);

            var listDisabledTrackingAvg = results.Average(x =>
x.ListDisabledTracking.TotalMilliseconds);
            Console.WriteLine("listDisabledTrackingAvg: {0} ms.", listDisabledTrackingAvg);

            var listNormalAvg = results.Average(x => x.ListNormal.TotalMilliseconds);
            Console.WriteLine("listNormalAvg: {0} ms.", listNormalAvg);
        }

        private static void updateDetachedEntityTotalPriceNormal()
        {
            using (var context = new MyContext(withTracking: true))
            {
                var detachedEntity = new Factor { Id = 1, TotalPrice = 10 };

                var attachedEntity = context.Factors.Find(detachedEntity.Id);
                if (attachedEntity != null)
                {
                    attachedEntity.TotalPrice = 100;

                    context.SaveChanges();
                }
            }
        }

        private static void updateDetachedEntityTotalPriceDisabledTracking()
        {
            using (var context = new MyContext(withTracking: false))
            {
                var detachedEntity = new Factor { Id = 2, TotalPrice = 10 };
                detachedEntity.TotalPrice = 200;

                context.CustomUpdate(detachedEntity); // custom update with change tracking disabled.
                context.SaveChanges();
            }
        }

        private static void updateListTotalPriceNormal()
        {
            using (var context = new MyContext(withTracking: true))
            {
                foreach (var item in context.Factors)
                {
                    item.TotalPrice += 10; // normal update with change tracking enabled.
                }
                context.SaveChanges();
            }
        }

        private static void updateListTotalPriceDisabledTracking()
        {
            using (var context = new MyContext(withTracking: false))
            {
                foreach (var item in context.Factors)
```

```
                {
                    item.TotalPrice += 10;
                    //نیازی به این دو سطر نیست
                    //context.ChangeTracker.DetectChanges();  // هربار باید محاسبه صورت گیرد در
غیراینصورت وضعیت تغییر نیافته گزارش می‌شود
                    //context.CustomUpdate(item); // custom update with change tracking disabled.
                }
                context.ChangeTracker.DetectChanges();  // در غیراینصورت وضعیت تغییر نیافته گزارش
می‌شود
                context.SaveChanges();
            }
        }

        private static void startDb()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            // Forces initialization of database on model changes.
            using (var context = new MyContext())
            {
                context.Database.Initialize(force: true);
            }
        }
    }

    public class PerformanceHelper
    {
        public static TimeSpan RunActionMeasurePerformance(Action action)
        {
            var stopwatch = new Stopwatch();
            stopwatch.Start();

            action();

            stopwatch.Stop();
            return stopwatch.Elapsed;
        }
    }
}
```

نتیجه این آزمایش بعد از 20 بار اجرا و اندازه گیری:

```
detachedEntityDisabledTrackingAvg: 22.32089 ms.
detachedEntityNormalAvg: 54.546815 ms.
listDisabledTrackingAvg: 413.615445 ms.
listNormalAvg: 393.194625 ms.
```

در حالت کار با یک شیء ساده، به روز رسانی حالت منقطع بسیار سریعتر است (چون یکبار رفت و برگشت کمتری داره به دیتابیس).
در حالت کار با لیستی از اشیاء دریافت شده از بانک اطلاعاتی، به روز رسانی حالت متصل به Context سریعتر است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۱۱

طول عمر یک شیء، کاری به خاموش یا روشن بودن سیستم ردیابی ندارد.
مقصود از متصل و غیرمتصلی که در اینجا عنوان شده، فعال و غیرفعال سازی مباحث Tracking در Context است و وضعیت یک شیء نسبت به Context (به علاوه خاموش کردن lazy loading و غیره). مثلا اگر خاصیت Name رو تغییر دادید، Context می‌دونه اتفاقی رخ داده یا اینکه وضعیت رو unchanged یا detached گزارش می‌ده؟

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۳۱

در کد آپدیت بالا هدف نشان دادن نحوه بروز رسانی یک شی اتچ شده بود که به اشتباه متد آپدیت رو قراردادم. (اصلاح شد)

```
T attachedEntity = set.Find(entity.Id);
var attachedEntry = dbContext.Entry(attachedEntity);
attachedEntry.CurrentValues.SetValues(entity);
```

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۷:۴۸

در حالت نه آنچنان Detached ! (دریافت یک لیست از Context ایی که ردیابی نداره)

....
در متن هم گفته شد وقتی همه چیز رو خاموش کردیم ما باید وضعیت موجودیت رو مشخص کنیم. مثلا لیستی از اشیا رو می‌سازیم کاربر یکی رو انتخاب می‌کنه تغییر می‌ده و ما در لحظه ذخیره سازی وضعیت اونو به "تغییر داده شده" تغییر می‌دیم.

```
dbContext.Entry(entity).State = EntityState.Modified;
```

در حقیقت همه اشیا CLR ساده هستند و در موقع درخواست ثبت تغییرات از ef کمک می‌گیریم.

نویسنده: arezoo
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۶:۲۸

سلام . من یه راهنمایی میخوام ممنون میشم کمکم کنین برا آپدیت مشکل دارم،کدش رو به این صورت نوشتم اما چنین اروری میده(An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot track multiple objects with the same key.)

```
public void InsertOrUpdate(Core.Models.PersonelAction personelAction {
            if (personelAction.Id == default(int))
            {
                _unitOfWork.Entry(personelAction).State = EntityState.Added;
            }
            else
            {
                 _unitOfWork.Entry(personelAction).State=EntityState.Modified;

            }
        }
```

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۷:۴۹

احتمالا رابطه‌های PersonelAction با دیگر موجودیت باعث این ارور شده، کدی که اینجا نوشتید برای پاسخ دادن خیلی ناقصه.

نویسنده: arezoo
تاریخ: ۱۳۹۲/۰۳/۳۰ ۲۲:۲۴

ممنون که جواب دادین ،موجودیت‌های من به این صورت هستن

```
(...
```

,

```
customerAction(customerId,Money,PersonelAction
```

```
(PersonelAction(Id,User,Pass,ReceivedMoeny
```

در واقع PersonelAction کارمندی هست که بعد از هر واریز مبلغ به موسسه باید مبلغ دریافتی کارمند مربوطه آپدیت شه و در

جدول customerAction هم مشخص میشه که کدوم کارمند دریافت وجه رو انجام داده

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۳۰ ۲۳:۲۱

- ممکنه نتونسته باشید unit of work رو درست مدیریت کنید و در پاس دادن اون به لایه‌های مختلف، چند وهله ازش ساخته شده. در این حالت خطای فوق رو می‌گیرد.
- ممکنه شیء در حال استفاده قبلا توسط Context بارگذاری شده و هنوز هست، مثلا در یک متد GetAll الان دوباره می‌خواهید اضافه‌اش کنید که نمی‌شود. یا مدیریت ناصحیح Context و باز نگه داشتن بیش از حد آن به ازای کل برنامه یا چندین فرم مختلف با هم که باز هم سبب این مساله می‌شود.
- یا حتی ممکنه وضعیت موجودیت EntityState.Detached باشه که باید اول Attach شود. (وضعیت اتصال موجودیت‌ها رو ابتدا چک کنید)
- اگر قراره موجودیت جدیدی اضافه بشه چرا از متد Add استفاده نکردید؟

نویسنده: arezoo
تاریخ: ۱۳۹۲/۰۳/۳۱ ۳:۷

بله حق با شما بود این موجودیت توی یکی از فرم‌ها به context اضافه شده بود
در مورد add کردن مشکلی نداشتم
میشه در مورد مدیریت unit of work یه توضیحی بدین؟ما توی هر فرم برای ذخیره‌ی تغییرات آیا یک instance از unit of work میسازیم؟

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۳/۳۱ ۸:۴۵

این خطا زمانی پیش می‌آید که personelAction ای قبلا با همین Id در DbContext شما موجود باشد و شما بخواهید وهله ای (نمونه ای) دیگر از personelAction را به DbContext جاری وارد کرده و State آن را Modifiedقرار بدید. بنابراین در else نیاز هست که چک کنید personelAction.Id قبلا در DbContext موجود است یا خیر

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۳/۳۱ ۸:۵۲

بستگی به فرم داره. اگر شما دو فرم داشته باشید که یکی Masterهست و دیگری Detail مثلا فرم سفارش و اقلام سفارش در این حالت می‌بایست از یک UnitOfWork برای دو فرم استفاده کنید چراکه دو فرم به هم وابسته هستند و نیاز هست که یکجا ذخیره شوند. نمی‌شود که اقلام سفارش ذخیره شود ولی خود سفارش ذخیره نشود

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۳۱ ۹:۹

یعنی الان یک Context به ازای کل برنامه دارید که دچار این تداخل شدید؟ معمولا در WPF و همچنین WinForms این Context به ازای هر فرم تعریف می‌شود و با بسته شدن آن تخریب.

حالا یک سؤال مهم! به نظر شما در اولین سؤالی که پرسیدید، یک شخص چطور می‌بایستی ساختار کار شما رو که بر مبنای یک Context در کل برنامه است، حدس می‌زد و عیب یابی می‌کرد؟!

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۸:۵۶

مطرح کردید:

در حالت کار با لیستی از اشیاء دریافت شده از بانک اطلاعاتی، به روز رسانی حالت متصل به Context سریعتر است.
چرا؟ چه توجیهی برای این هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۸:۵۸

چون عناصر آن متصل هستند. یعنی Context نیازی به اتصال مجدد و بررسی وضعیت تک تک عناصر آن برای تولید SQL صحیح ندارد و همه چیز از پیش محاسبه شده است (این دو مورد اتصال و محاسبه وضعیت، زمانبر است؛ برای 20 عنصر در محاسبات فوق نزدیک به 20 میلی ثانیه تفاوتش است).

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۱ ۲۳:۲۲

به نظر من عنوان صحیح نیست. اصولا EF نامتصل هست. اینی که اینجا بحث شده در حقیقت دو حالت Dispose شده یا نشده Context هست نه چیزی به عنوان Connected یا Disconnected

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۱ ۲۳:۳۳

اگر این توضیح صحیح باشه باید برای هر تعداد از عناصر هم صحیح باشه. یعنی با هر تعداد شیی در لیست. خوب حالا فرض کنید لیست ما 1 عنصر داره. این یعنی همون حالت "کار با یک شیی ساده". چرا در این حالت توضیحی که شما گفتید صادق نیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۱ ۲۳:۵۱

- سورس آزمایش به عمد ارسال شد، تا بتونید خودتون اجراش کنید و اندازه گیری کنید. این‌ها چشم بندی نبوده یا نظر شخصی نیست. یک سری اندازه گیری است.
- توضیح دادم در انتهای همان آزمایش. برای تکرار مجدد: چون یکبار رفت و برگشت کمتری داره به دیتابیس. چون تغییر State یک شیء و ورود آن به سیستم ردیابی، خیلی سریعتر است از واکشی اطلاعات از بانک اطلاعاتی. اما در مورد لیستی از اشیاء، توسط context.Factors سیستم EF دسترسی به IDها پیدا می‌کنه (در هر دو حالت متصل و منقطع). اگر سیستم ردیابی خاموش شود، برای اتصال مجدد این‌ها زمان خواهد برد (چون IDهای دریافت شده از بانک اطلاعاتی ردیابی نمی‌شوند)، اما در حالت متصل، همان بار اولی که کوئری گرفته شده، همانجا اتصال هم برقرار شده و در حین به روز رسانی اطلاعات می‌داند چه تغییراتی رخ داده و چگونه سریعا باید محاسبات رو انجام بده. اما در حالت منقطع توسط متد DetectChanges تازه شروع به اتصال و محاسبه می‌کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۲ ۰:۰

واژه‌های Attached و Detached مانند EntityState.Detached جزو فرهنگ لغات EF هستند. این معانی هم مرتبط هستند با این کلمات و نه هیچ برداشت دیگری.

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱:۱۷

احتمالا برداشت شما متصل بودن به دیتابیس است ، اینجا منظور متصل بودن یک شی به DbContext است. که این متصل بودن مزایایی همچون ردگیری تغییرات توسط DbContext را دارد.

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱۱:۴۸

درسته شاید پیدا کردن 2 واژه فارسی متفاوت برای Attached و Connected کمی سخت باشه. زبان فارسی در رشته ما کمی ناکارآمده.

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱۱:۵۷

درسته. من کد رو با دقت نخونده بودم مخصوصا متد CustomUpdate . ممنون از توضیح دوباره.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۴:۴۵

در حالت غیر متصل؛ روش پیگیری و مدیریت تغییرات Entityهای سمت کلاینت و اعمال اونها سمت سرور به چه صورته؟
منظورم اینه که فرض کنید من یک موجودیت سفارش با چندین آیتم سفارش دارم(در واقع دو موجودیت) که کاربر ممکنه وقتی یک سفارش رو ویرایش میکنه، یک آیتم رو اضافه کنه، یک آیتم رو حذف و یکی رو ویرایش کنه، در این حالت چطوری همه این تغییرات در یک UOW و توسط یک .SaveChanges() در سمت سرور اعمال میشن؟

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۸:۳۴

قسمت Find متد Update شما در حالت detached اضافی است. یعنی اگر می‌دونید که این Id در دیتابیس وجود داره نیازی به Findاش نیست. فقط State اون رو تغییر بدید کار می‌کنه .
آیا این قسمت از کد برای تشخیص objectهای تکراری در گراف objectهای ما نیست؟ ونبایستی uncomment شود؟ طبق این لینک .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۸:۵۳

صورت مساله در *اینجا* بر اساس خاموش کردن سیستم ردیابی بود (به سازنده کلاس توجه کنید). مابقی جملات هم بر این اساس نوشته شد. زمانیکه سیستم ردیابی خاموش شود، دیگر گراف objectایی از ابتدای کار وجود ندارد؛ چیزی ردیابی نمی‌شود.
بنابراین Find در حلقه‌ایی بر اساس آپدیت کردن مثلا 100 شیء منقطع جدید، مجبور میشه 100 بار به دیتابیس مراجعه کنه؛ چون EF هنوز از وجود آن‌ها مطلع نشده و کشی رو برای نگهداری اطلاعات آن‌ها تشکیل نداده.

نویسنده: محمد واحدی
تاریخ: ۱۳۹۲/۰۷/۰۹ ۱۳:۴۹

منم همین مشکل را دارم. چون می‌خوام از WCF استفاده کنم مجبورم از حالت غیر متصل استفاده کنم. فرم Header-Detail هست چند تا از آیتمها تغییر کردند یا حذف شدند. می‌خوام آبجکت هدر را وقتی میدم به سرور با بچه هاش بره چه کار باید کنم لطفا راهنمایی کنید؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۱ ۸:۳۸

Using WCF Data Services 5.6.0 with Entity Framework 6

عنوان: **نمایش خروجی SQL کدهای Entity framework 6 در کنسول دیباگ ویژوال استودیو**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۲ ۱۱:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

تا قبل از EF 6 برای تهیه لاگ SQL تولیدی توسط Entity framework نیاز بود به ابزارهای ثالث متوسل شد. برای مثال از انواع پروفایلرها استفاده کرد (^ و ^ و ^). اما در EF 6 امکان توکاری به نام Command Interception تدارک دیده شده است تا توسط آن بتوان بدون نیاز به ابزارهای جانبی، به درون سیستم EF متصل شد و دستورات تولیدی آن‌را پیش از اجرای بر روی بانک اطلاعاتی دریافت و مثلا لاگ کرد. در ادامه نمونه‌ای از این عملیات را بررسی خواهیم کرد.

تهیه کلاس SimpleInterceptor

برای اتصال به متدهای اجرای دستورات SQL در EF 6 تنها کافی است یک کلاس جدید را از کلاس پایه DbCommandInterceptor مشتق کرده و سپس متدهای کلاس پایه را override کنیم. در این متدها، فراخوانی متدهای کلاس پایه، معادل خواهند بود با اجرای واقعی دستور بر روی بانک اطلاعاتی. به این ترتیب حتی می‌توان مدت زمان انجام عملیات را نیز بدست آورد. در اینجا command.CommandText معادل است با دستور SQL در حال اجرا و همچنین نیاز است تا تمام سطوح تو در توی استثناهای احتمالی رخ داده را نیز بررسی کرد:

```
using System;
using System.Data.Common;
using System.Data.Entity.Infrastructure.Interception;
using System.Diagnostics;
using System.Text;

namespace EFCommandInterception
{
    public class SimpleInterceptor : DbCommandInterceptor
    {
        public override void ScalarExecuting(DbCommand command, DbCommandInterceptionContext<object>
interceptionContext)
        {
            var timespan = runCommand(() => base.ScalarExecuting(command, interceptionContext));
            logData(command, interceptionContext.Exception, timespan);
        }

        public override void NonQueryExecuting(DbCommand command, DbCommandInterceptionContext<int>
interceptionContext)
        {
            var timespan = runCommand(() => base.NonQueryExecuting(command, interceptionContext));
            logData(command, interceptionContext.Exception, timespan);
        }

        public override void ReaderExecuting(DbCommand command,
DbCommandInterceptionContext<DbDataReader> interceptionContext)
        {
            var timespan = runCommand(() => base.ReaderExecuting(command, interceptionContext));
            logData(command, interceptionContext.Exception, timespan);
        }

        private static Stopwatch runCommand(Action command)
        {
            var timespan = Stopwatch.StartNew();
            command();
            timespan.Stop();
            return timespan;
        }

        private static void logData(DbCommand command, Exception exception, Stopwatch timespan)
        {
            if (exception != null)
            {
                Trace.TraceError(formatException(exception, "Error executing command: {0}",
command.CommandText));
            }
            else
            {
                Trace.TraceInformation(string.Concat("Elapsed time: ", timespan.Elapsed, " Command: ",
command.CommandText));
            }
```

```
        }

        private static string formatException(Exception exception, string fmt, params object[] vars)
        {
            var sb = new StringBuilder();
            sb.Append(string.Format(fmt, vars));
            sb.Append(" Exception: ");
            sb.Append(exception.ToString());
            while (exception.InnerException != null)
            {
                sb.Append(" Inner exception: ");
                sb.Append(exception.InnerException.ToString());
                exception = exception.InnerException;
            }
            return sb.ToString();
        }
    }
}
```

نحوه استفاده از کلاس SimpleInterceptor

کلاس فوق را کافی است تنها یکبار در آغاز برنامه (مثلا در متد Application_Start برنامه‌های وب) به EF 6 معرفی کرد:

```
DbInterception.Add(new SimpleInterceptor());
```

اکنون اگر برنامه را اجرا کنیم، خروجی SQL و زمان‌های اجرای عملیات را در پنجره دیباگ VS.NET می‌توان مشاهده کرد:

نظرات خوانندگان

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۸/۱۲ ۱۷:۴۳

سلام، چه کتابی برای EF6 پیشنهاد میکنین؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۲ ۱۸:۸

از قسمت اول سری EF Code first شروع کنید. مباحث پایه‌ای همان است. فقط یک سری افزونه بیشتر شده.

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۰:۳۶

البته با استفاده از خصوصیت Log نیز می‌توانیم دستورات TSQL را در خروجی لاگ کنیم :

```
public MyContext():base("MyConnectionString")
{
            Database.Log = sql => Debug.Write(sql);
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۷ ۱۲:۲۱

یک نکته‌ی تکمیلی

در EF 6.1 به بعد، کل روش ارائه شده در اینجا را می‌توانید به نحو ذیل در فایل کانفیگ برنامه‌های وب یا ویندوزی، برای ذخیره در فایل، فعال کنید (بدون نیاز به کدنویسی اضافه‌تری):

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
    <parameters>
      <parameter value="C:\Temp\LogOutput.txt"/>
      <parameter value="true" type="System.Boolean"/>
    </parameters>
  </interceptor>
</interceptors>
```

عنوان: **بالا بردن سرعت DbContext هنگام ثبت داده های زیاد**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۳ ۲۲:۵
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, EF Code First, DetectChanged

تشریح مسئله : شاید شما هم هنگام ثبت، ویرایش و حتی حذف داده‌های زیاد در Code First متوجه کاهش چشمگیر کارایی پروژه خود شده باشید.(برای مثال ثبت 5000 داده یا بیشتر به صورت هم زمان).برای رفع مشکل بالا چه باید کرد؟

نکته : آشنایی اولیه با مفاهیم EF CodeFirst برای درک بهتر مفاهیم الزامی است.

EntityFramework Code First هنگام کار با Poco Entities برای اینکه مشخص شود که چه داده هایی باید به دیتابیس ارسال شود مکانیزمی به نام Detect Changed معرفی کرده است که وظیفه آن بررسی تفاوت‌های بین مقادیر خواص CurrentValue و OriginalValue هر Entity است که باعث افت چشمگیر سرعت هنگام اجرای عملیات CRUD می‌شود.
هنگامی که از یک Entity کوئری گرفته می‌شود یا از دستور Attach برای یک Entity استفاده می‌کنیم مقادیر مورد نظر در حافظه ذخیره می‌شوند. استفاده از هر کدام دستورات زیر DbContext را مجبور به فراخوانی الگوریتم Automatic Detect Changed می‌کند.

DbSet.Find
DbSet.Local
DbSet.Remove
DbSet.Add
DbSet.Attach
DbContext.SaveChanges
DbContext.GetValidationErrors
DbContext.Entry
DbChangeTracker.Entries

البته Code First امکانی را فراهم کرده است که هنگام پیاده سازی عملیات CRUD اگر تعداد داده‌های شرکت کننده زیاد است برای رفع مشکل کاهش سرعت بهتر است این رفتار را غیر فعال کنیم . به صورت زیر:

```
using (var context = new BookContext())
{
    try
    {
        context.Configuration.AutoDetectChangesEnabled = false;

        foreach (var book in aLotOfBooks)
        {
            context.Books.Add(book);
        }
    }
    finally
    {
        context.Configuration.AutoDetectChangesEnabled = true;
    }
```

در پایان هم وضعیت را به حالت قبل بر می‌گردانیم.
در مورد کاهش مصرف حافظه EF CodeFirst هنگام واکشی داده‌های زیاد هم می‌تونید از این مقاله استفاده کنید.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۳ ۲۳:۹

ممنون.
- نکته دیگری که به شدت روی سرعت bulk insert حین کار با ORMها (فرقی نمی‌کنه؛ تمامشون) تاثیر داره، استراتژی انتخاب نوع primary key است. زمانیکه کلید اصلی از نوع auto generated توسط بانک اطلاعاتی باشه، ORM بعد از insert سعی می‌کنه این Id رو به مصرف کننده برگردونه (چون برنامه نقشی در تعیین اون نداره). یعنی عملا با یک insert و بلافاصله با یک select مواجه خواهیم بود. البته این مورد هم باید اضافه بشه که ORMها برای فرآیندها و تراکنش‌هایی کوتاه طراحی شدن و این مساله در حالت متداول کار با ORMها اهمیتی نداره و اصلا به چشم نمیاد.
همین مساله سرعت insert رو «فقط» در حالت فراخوانی با تعداد بالا در یک حلقه برای مثال به شدت پایین میاره و اگر مثلا کلید اصلی توسط خود برنامه مدیریت بشه (مثلا از نوع Guid تولید شده در برنامه باشه)، سرعت bulk insert به شدت بالا میره چون به select بعد از insert نیازی نخواهد بود.
- در حالت bulk insert اگر شخص مطمئن هست که اطلاعات ارسالی توسط او اعتبارسنجی شدن، بهتره تنظیم context.Configuration.ValidateOnSaveEnabled = false رو هم انجام بده. این مورد اعتبارسنجی در حین ذخیره سازی رو غیرفعال می‌کنه.
- همچنین شخصی در اینجا در مورد تعداد بار فراخوانی متد SaveChanges در یک حلقه، تحقیقی رو انجام داده که جالب است.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۳ ۲۳:۳۸

ممنون از توضیحات تکمیلی.
تحقیق مورد نظر رو هم بررسی کردم. در نوع خودش جالب بود.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۲:۲

سلام ... خیلی ممنون بابت مطلب مفیدتون ...
در چه مواردی نباید از این روش استفاده کرد؟!

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۲:۴۹

در کل هر زمان که قصد انجام Bulk Insert رو ندارید این رفتار را غیر فعال نکنید.(به صورت پیش فرض فعال است)
البته بهتره که هر زمان در عملیات Bulk Insert تعداد رکوردهای مورد نظر خیلی زیاد بود به ازای یک تعداد مشخص از Entity ها(برای مثال 1000) یک بار DbContext رو SaveChanged کرده و اونو Dispose کنید و دوباره یک Instance جدید از DbContext بسازید و ادامه کار(دلیل دوباره ساختن DbContext هم اینه که DbContext ، بعد از دستور SaveChanged دیتای مورد نظر رو در دیتابیس ذخیره می‌کنهه ولی فقط State هر Entity رو به Unchaged تغییر میده و خود Entity رو Detach نمی‌کنه که این خود باعث افزایش ObjectGraph موجود در DbContext می‌شود و در نتیجه کاهش کارایی).
در ضمن می‌تونید با فراخوانی دستور DetectChanged مستقیما DbContext رو مجبور به بررسی وضعیت خواص CurrentValue و OriginalValue هر Entity بکنید.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۰۵ ۹:۲۶

کتابخانه extended ef هم به نظر در این مورد جالب است.

عنوان: **استفاده از EF در اپلیکیشن های N-Tier : قسمت اول**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۰۷ ۲:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Tips, ASP.NET Web API, N-Layer Architecture, Entity Framework 6

تمام اپلیکیشن‌ها را نمی‌توان در یک پروسس بسته بندی کرد، بدین معنا که تمام اپلیکیشن روی یک سرور فیزیکی قرار گیرد. در عصر حاظر معماری بسیاری از اپلیکیشن‌ها چند لایه است و هر لایه روی سرور مجزایی توزیع می‌شود. بعنوان مثال یک معماری کلاسیک شامل سه لایه نمایش (presentation)، اپلیکیشن (application) و داده (data) است. لایه بندی منطقی (logical layering) یک اپلیکیشن می‌تواند در یک App Domain واحد پیاده سازی شده و روی یک کامپیوتر میزبانی شود. در این صورت لازم نیست نگران مباحثی مانند پراکسی ها، مرتب سازی (serialization)، پروتوکل‌های شبکه و غیره باشیم. اما اپلیکیشن‌های بزرگی که چندین کلاینت دارند و در مراکز داده میزبانی می‌شوند باید تمام این مسائل را در نظر بگیرند. خوشبختانه پیاده سازی چنین اپلیکیشن هایی با استفاده از Entity Framework و دیگر تکنولوژی‌های مایکروسافت مانند WCF, Web API ساده‌تر شده است. منظور از n-Tier معماری اپلیکیشن هایی است که لایه‌های نمایش، منطق تجاری و دسترسی داده هر کدام روی سرور مجزایی میزبانی می‌شوند. این تفکیک فیزیکی لایه‌ها به بسط پذیری، مدیریت و نگهداری اپلیکیشن‌ها در دراز مدت کمک می‌کند، اما معمولا تاثیری منفی روی کارایی کلی سیستم دارد. چرا که برای انجام عملیات مختلف باید از محدوده ماشین‌های فیریکی عبور کنیم.

معماری N-Tier چالش‌های بخصوصی را برای قابلیت‌های change-tracking در EF اضافه می‌کند. در ابتدا داده‌ها توسط یک آبجکت EF Context بارگذاری می‌شوند اما این آبجکت پس از ارسال داده‌ها به کلاینت از بین می‌رود. تغییراتی که در سمت کلاینت روی داده‌ها اعمال می‌شوند ردیابی (track) نخواهند شد. هنگام بروز رسانی، آبجکت Context جدیدی برای پردازش اطلاعات ارسالی باید ایجاد شود. مسلما آبجکت جدید هیچ چیز درباره Context پیشین یا مقادیر اصلی موجودیت‌ها نمی‌داند.

در نسخه‌های قبلی Entity Framework توسعه دهندگان با استفاده از قالب ویژه ای بنام Self-Tracking Entities می‌توانستند تغییرات موجودیت‌ها را ردیابی کنند. این قابلیت در نسخه EF 6 از رده خارج شده است و گرچه هنوز توسط ObjectContext پشتیبانی می‌شود، آبجکت DbContext از آن پشتیبانی نمی‌کند.

در این سری از مقالات روی عملیات پایه CRUD تمرکز می‌کنیم که در اکثر اپلیکیشن‌های n-Tier استفاده می‌شوند. همچنین خواهیم دید چگونه می‌توان تغییرات موجودیت‌ها را ردیابی کرد. مباحثی مانند همزمانی (concurrency) و مرتب سازی (serialization) نیز بررسی خواهند شد. در قسمت یک این سری مقالات، به بروز رسانی موجودیت‌های منفصل (disconnected) توسط سرویس‌های Web API نگاهی خواهیم داشت.

بروز رسانی موجودیت‌های منفصل با Web API

سناریویی را فرض کنید که در آن برای انجام عملیات CRUD از یک سرویس Web API استفاده می‌شود. همچنین مدیریت داده‌ها با مدل Code-First پیاده سازی شده است. در مثال جاری یک کلاینت Console Application خواهیم داشت که یک سرویس Web API را فراخوانی می‌کند. توجه داشته باشید که هر اپلیکیشن در Solution مجزایی قرار دارد. تفکیک پروژه‌ها برای شبیه سازی یک محیط n-Tier انجام شده است.

فرض کنید مدلی مانند تصویر زیر داریم.

همانطور که می‌بینید مدل جاری، سفارشات یک اپلیکیشن فرضی را معرفی می‌کند. می‌خواهیم مدل و کد دسترسی به داده‌ها را در یک سرویس Web API پیاده سازی کنیم، تا هر کلاینتی که از HTTP استفاده می‌کند بتواند عملیات CRUD را انجام دهد. برای ساختن سرویس مورد نظر مراحل زیر را دنبال کنید.

در ویژوال استودیو پروژه جدیدی از نوع ASP.NET Web Application بسازید و قالب پروژه را Web API انتخاب کنید. نام پروژه را به Recipe1.Service تغییر دهید.

کنترلر جدیدی از نوع WebApi Controller با نام OrderController به پروژه اضافه کنید.

کلاس جدیدی با نام Order در پوشه مدل‌ها ایجاد کنید و کد زیر را به آن اضافه نمایید.

```
public class Order
{
    public int OrderId { get; set; }
    public string Product { get; set; }
    public int Quantity { get; set; }
    public string Status { get; set; }
    public byte[] TimeStamp { get; set; }
}
```

با استفاده از NuGet Package Manager کتابخانه Entity Framework 6 را به پروژه اضافه کنید.

حال کلاسی با نام Recipe1Context ایجاد کنید و کد زیر را به آن اضافه نمایید.

```
public class Recipe1Context : DbContext
{
    public Recipe1Context() : base("Recipe1ConnectionString") { }

    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Order>().ToTable("Orders");
        // Following configuration enables timestamp to be concurrency token
        modelBuilder.Entity<Order>().Property(x => x.TimeStamp)
            .IsConcurrencyToken()
            .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Computed);
    }
}
```

فایل Web.config پروژه را باز کنید و رشته اتصال زیر را به قسمت ConnectionStrings اضافه نمایید.

```
<connectionStrings>
  <add name="Recipe1ConnectionString"
    connectionString="Data Source=.;
    Initial Catalog=EFRecipes;
    Integrated Security=True;
    MultipleActiveResultSets=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

فایل Global.asax را باز کنید و کد زیر را به آن اضافه نمایید. این کد بررسی Entity Framework Compatibility را غیرفعال می‌کند.

```
protected void Application_Start()
{
    // Disable Entity Framework Model Compatibilty
    Database.SetInitializer<Recipe1Context>(null);
    ...
}
```

در آخر کد کنترلر Order را با لیست زیر جایگزین کنید.

```
public class OrderController : ApiController
{
    // GET api/order
    public IEnumerable<Order> Get()
    {
        using (var context = new Recipe1Context())
        {
            return context.Orders.ToList();
        }
    }

    // GET api/order/5
    public Order Get(int id)
    {
        using (var context = new Recipe1Context())
        {
            return context.Orders.FirstOrDefault(x => x.OrderId == id);
        }
    }

    // POST api/order
    public HttpResponseMessage Post(Order order)
    {
        // Cleanup data from previous requests
        Cleanup();

        using (var context = new Recipe1Context())
        {
            context.Orders.Add(order);
            context.SaveChanges();
            // create HttpResponseMessage to wrap result, assigning Http Status code of 201,
            // which informs client that resource created successfully
            var response = Request.CreateResponse(HttpStatusCode.Created, order);
            // add location of newly-created resource to response header
            response.Headers.Location = new Uri(Url.Link("DefaultApi",
                new { id = order.OrderId }));
            return response;
        }
    }

    // PUT api/order/5
    public HttpResponseMessage Put(Order order)
    {
        using (var context = new Recipe1Context())
        {
            context.Entry(order).State = EntityState.Modified;
            context.SaveChanges();
            // return Http Status code of 200, informing client that resouce updated successfully
            return Request.CreateResponse(HttpStatusCode.OK, order);
        }
    }

    // DELETE api/order/5
```

```
    public HttpResponseMessage Delete(int id)
    {
        using (var context = new Recipe1Context())
        {
            var order = context.Orders.FirstOrDefault(x => x.OrderId == id);
            context.Orders.Remove(order);
            context.SaveChanges();
            // Return Http Status code of 200, informing client that resouce removed successfully
            return Request.CreateResponse(HttpStatusCode.OK);
        }
    }

    private void Cleanup()
    {
        using (var context = new Recipe1Context())
        {
            context.Database.ExecuteSqlCommand("delete from [orders]");
        }
    }
}
```

قابل ذکر است که هنگام استفاده از Entity Framework در MVC یا Web API، بکارگیری قابلیت Scaffolding بسیار مفید است. این فریم ورک‌های ASP.NET می‌توانند کنترلرهایی کاملا اجرایی برایتان تولید کنند که صرفه جویی چشمگیری در زمان و کار شما خواهد بود.

در قدم بعدی اپلیکیشن کلاینت را می‌سازیم که از سرویس Web API استفاده می‌کند.

در ویژوال استودیو پروژه جدیدی از نوع Console Application بسازید و نام آن را به Recipe1.Client تغییر دهید.
کلاس موجودیت Order را به پروژه اضافه کنید. همان کلاسی که در سرویس Web API ساختیم.

نکته: قسمت هایی از اپلیکیشن که باید در لایه‌های مختلف مورد استفاده قرار گیرند - مانند کلاس‌های موجودیت‌ها - بهتر است در لایه مجزایی قرار داده شده و به اشتراک گذاشته شوند. مثلا می‌توانید پروژه ای از نوع Class Library بسازید و تمام موجودیت‌ها را در آن تعریف کنید. سپس لایه‌های مختلف این پروژه را ارجاع خواهند کرد.

فایل program.cs را باز کنید و کد زیر را به آن اضافه نمایید.

```
private HttpClient _client;
private Order _order;

private static void Main()
{
    Task t = Run();
    t.Wait();

    Console.WriteLine("\nPress <enter> to continue...");
    Console.ReadLine();
}

private static async Task Run()
{
    // create instance of the program class
    var program = new Program();
    program.ServiceSetup();
    program.CreateOrder();
    // do not proceed until order is added
    await program.PostOrderAsync();
    program.ChangeOrder();
    // do not proceed until order is changed
    await program.PutOrderAsync();
    // do not proceed until order is removed
    await program.RemoveOrderAsync();
}

private void ServiceSetup()
{
    // map URL for Web API cal
    _client = new HttpClient { BaseAddress = new Uri("http://localhost:3237/") };
    // add Accept Header to request Web API content
```

```
    // negotiation to return resource in JSON format
    _client.DefaultRequestHeaders.Accept.
        Add(new MediaTypeWithQualityHeaderValue("application/json"));
}

private void CreateOrder()
{
    // Create new order
    _order = new Order { Product = "Camping Tent", Quantity = 3, Status = "Received" };
}

private async Task PostOrderAsync()
{
    // leverage Web API client side API to call service
    var response = await _client.PostAsJsonAsync("api/order", _order);
    Uri newOrderUri;

    if (response.IsSuccessStatusCode)
    {
        // Capture Uri of new resource
        newOrderUri = response.Headers.Location;
        // capture newly-created order returned from service,
        // which will now include the database-generated Id value
        _order = await response.Content.ReadAsAsync<Order>();
        Console.WriteLine("Successfully created order. Here is URL to new resource: {0}",
newOrderUri);
    }
    else
        Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);
}

private void ChangeOrder()
{
    // update order
    _order.Quantity = 10;
}

private async Task PutOrderAsync()
{
    // construct call to generate HttpPut verb and dispatch
    // to corresponding Put method in the Web API Service
    var response = await _client.PutAsJsonAsync("api/order", _order);

    if (response.IsSuccessStatusCode)
    {
        // capture updated order returned from service, which will include new quanity
        _order = await response.Content.ReadAsAsync<Order>();
        Console.WriteLine("Successfully updated order: {0}", response.StatusCode);
    }
    else
        Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);
}

private async Task RemoveOrderAsync()
{
    // remove order
    var uri = "api/order/" + _order.OrderId;
    var response = await _client.DeleteAsync(uri);

    if (response.IsSuccessStatusCode)
        Console.WriteLine("Sucessfully deleted order: {0}", response.StatusCode);
    else
        Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);
}
```

اگر اپلیکیشن کلاینت را اجرا کنید باید با خروجی زیر مواجه شوید:

```
Successfully created order: http://localhost:3237/api/order/1054
```

```
Successfully updated order: OK
```

```
Sucessfully deleted order: OK
```

شرح مثال جاری

با اجرای اپلیکیشن Web API شروع کنید. این اپلیکیشن یک کنترلر Web API دارد که پس از اجرا شما را به صفحه خانه هدایت می‌کند. در این مرحله اپلیکیشن در حال اجرا است و سرویس‌های ما قابل دسترسی هستند.

حال اپلیکیشن کنسول را باز کنید. روی خط اول کد program.cs یک breakpoint تعریف کرده و اپلیکیشن را اجرا کنید. ابتدا آدرس سرویس Web API را پیکربندی کرده و خاصیت Accept Header را مقدار دهی می‌کنیم. با این کار از سرویس مورد نظر درخواست می‌کنیم که داده‌ها را با فرمت JSON بازگرداند. سپس یک آبجکت Order می‌سازیم و با فراخوانی متد PostAsJsonAsync آن را به سرویس ارسال می‌کنیم. این متد روی آبجکت HttpClient تعریف شده است. اگر به اکشن متد Post در کنترلر Order یک breakpoint اضافه کنید، خواهید دید که این متد سفارش جدید را بعنوان یک پارامتر دریافت می‌کند و آن را به لیست موجودیت‌ها در Context جاری اضافه می‌نماید. این عمل باعث می‌شود که آبجکت جدید بعنوان Added علامت گذاری شود، در این مرحله Context جاری شروع به ردیابی تغییرات می‌کند. در آخر با فراخوانی متد SaveChanges داده‌ها را ذخیره می‌کنیم. در قدم بعدی کد وضعیت 201 (Created) و آدرس منبع جدید را در یک آبجکت HttpResponseMessage قرار می‌دهیم و به کلاینت ارسال می‌کنیم. هنگام استفاده از Web API باید اطمینان حاصل کنیم که کلاینت‌ها درخواست‌های ایجاد رکورد جدید را بصورت POST ارسال می‌کنند. درخواست‌های HTTP Post بصورت خودکار به اکشن متد متناظر نگاشت می‌شوند.

در مرحله بعد عملیات بعدی را اجرا می‌کنیم، تعداد سفارش را تغییر می‌دهیم و موجودیت جاری را با فراخوانی متد PutAsJsonAsync به سرویس Web API ارسال می‌کنیم. اگر به اکشن متد Put در کنترلر سرویس یک breakpoint اضافه کنید، خواهید دید که آبجکت سفارش بصورت یک پارامتر دریافت می‌شود. سپس با فراخوانی متد Entry و پاس دادن موجودیت جاری بعنوان رفرنس، خاصیت State را به Modified تغییر می‌دهیم، که این کار موجودیت را به Context جاری می‌چسباند. حال فراخوانی متد SaveChanges یک اسکریپت بروز رسانی تولید خواهد کرد. در مثال جاری تمام فیلدهای آبجکت Order را بروز رسانی می‌کنیم. در شماره‌های بعدی این سری از مقالات، خواهیم دید چگونه می‌توان تنها فیلدهایی را بروز رسانی کرد که تغییر کرده اند. در آخر عملیات را با بازگرداندن کد وضعیت 200 (OK) به اتمام می‌رسانیم.

در مرحله بعد، عملیات نهایی را اجرا می‌کنیم که موجودیت Order را از منبع داده حذف می‌کند. برای اینکار شناسه (Id) رکورد مورد نظر را به آدرس سرویس اضافه می‌کنیم و متد DeleteAsync را فراخوانی می‌کنیم. در سرویس Web API رکورد مورد نظر را از دیتابیس دریافت کرده و متد Remove را روی Context جاری فراخوانی می‌کنیم. این کار موجودیت مورد نظر را بعنوان Deleted علامت گذاری می‌کند. فراخوانی متد SaveChanges یک اسکریپت Delete تولید خواهد کرد که نهایتا منجر به حذف شدن رکورد می‌شود.

در یک اپلیکیشن واقعی بهتر است کد دسترسی داده‌ها از سرویس Web API تفکیک شود و در لایه مجزایی قرار گیرد.

عنوان: **نحوه کاهش مصرف حافظه EF Code first حین گزارشگیری از اطلاعات**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۶ ۱۹:۶
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

تمام ORMهای خوب، دارای سطح اول کش هستند. از این سطح جهت نگهداری اطلاعات تغییرات صورت گرفته روی اشیاء و سپس اعمال نهایی آن‌ها در پایان یک تراکنش استفاده می‌شود. بدیهی است جمع آوری این اطلاعات اندکی بر روی سرعت انجام کار و همچنین بر روی میزان مصرف حافظه برنامه تاثیرگذار است. به علاوه یک سری از اعمال مانند گزارشگیری نیازی به این سطح اول کش ندارند. اطلاعات مورد استفاده در آن‌ها مانند نمایش لیستی از اطلاعات در یک گرید، حالت فقط خواندنی دارد. در EF Code first برای یک چنین مواردی استفاده از متد الحاقی AsNoTracking تدارک دیده شده است که سبب خاموش شدن سطح اول کش می‌شود. در ادامه در طی یک مثال، اثر این متد را بر روی سرعت و میزان مصرف حافظه برنامه بررسی خواهیم کرد.

کدهای کامل این مثال را در ذیل ملاحظه می‌کنید:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Diagnostics;
using System.Linq;

namespace EFGeneral
{
    public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public class MyContext : DbContext
    {
        public DbSet<User> Users { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            for (int i = 0; i < 21000; i++)
            {
                context.Users.Add(new User { Name = "name " + i });
                if (i % 1000 == 0)
                    context.SaveChanges();
            }
            base.Seed(context);
        }
    }

    public class PerformanceHelper
    {
        public static string RunActionMeasurePerformance(Action action)
        {
            GC.Collect();
            long initMemUsage = Process.GetCurrentProcess().WorkingSet64;

            var stopwatch = new Stopwatch();
            stopwatch.Start();

            action();

            stopwatch.Stop();

            var currentMemUsage = Process.GetCurrentProcess().WorkingSet64;
            var memUsage = currentMemUsage - initMemUsage;
            if (memUsage < 0) memUsage = 0;
```

```
            return string.Format("Elapsed time: {0}, Memory Usage: {1:N2} KB", stopwatch.Elapsed,
memUsage / 1024);
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            StartDb();

            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine("\nRun {0}", i + 1);

                var memUsage = PerformanceHelper.RunActionMeasurePerformance(() => LoadWithTracking());
                Console.WriteLine("LoadWithTracking:\n{0}", memUsage);

                memUsage = PerformanceHelper.RunActionMeasurePerformance(() => LoadWithoutTracking());
                Console.WriteLine("LoadWithoutTracking:\n{0}", memUsage);
            }
        }

        private static void StartDb()
        {
            using (var ctx = new MyContext())
            {
                var user = ctx.Users.Find(1);
                if (user != null)
                {
                    // keep the object in memory
                }
            }
        }

        private static void LoadWithTracking()
        {
            using (var ctx = new MyContext())
            {
                var list = ctx.Users.ToList();
                if (list.Any())
                {
                    // keep the list in memory
                }
            }
        }

        private static void LoadWithoutTracking()
        {
            using (var ctx = new MyContext())
            {
                var list = ctx.Users.AsNoTracking().ToList();
                if (list.Any())
                {
                    // keep the list in memory
                }
            }
        }
    }
}
```

توضیحات:

مدل برنامه یک کلاس ساده کاربر است به همراه id و نام او.

سپس این کلاس توسط Context برنامه در معرض دید EF Code first قرار می‌گیرد.

در کلاس Configuration تعدادی رکورد را در ابتدای کار برنامه در بانک اطلاعاتی ثبت خواهیم کرد. قصد داریم میزان مصرف حافظه بارگذاری این اطلاعات را بررسی کنیم.

کلاس PerformanceHelper معرفی شده، دو کار اندازه گیری میزان مصرف حافظه برنامه در طی اجرای یک فرمان خاص و همچنین مدت زمان سپری شدن آن‌را اندازه‌گیری می‌کند.

در کلاس Test فوق چندین متد به شرح زیر وجود دارند:

متد StartDb سبب می‌شود تا تنظیمات ابتدایی برنامه به بانک اطلاعاتی اعمال شوند. تا زمانیکه کوئری خاصی به بانک اطلاعاتی

ارسال نگردد، EF Code first بانک اطلاعاتی را آغاز نخواهد کرد.
در متد LoadWithTracking اطلاعات تمام رکوردها به صورت متداولی بارگذاری شده است.
در متد LoadWithoutTracking نحوه استفاده از متد الحاقی AsNoTracking را مشاهده می‌کنید. در این متد سطح اول کش به این ترتیب خاموش می‌شود.
و متد RunTests، این متدها را در سه بار متوالی اجرا کرده و نتیجه عملیات را نمایش خواهد داد.

برای نمونه این نتیجه در اینجا حاصل شده است:

```
Run 1
LoadWithTracking:
Elapsed time: 00:00:00.9332636, Memory Usage: 8,528.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.3119418, Memory Usage: 256.00 KB

Run 2
LoadWithTracking:
Elapsed time: 00:00:00.3611471, Memory Usage: 4,100.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.1590219, Memory Usage: 256.00 KB

Run 3
LoadWithTracking:
Elapsed time: 00:00:00.3750008, Memory Usage: 4,332.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.1593993, Memory Usage: 216.00 KB
```

همانطور که ملاحظه کنید، بین این دو حالت، تفاوت بسیار قابل ملاحظه است؛ چه از لحاظ مصرف حافظه و چه از لحاظ سرعت.

نتیجه گیری:
اگر قصد ندارید بر روی اطلاعات دریافتی از بانک اطلاعاتی تغییرات خاصی را انجام دهید و فقط قرار است از آن‌ها به صورت فقط خواندنی گزارشگیری شود، بهتر است سطح اول کش را به کمک متد الحاقی AsNoTracking خاموش کنید.

نظرات خوانندگان

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۰/۲۳ ۱۷:۸

سلام.
وقتی از متد Where و ... برای فیلتر کردن استفاده کنیم دیگه قادر به انجام این کار (خاموش کردن سطح اول کش) نیستیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۳ ۱۷:۱۵

```
// Query for all users without tracking them
var users = context.Users.AsNoTracking();

// Query for some users without tracking them
var someUsers = context.Users
                        .Where(u => u.Name.EndsWith("st"))
                        .AsNoTracking()
                        .ToList();
//Or ...
var items = context.Users.AsNoTracking().Where(...);
```

نویسنده: احمد ولی پور
تاریخ: ۱۳۹۱/۱۲/۰۹ ۱۹:۴۴

با سلام
اگر کش سطح اول رو غیر فعال کنیم و توی صفحه عمل Update یا Delete انجام بدیم چه اتفاقی رخ میده؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۹ ۲۱:۲۸

با استفاده از متد AsNoTracking و یا غیرفعال سازی کلی این فرآیند توسط تنظیم
context.Configuration.AutoDetectChangesEnabled = false ، شیء از context جدا شده در نظر گرفته می‌شود. بنابراین context از تغییرات شما بی‌خبر بوده و ... اتفاق خاصی رخ نخواهد داد.

نویسنده: پدرام جباری
تاریخ: ۱۳۹۱/۱۲/۱۰ ۲۳:۵۲

اگر نیاز دارید مدل رو از یک Context جدا کنید (کش کردن اون رو غیر فعال کنید) باید توجه داشته باشید که غبر فعال کردن AutoDetectChangesEnabled کافی نیست باید متد AsNoTracking رو هم استفاده کنید ، مخصوصا برای زمانی که لازم داشته باشید در یک شی دیگه از Context اون مدل رو Attach کنید، اگر هر دو رو غیر فعال نکنید Attach کردن مدل (بسته به پیچیدگی مدل) زمانی تا 5 ثانیه یا حتی بیشتر میبره.

غیر فعال کردن کلی AutoDetectChangesEnabled بیشتر زمانی که می‌خواهید رکورد به دیتابیس اضافه کنید بسیار مورد نیاز هست، سرعت رو به مقدار قابل توجهی افزایش میده (البته برای تعداد رکورد بالا تاثیر خودش رو نشون میده)
برای آپدیت و حذف رکورد ، اگر از وجود رکورد اطمینان دارید (مخصوصا برای ویرایش مدل) بهتر هست مدل رو به Context ای که دارید Attach کنید که خوب بهتر از Select زدن از دیتابیس هست

نویسنده: ahmad.valipour
تاریخ: ۱۳۹۲/۰۲/۱۴ ۲۰:۴۸

با سلام
متد بالا رو برای DataBase First هم میشه استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۴ ۲۱:۵۳

برای حالت استفاده مستقیم از ObjectContext:

```
var context = new NorthwindDataContext();
context.tblCities.MergeOption = MergeOption.NoTracking;
```

واقعیت این است که یک EF بیشتر وجود خارجی ندارد. سورس EF هم در دسترس است :

```
public virtual IInternalQuery<TElement> AsNoTracking()
{
   return (IInternalQuery<TElement>) new InternalQuery<TElement>(this._internalContext, (ObjectQuery)
DbHelpers.CreateNoTrackingQuery((ObjectQuery) this._objectQuery));
}

public static IQueryable CreateNoTrackingQuery(ObjectQuery query)
{
  IQueryable queryable = (IQueryable) query;
  ObjectQuery objectQuery = (ObjectQuery) queryable.Provider.CreateQuery(queryable.Expression);
  objectQuery.MergeOption = MergeOption.NoTracking; // اینجا کار خاموش سازی ردیابی انجام شده
  return (IQueryable) objectQuery;
}
```

همانطور که مشاهده می‌کنید، متد الحاقی AsNoTracking در پشت صحنه همان کار تنظیم MergeOption = MergeOption.NoTracking رو انجام می‌ده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۹ ۱۲:۴۶

اهمیت استفاده از AsNoTracking زمانیکه نیازی به آن نیست:

Fetch performance of various .NET ORM / Data-access frameworks

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۹/۲۳ ۹:۵

آیا ممکن است AsNoTracking برای همه‌ی Queryها فعال شود ؟ یعنی کلا Change Tracker , سطح اول Cache خاموش شود.
در مثال پست جاری تابع LoadWithoutTracking را با کد زیر جایگزین کردم :

```
private static void LoadWithoutTracking()
        {
            using (var ctx = new MyContext())
            {
                ctx.Configuration.AutoDetectChangesEnabled = false;
                var list = ctx.Users.ToList();
                if (list.Any())
                {
                    // keep the list in memory
                }
            }
        }
```

با توجه نتیجه‌ی حاصل شده به نظر مصرف حافظه بهبود نیافته :

```
Run 1
LoadWithTracking:
Elapsed time: 00:00:00.2917882, Memory Usage: 22,040.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.2280811, Memory Usage: 18,408.00 KB

Run 2
LoadWithTracking:
Elapsed time: 00:00:00.2239667, Memory Usage: 16,720.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.2254920, Memory Usage: 18,160.00 KB

Run 3
LoadWithTracking:
Elapsed time: 00:00:00.2216326, Memory Usage: 16,720.00 KB
LoadWithoutTracking:
Elapsed time: 00:00:00.2252134, Memory Usage: 18,200.00 KB
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۰:۴۵

با توجه به سورس EF ، در متد CreateNoTrackingQuery کار MergeOption.NoTracking به ازای یک ObjectSet (و نه حتی DbSet) انجام می‌شود و الزاما معادل نیست با AutoDetectChangesEnabled = false تنها.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۱۱/۲۸ ۱۷:۲۸

سلام؛ من اینجا و اینجا و اینجا را ... مطالعه کردم. اما متوجه نشدم در زمان گزارش گیری با هر ابزاری (Stmulsoft, FastReport, ..) اطلاعات باید به صورت یک BusinesObject و یا هر عنوان دیگه ای به ابزار مورد نظر در قالب یک IEnumeable ارسال شود. در حالی که اگر ما IQueable را با .ToList() به یک IEnumerable تبدیل شود، ممکن است در برگیرنده کل اطلاعات باشد. در این موارد راهی برای کاهش حافظه و سربار کم وجود دارد؟ متد ToList را نمی‌توان به صورت Lazy پیاده سازی کرد یعنی اگر ابزار گزارش ساز فرضا صفحه 1 را نمایش دهد اطلاعات تا صفحه یک از بانک واکشی بشود. اگر گزارش ما 200 صفحه باشد در حالت عادی کل اطلاعات در سرور لود شده و برنامه‌های گزارش ساز صرفا پس از تهیه گزارش اطلاعات را به صورت صفحه بندی نمایش می‌دهند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۸ ۱۷:۴۲

مورد مدنظر شما اصطلاحا paging نام دارد و در گزارش گیری‌های خصوصا برنامه‌های تحت وب که گرید نهایی را برنامه نویس با کدنویسی و ارائه منبع داده مناسبی طراحی و پیاده سازی می‌کند، بسیار مرسوم است (یک Take و Skip است در سمت کوئری LINQ نوشته شده). مثلا:

« واکشی اطلاعات به صورت chunk chunk (تکه تکه) و نمایش در ListView »

این قابلیت اگر در نرم افزارهای گزارشگیری یاد شده، پیاده سازی شده‌است (مانند مثال یاد شده *MaximumRows* و *StartRowIndex* را هربار در اختیار برنامه نویس قرار می‌دهند)، آنگاه قابل استفاده و پیاده سازی خواهد بود. در غیراینصورت، کار خاصی را نمی‌توان انجام داد و باید مطابق نیاز تجاری آن‌ها رفتار کرد.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۱۱/۳۰ ۱۰:۱۳

سلام؛ ممنون از پاسختون. من تمام برنامه‌های گزارش سازی را که میشناختم (Telerik, Fastreport, ReportViewer, DevExpress) همه را بررسی کردم . اما هیچ کدام چنین پراپرتی و یا مشابه اون و نداشتند. یعنی هرکسی می‌خواد گزارش بگیرد یک دفعه کل اطلاعات و از بانک می‌خونه! هرچقدر هم با فیلترهای مختلف گزارش و با فیلترهای مختلف مثلا تاریخ یا شماره رکورد محدود کنیم اما در کل کاربر در هر لحظه یک صفحه را که بیشتر نمی‌تواند ببیند. مباحث مذکور به خوبی در انواع گرید و کنترل‌های مختلف پیاده سازی شده و شرکت‌های مختلف راه حل‌های مختلفی همانند مواردی که شما ذکر کردید ارائه کرده اند اما برای گزارش خیر !

عنوان: EF Code First #13
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۷ ۱۰:۰۶:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

استفاده مستقیم از عبارات SQL در EF Code first

طراحی اکثر ORMهای موجود به نحوی است که برنامه نهایی شما را مستقل از بانک اطلاعاتی کنند و این پروایدر نهایی است که معادل‌های صحیح بسیاری از توابع توکار بانک اطلاعاتی مورد استفاده را در اختیار EF قرار می‌دهد. برای مثال در یک بانک اطلاعاتی تابعی به نام substr تعریف شده، در بانک اطلاعاتی دیگری همین تابع substring نام دارد. اگر برنامه را به کمک کوئری‌های LINQ تهیه کنیم، نهایتا پروایدر نهایی مخصوص بانک اطلاعاتی مورد استفاده است که این معادل‌ها را در اختیار EF قرار می‌دهد و برنامه بدون مشکل کار خواهد کرد. اما یک سری از موارد شاید معادلی در سایر بانک‌های اطلاعاتی نداشته باشند؛ برای مثال رویه‌های ذخیره شده یا توابع تعریف شده توسط کاربر. امکان استفاده از یک چنین توانایی‌هایی نیز با اجرای مستقیم عبارات SQL در EF Code first پیش بینی شده و بدیهی است در این حالت برنامه به یک بانک اطلاعاتی خاص گره خواهد خورد؛ همچنین مزیت استفاده از کوئری‌های Strongly typed تحت نظر کامپایلر را نیز از دست خواهیم داد. به علاوه باید به یک سری مسایل امنیتی نیز دقت داشت که در ادامه بررسی خواهند شد.

کلاس‌های مدل مثال جاری

در مثال جاری قصد داریم نحوه استفاده از رویه‌های ذخیره شده و توابع تعریف شده توسط کاربر مخصوص SQL Server را بررسی کنیم. در اینجا کلاس‌های پزشک و بیماران او، کلاس‌های مدل برنامه را تشکیل می‌دهند:

```
using System.Collections.Generic;

namespace EF_Sample08.DomainClasses
{
    public class Doctor
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<Patient> Patients { set; get; }
    }
}
```

```
namespace EF_Sample08.DomainClasses
{
    public class Patient
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual Doctor Doctor { set; get; }
    }
}
```

کلاس **Context** برنامه به نحو زیر تعریف شده:

```
using System.Data.Entity;
using EF_Sample08.DomainClasses;
```

```
namespace EF_Sample08.DataLayer.Context
{
    public class Sample08Context : DbContext
    {
        public DbSet<Doctor> Doctors { set; get; }
        public DbSet<Patient> Patients { set; get; }
    }
}
```

و اینبار کلاس **DbMigrationsConfiguration** تعریف شده اندکی با مثال‌های قبلی متفاوت است:

```
using System.Data.Entity.Migrations;
using EF_Sample08.DomainClasses;
using System.Collections.Generic;

namespace EF_Sample08.DataLayer.Context
{
    public class Configuration : DbMigrationsConfiguration<Sample08Context>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(Sample08Context context)
        {
            addData(context);
            addSP(context);
            addFn(context);
            base.Seed(context);
        }

        private static void addData(Sample08Context context)
        {
            var patient1 = new Patient { Name = "p1" };
            var patient2 = new Patient { Name = "p2" };
            var doctor1 = new Doctor { Name = "doc1", Patients = new List<Patient> { patient1, patient2
} };
            context.Doctors.Add(doctor1);
        }

        private static void addFn(Sample08Context context)
        {
            context.Database.ExecuteSqlCommand(
                @"IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[FindDoctorPatientsCount]')
                  AND type in (N'FN', N'IF', N'TF', N'FS', N'FT'))
                     DROP FUNCTION [dbo].[FindDoctorPatientsCount]");
            context.Database.ExecuteSqlCommand(
                @"CREATE FUNCTION FindDoctorPatientsCount(@Doctor_Id INT)
                    RETURNS INT
                    BEGIN
                     RETURN
                     (
                         SELECT COUNT(*)
                         FROM   Patients
                         WHERE  Doctor_Id = @Doctor_Id
                     );
                    END");
        }

        private static void addSP(Sample08Context context)
        {
            context.Database.ExecuteSqlCommand(
                @"IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[FindDoctorsStartWith]')
                  AND type in (N'P', N'PC'))
                       DROP PROCEDURE [dbo].[FindDoctorsStartWith]
                 ");
            context.Database.ExecuteSqlCommand(
                @"CREATE PROCEDURE FindDoctorsStartWith(@name NVARCHAR(400))
                  AS
                 SELECT *
                 FROM   Doctors
```

```
                WHERE  [Name] LIKE @name + '%'");
        }
    }
}
```

در اینجا از متد Seed علاوه بر مقدار دهی اولیه جداول، برای تعریف یک رویه ذخیره شده به نام FindDoctorsStartWith و یک تابع سفارشی به نام FindDoctorPatientsCount نیز استفاده شده است. متد context.Database.ExecuteSqlCommand مستقیما یک عبارت SQL را بر روی بانک اطلاعاتی اجرا می‌کند.

در ادامه کدهای کامل برنامه نهایی را ملاحظه می‌کنید:

```
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Objects.SqlClient;
using System.Data.SqlClient;
using System.Linq;
using EF_Sample08.DataLayer.Context;
using EF_Sample08.DomainClasses;

namespace EF_Sample08
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample08Context,
Configuration>());

            using (var db = new Sample08Context())
            {
                runSp(db);
                runFn(db);
                usingSqlFunctions(db);
            }
        }

        private static void usingSqlFunctions(Sample08Context db)
        {
            var doctorsWithNumericNameList = db.Doctors.Where(x => SqlFunctions.IsNumeric(x.Name) ==
1).ToList();
            if (doctorsWithNumericNameList.Any())
            {
                //do something
            }
        }

        private static void runFn(Sample08Context db)
        {
            var doctorIdParameter = new SqlParameter
            {
                ParameterName = "@doctor_id",
                Value = 1,
                SqlDbType = SqlDbType.Int
            };
            var patientsCount = db.Database.SqlQuery<int>("select
dbo.FindDoctorPatientsCount(@doctor_id)", doctorIdParameter).FirstOrDefault();
            Console.WriteLine(patientsCount);
        }

        private static void runSp(Sample08Context db)
        {
            var nameParameter = new SqlParameter
            {
                ParameterName = "@name",
                Value = "doc",
                Direction = ParameterDirection.Input,
                SqlDbType = SqlDbType.NVarChar
            };
            var doctors = db.Database.SqlQuery<Doctor>("exec FindDoctorsStartWith @name",
nameParameter).ToList();
            if (doctors.Any())
            {
```

```
                foreach (var item in doctors)
                {
                    Console.WriteLine(item.Name);
                }
            }
        }
    }
}
```

توضیحات

همانطور که ملاحظه می‌کنید، برای اجرای مستقیم یک عبارت SQL صرفنظر از اینکه یک رویه ذخیره شده است یا یک تابع و یا یک کوئری معمولی، باید از متد db.Database.SqlQuery استفاده کرد. خروجی این متد از نوع IEnumerable است و این توانایی را دارد که رکوردهای بازگشت داده شده از بانک اطلاعاتی را به خواص یک کلاس به صورت خودکار نگاشت کند.
پارامتر اول متد db.Database.SqlQuery، عبارت SQL مورد نظر است. پارامتر دوم آن باید توسط وهله‌هایی از کلاس SqlParameter مقدار دهی شود. به کمک SqlParameter نام پارامتر مورد استفاده، مقدار و نوع آن مشخص می‌گردد. همچنین Direction آن نیز برای استفاده از رویه‌های ذخیره شده ویژه‌ای که دارای پارامتری از نوع out هستند درنظر گرفته شده است.

چند نکته

- در متد runSp فوق، متد الحاقی ToList را حذف کرده و برنامه را اجرا کنید. بلافاصله پیغام خطای «The SqlParameter is already contained by another SqlParameterCollection.» ظاهر خواهد شد. علت هم این است که با بکارگیری متد ToList، تمام عملیات یکبار انجام شده و نتیجه بازگشت داده می‌شود اما اگر به صورت مستقیم از خروجی IEnumerable آن استفاده کنیم، در حلقه foreach تعریف شده، ممکن است این فراخوانی چندبار انجام شود. به همین جهت ذکر متد ToList در اینجا ضروری است.

- عنوان شد که در اینجا باید به مسایل امنیتی دقت داشت. بدیهی است امکان نوشتن یک چنین کوئری‌هایی نیز وجود دارد:

```
db.Database.SqlQuery<Doctor>("exec FindDoctorsStartWith "+ txtName.Text, nameParameter).ToList()
```

در این حالت به ظاهر مشغول به استفاده از رویه‌های ذخیره شده‌ای هستیم که عنوان می‌شود در برابر حملات تزریق SQL در امان هستند، اما چون در کدهای ما به نحو ناصحیحی با جمع زدن رشته‌ها مقدار دهی شده است، برنامه و بانک اطلاعاتی دیگر در امان نخواهند بود. بنابراین در این حالت استفاده از پارامترها را نباید فراموش کرد.
زمانیکه از کوئری‌های LINQ استفاده می‌شود تمام این مسایل توسط EF مدیریت خواهد شد. اما اگر قصد دارید مستقیما عبارات SQL را فراخوانی کنید، تامین امنیت برنامه به عهده خودتان خواهد بود.

- در متد usingSqlFunctions از SqlFunctions.IsNumeric استفاده شده است. این مورد مختص به SQL Server است و امکان استفاده از توابع توکار ویژه SQL Server را در کوئری‌های LINQ برنامه فراهم می‌سازد. برای مثال متدالحاقی از پیش تعریف شده‌ای به نام IsNumeric به صورت مستقیم در دسترس نیست، اما به کمک کلاس SqlFunctions این تابع و بسیاری از توابع دیگر توکار SQL Server قابل استفاده خواهند بود.
اگر علاقمند هستید که لیست این توابع را مشاهده کنید، در ویژوال استودیو بر روی SqlFunctions کلیک راست کرده و گزینه Go to definition را انتخاب کنید.

نظرات خوانندگان

نویسنده: Hassan
تاریخ: ۱۳۹۱/۰۲/۲۹ ۱۶:۴۸:۱۷

سلام
در صورتی که در query از join و group استفاده کنیم، یعنی در خروجی ResultSet فیلدهای چند جدول و همچنین یکسری فیلدهای جدید که توسط توابع تولید می شوند را داشته باشیم، نحوه نگاشت به کلاس ها چگونه خواهد بود؟ EF خودش آن را مدیریت می کند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۹ ۱۷:۱۱:۲۸

یک کلاس جدید تعریف کنید که شامل فیلدهای متناظر با select شما باشد. کار نگاشت نهایی به این‌ها خودکار خواهد بود.

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۳۰ ۰۰:۲۱:۰۹

زبان eSql را شما کاربردی برایش می بینید؟
شاید مایکروسافت می خواست یک MSIL برای تکنولوژی Data Access داشته باشد!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۳۰ ۰۸:۱۷:۲۷

ESQL هم قابل استفاده است: (^)

نویسنده: آرش عظیمی
تاریخ: ۱۳۹۲/۰۵/۱۲ ۲:۲۲

در این روش چطوری می‌شه دستور Where رو به صورت یک رشته اجرا نمود
به طور مثال این دستور

```
DBEntities MyDB = new DBEntities();
        var Query1 = from P in MyDB.Per
        where P.IDRANK == 2
        select P;
```

تبدیل بشه به یه چنین دستوری

```
string strquery = "where P.IDRANK == 2";
    DBEntities MyDB = new DBEntities();
    var Query1 = from P in MyDB.Per
         strquery
    select P;
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۲ ۹:۴۲

باید از Dynamic LINQ استفاده کنید.

نویسنده: reza110
تاریخ: ۱۳۹۳/۰۸/۲۹ ۹:۱۶

با سلام

یک SP داریم که محتویات یک جدول را از دیتابیس موجود در سرور 1 به دیتابیس موجود در سرور 2 انتقال می‌دهد. بین دو سرور از لینک سرور استفاده کرده ایم و وقتی SP را از محیط دیتابیس که در سرور 2 قرار دارد مستقیما اجرا می کنیم بدرستی کار می‌کند ولی وقتی از دستور db.Database.SqlQuery برای اجرای SP استفاده می‌کنیم برنامه ساعتها در حالت اجرا می‌ماند و کاری نمی‌کند و خطایی هم نمی‌دهد و داده ای هم منتقل نمی‌کند حتی timeout را هم افزایش داده ام. آیا این نوع SPها در CodeFirst قابل اجرا هستند؟

با تشکر فراوان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۲۹ ۹:۲۴

EF برای اجرای بسیاری از اعمال خودش، دستورات را داخل تراکنش‌ها اجرا می‌کند:

« سرورهای متصل شده‌ی SQL Server و مبحث تراکنش‌ها »

عنوان: EF Code First #15
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۳۰ ۱۲:۲۰:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

EF Code first و بانک‌های اطلاعاتی متفاوت

در آخرین قسمت از سری EF Code first بد نیست نحوه استفاده از بانک‌های اطلاعاتی دیگری را بجز SQL Server نیز بررسی کنیم. در اینجا کلاس‌های مدل و کدهای مورد استفاده نیز همانند قسمت 14 است و تنها به ذکر تفاوت‌ها و نکات مرتبط اکتفاء خواهد شد.

حالت کلی پشتیبانی از بانک‌های اطلاعاتی مختلف توسط EF Code first

EF Code first با کلیه پروایدرهای تهیه شده برای ADO.NET 3.5 که پشتیبانی از EF را لحاظ کرده باشند، به خوبی کار می‌کند.
پروایدرهای مخصوص ADO.NET 4.0، تنها سه گزینه DeleteDatabase/CreateDatabase/DatabaseExists را نسبت به نگارش قبلی بیشتر دارند و EF Code first ویژگی‌های بیشتری را طلب نمی‌کند.
بنابراین اگر حین استفاده از پروایدر ADO.NET مخصوص بانک اطلاعاتی خاصی با پیغام «CreateDatabase is not supported by the provider» مواجه شدید، به این معنا است که این پروایدر برای دات نت 4 به روز نشده است. اما به این معنا نیست که با EF Code first کار نمی‌کند. فقط باید یک دیتابیس خالی از پیش تهیه شده را به برنامه معرفی کنید تا مباحث Database Migrations به خوبی کار کنند؛ یا اینکه کلا می‌توانید Database Migrations را خاموش کرده (متد Database.SetInitializer را با پارامتر نال فراخوانی کنید) و فیلدها و جداول را دستی ایجاد کنید.

استفاده از EF Code first با SQLite

برای استفاده از SQLite در دات نت ابتدا نیاز به پروایدر ADO.NET آن است: « مکان دریافت درایورهای جدید SQLite مخصوص دات نت »
ضمن اینکه به نکته « استفاده از اسمبلی‌های دات نت 2 در یک پروژه دات نت 4 » نیز باید دقت داشت.
و یکی از بهترین management studio هایی که برای آن تهیه شده: « SQLite Manager »
پس از دریافت پروایدر آن، ارجاعی را به اسمبلی System.Data.SQLite.dll به برنامه اضافه کنید.
سپس فایل کانفیگ برنامه را به نحو زیر تغییر دهید:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=4.3.1.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0"/>
  </startup>

  <connectionStrings>
    <clear/>
    <add name="Sample09Context"
         connectionString="Data Source=CodeFirst.db"
         providerName="System.Data.SQLite"/>
  </connectionStrings>
</configuration>
```

همانطور که ملاحظه می‌کنید، تفاوت آن با قبل، تغییر connectionString و providerName است.

اکنون اگر همان برنامه قسمت قبل را اجرا کنیم به خطای زیر برخواهیم خورد:
«The given key was not present in the dictionary»
در این مورد هم توضیح داده شد. سه گزینه DeleteDatabase/CreateDatabase/DatabaseExists در پروایدر جاری SQLite برای دات نت وجود ندارد. به همین جهت نیاز است فایل «CodeFirst.db» ذکر شده در کانکشن استرینگ را ابتدا دستی درست کرد.
برای مثال از افزونه SQLite Manager استفاده کنید. ابتدا یک بانک اطلاعاتی خالی را درست کرده و سپس دستورات زیر را بر روی بانک اطلاعاتی اجرا کنید تا دو جدول خالی را ایجاد کند (در برگه Execute sql افزونه SQLite Manager):

```
CREATE TABLE [Payees](
 [Id] [integer] PRIMARY KEY AUTOINCREMENT NOT NULL,
 [Name] [text] NULL,
 [CreatedOn] [datetime] NOT NULL,
 [CreatedBy] [text] NULL,
 [ModifiedOn] [datetime] NOT NULL,
 [ModifiedBy] [text] NULL
);

CREATE TABLE [Bills](
 [Id] [integer] PRIMARY KEY AUTOINCREMENT NOT NULL,
 [Amount] [float](18, 2) NOT NULL,
 [Description] [text] NULL,
 [CreatedOn] [datetime] NOT NULL,
 [CreatedBy] [text] NULL,
 [ModifiedOn] [datetime] NOT NULL,
 [ModifiedBy] [text] NULL,
 [Payee_Id] [integer] NULL
);
```

سپس سطر زیر را نیز به ابتدای برنامه اضافه کنید:

```
Database.SetInitializer<Sample09Context>(null);
```

به این ترتیب database migrations خاموش می‌شود و اکنون برنامه بدون مشکل کار خواهد کرد.
فقط باید به یک سری نکات مانند نوع داده‌ها در بانک‌های اطلاعاتی مختلف دقت داشت. برای مثال integer در اینجا از نوع Int64 است؛ بنابراین در برنامه نیز باید به همین ترتیب تعریف شود تا نگاشت‌ها به درستی انجام شوند.

در کل تنها مشکل پروایدر فعلی SQLite عدم پشتیبانی از مباحث database migrations است. این مورد را خاموش کرده و تغییرات ساختار بانک اطلاعاتی را به صورت دستی به بانک اطلاعاتی اعمال کنید. بدون مشکل کار خواهد کرد.

البته اگر به دنبال پروایدری تجاری با پشتیبانی از آخرین نگارش EF Code first هستید، گزینه زیر نیز مهیا است:
http://devart.com/dotconnect/sqlite/
برای مثال اگر علاقمند به استفاده از حالت تشکیل بانک اطلاعاتی SQLite در حافظه هستید (با رشته اتصالی ویژه Data Source=:memory:;Version=3;New=True;)، فعلا تنها گزینه مهیا استفاده از پروایدر تجاری فوق است؛ زیرا مبحث Database Migrations را به خوبی پشتیبانی می‌کند.

استفاده از EF Code first با SQL Server CE

قبلا در مورد « استفاده از SQL-CE به کمک NHibernate » مطلبی را در این سایت مطالعه کرده‌اید. سه مورد اول آن با EF Code first یکی است و تفاوتی نمی‌کند (یک سری بحث عمومی مشترک است). البته با یک تفاوت؛ در اینجا EF Code first قادر است یک بانک اطلاعاتی خالی SQL Server CE را به صورت خودکار ایجاد کند و نیازی نیست تا آن‌را دستی ایجاد کرد. مباحث database migrations و به روز رسانی خودکار ساختار بانک اطلاعاتی نیز در اینجا پشتیبانی می‌شود.

برای استفاده از آن ابتدا ارجاعی را به اسمبلی System.Data.SqlServerCe.dll قرار گرفته در مسیر Program Files\Microsoft SQL Server Compact Edition\v4.0\Desktop اضافه کنید.
سپس رشته اتصالی به بانک اطلاعاتی و providerName را به نحو زیر تغییر دهید:

```
<connectionStrings>
    <clear/>
    <add name="Sample09Context"
         connectionString="Data Source=mydb.sdf;Password=1234;Encrypt Database=True"
         providerName="System.Data.SqlServerCE.4.0"/>
</connectionStrings>
```

بدون نیاز به هیچگونه تغییری در کدهای برنامه، همین مقدار تغییر در تنظیمات ابتدایی برنامه برای کار با SQL Server CE کافی است.
ضمنا مشکلی هم با فیلد Identity در آخرین نگارش EF Code first وجود ندارد؛ برخلاف حالت database first آن که پیشتر این اجازه را نمی‌داد و خطای «Server-generated keys and server-generated values are not supported by SQL Server Compact» را ظاهر می‌کرد.

استفاده از EF Code first با MySQL

برای استفاده از EF Code first با MySQL (نگارش 5 به بعد البته) ابتدا نیاز است پروایدر مخصوص ADO.NET آن‌را دریافت کرد: (^)
که از EF نیز پشتیبانی می‌کند. پس از نصب آن، ارجاعی را به اسمبلی MySql.Data.dll قرار گرفته در مسیر Program Files\MySQL\MySQL Connector Net 6.5.4\Assemblies\v4.0 به پروژه اضافه نمائید.
سپس رشته اتصالی و providerName را به نحو زیر تغییر دهید:

```
<connectionStrings>
    <clear/>
    <add name="Sample09Context"
         connectionString="Datasource=localhost; Database=testdb2; Uid=root; Pwd=123;"
         providerName="MySql.Data.MySqlClient"/>
</connectionStrings>

<system.data>
    <DbProviderFactories>
      <remove invariant="MySql.Data.MySqlClient"/>
      <add name="MySQL Data Provider"
           invariant="MySql.Data.MySqlClient"
           description=".Net Framework Data Provider for MySQL"
           type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.5.4.0,
Culture=neutral, PublicKeyToken=c5687fc88969c44d" />
    </DbProviderFactories>
</system.data>
```

همانطور که مشاهده می‌کنید در اینجا شماره نگارش دقیق پروایدر مورد استفاده نیز ذکر شده است. برای مثال اگر چندین پروایدر روی سیستم نصب است، با مقدار دهی DbProviderFactories می‌توان از نگارش مخصوصی استفاده کرد.

با این تغییرات پس از اجرای برنامه قسمت قبل، به خطای زیر برخواهیم خورد:
The given key was not present in the dictionary

توضیحات این مورد با قسمت SQLite یکی است؛ به عبارتی نیاز است بانک اطلاعاتی testdb را دستی درست کرد. همچنین

جداول و فیلدها را نیز باید دستی ایجاد کرد و database migrations را نیز باید خاموش کرد (پارامتر Database.SetInitializer را به نال مقدار دهی کنید).
برای این منظور یک دیتابیس خالی را ایجاد کرده و سپس دو جدول زیر را به آن اضافه کنید:

```
CREATE TABLE IF NOT EXISTS `bills` (
  `Id` int(11) NOT NULL AUTO_INCREMENT,
  `Amount` float DEFAULT NULL,
  `Description` varchar(400) CHARACTER SET utf8 COLLATE utf8_persian_ci NOT NULL,
  `CreatedOn` datetime NOT NULL,
  `CreatedBy` varchar(400) CHARACTER SET utf8 COLLATE utf8_persian_ci NOT NULL,
  `ModifiedOn` datetime NOT NULL,
  `ModifiedBy` varchar(400) CHARACTER SET utf8 COLLATE utf8_persian_ci NOT NULL,
  `Payee_Id` int(11) NOT NULL,
  PRIMARY KEY (`Id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_persian_ci AUTO_INCREMENT=1 ;

CREATE TABLE IF NOT EXISTS `payees` (
  `Id` int(11) NOT NULL AUTO_INCREMENT,
  `Name` varchar(400) CHARACTER SET utf8 COLLATE utf8_persian_ci NOT NULL,
  `CreatedOn` datetime NOT NULL,
  `CreatedBy` varchar(400) CHARACTER SET utf8 COLLATE utf8_persian_ci NOT NULL,
  `ModifiedOn` datetime NOT NULL,
  `ModifiedBy` varchar(400) CHARACTER SET utf8 COLLATE utf8_persian_ci NOT NULL,
  PRIMARY KEY (`Id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_persian_ci AUTO_INCREMENT=1 ;
```

پس از این تغییرات، برنامه بدون مشکل اجرا خواهد شد (ایجاد بانک اطلاعاتی خالی به همراه ایجاد ساختار جداول و خاموش کردن database migrations که توسط این پروایدر پشتیبانی نمی‌شود).

به علاوه پروایدر تجاری دیگری هم در سایت devart.com برای MySQL و EF Code first مهیا است که مباحث database migrations را به خوبی مدیریت می‌کند.

مشکل!

اگر به همین نحو برنامه را اجرا کنیم، فیلدهای یونیکد فارسی ثبت شده در MySQL با «??????? ?? ????» مقدار دهی خواهند شد و تنظیم CHARACTER SET utf8 COLLATE utf8_persian_ci نیز کافی نبوده است (این مورد با SQLite یا نگارش‌های مختلف SQL Server بدون مشکل کار می‌کند و نیاز به تنظیم اضافه‌تری ندارد):

```
ALTER TABLE `bills` DEFAULT CHARACTER SET utf8 COLLATE utf8_persian_ci
```

برای رفع این مشکل توصیه شده است که CharSet=UTF8 را به رشته اتصالی به بانک اطلاعاتی اضافه کنیم. اما در این حالت خطای زیر ظاهر می‌شود:

The provider did not return a ProviderManifestToken string

این مورد فقط به اشتباه بودن تعاریف رشته اتصالی بر می‌گردد؛ یا عدم پشتیبانی از تنظیم اضافه‌ای که در رشته اتصالی ذکر شده است.

مقدار صحیح آن دقیقا مساوی CHARSET=utf8 است (با همین نگارش و رعایت کوچکی و بزرگی حروف؛ مهم!):

```
<connectionStrings>
    <clear/>
    <add name="Sample09Context"
         connectionString="Datasource=localhost; Database=testdb; Uid=root; Pwd=123;CHARSET=utf8"
         providerName="MySql.Data.MySqlClient"/>
</connectionStrings>
```

به این ترتیب، مشکل ثبت عبارات یونیکد فارسی برطرف می‌شود (البته جدول هم بهتر است به DEFAULT CHARACTER SET utf8 COLLATE utf8_persian_ci تغییر پیدا کند؛ مطابق دستور Alter ایی که در بالا ذکر شد).

نظرات خوانندگان

نویسنده: MehdiPayervand
تاریخ: ۱۳۹۱/۰۲/۳۰ ۱۲:۴۱:۵۱

اول اینکه دستتون درد نکنه، واقعا سری هایی که شما برای اشتراک دانشتون توی بلاگ میذاریم بروز و کارآمد هستند، امیدوارم بتونیم به بهترین نحو از این دانسته ها استفاده کنیم.
و دانشمون اونقدری بشه که به اشتراک گذاشت (;

نویسنده: NTC
تاریخ: ۱۳۹۱/۰۲/۳۰ ۲۰:۳۶:۱۴

مطالب کامل و جامعی رو نوشتید.
از زحمات بی دریغتون کمال تشکر را دارم.
اما همچنان منتظر مطالب دیگر هم هستیم.

نویسنده: فرشید ابراهیمی
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۸:۳۷

اگر امکان دارد نحوه اتصال به اراکل را نیز توضیح دهید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۸:۴۷

تعداد بانک‌های اطلاعاتی مهیا خیلی زیاد است. اگر این قسمت را مرور کرده باشید، حدود کار دستتان آمده است و این مهم است. چه خطاهایی را ممکن است دریافت کنید. راه حل چیست. پروایدر مخصوص نیاز دارید که احتمالا در سایت مربوطه قابل دسترسی است و از این دست موارد.
برای نمونه پروایدر رسمی Oracle ODP.Net با EF Code first کار می‌کند و یا یک نمونه دیگر در اینجا (^)

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۶/۰۴ ۱۹:۹

میخواستم بدونم EntityFramework.SqlServerCompact که در Nuget هستش چه تفاوتی با Entity Framework معمولی داره؟
من دیتابیسم Sql Ce هستش. یعنی استفاده از این Package بهتر هستش؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۰۴ ۱۹:۲۰

سورس کد EF رو که دریافت کنید، یک پوشه به نام EntityFramework.SqlServerCompact داخل آن هست. به عبارتی آخرین نگارش EF به همراه پروایدر توکار SQL CE هم هست (و بوده). ضمنا این پروایدر به تنهایی کار نمی‌کند و نیاز خواهید داشت که پروایدر ADO.NET مربوط به SQL CE را هم به پروژه اضافه کنید .

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۸/۱۸ ۱۳:۵۹

لطفا پروایدر MySQL مخصوص ADO.NET را که در وب سایت رسمی MySQL قابل دریافت نیست،در صورتی که براتون مقدور بود (حجم و ...) در اینجا بارگذاری کنید
با تشکر

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۸/۱۸ ۱۴:۱۶

- لینک اصلی: http://cdn.mysql.com/Downloads/Connector-Net/mysql-connector-net-6.7.4.msi
- لینک کمکی در اینجا
- جزئیات نحوه استفاده از آن توسط یکی از اعضای تیم EF در اینجا: Entity Framework on MySQL

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۸/۱۸ ۱۴:۲۷

بسیار عالی / با سپاس

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۰/۰۴ ۲۳:۴۷

با سلام؛ در دیتابیس Sql Server برای اینکه اطلاعات در صورت Valid بودن از Log فایل حذف بشه می‌یابیم و دیتابیس رو به صورت Simple قرار می‌دیم تا اندازه فایل Log افزایش پیدا نکنه و دستورش هم به این صورته

```
ALTER DataBase DBName SET RECOVERY SIMPLE
```

حالا این رو در ORMها و بطبع در EF Code First چطور میشه پیاده سازی کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۵ ۰:۱۱

- امکان « استفاده مستقیم از عبارات SQL در EF Code first » وجود دارد؛ برای تمام حالاتی که EF آن‌ها را به صورت توکار پشتیبانی نمی‌کند.
- محل قرار دادن تنظیمات مقدماتی از این دست، در متد Seed مهاجرت هست. یک مثال و مثالی دیگر در مورد کار با Seed.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۱۱/۲۹ ۱۷:۹

نمونه‌ای از پیاده سازی اتصال به اوراکل 11g در Entity Framework 6 بوسیله پروایدر تجاری شرکت devart :

ابتدا نسخه آزمایشی dotconnect for oracle 8.2 professional را از این آدرس دریافت و آن را نصب می‌کنیم.

نصب آخرین نسخه Entity Framework از طریق پاور شل نیوگت.
افزودن Devart.Data و Devart.Data.Oracle و Devart.Data.Oracle.Entity به Solution.
حذف تگ defaultConnectionFactory در entityFramework.
افزودن تگ زیر در قسمت providers همانند کد زیر:

```
<provider invariantName="Devart.Data.Oracle"
type="Devart.Data.Oracle.Entity.OracleEntityProviderServices, Devart.Data.Oracle.Entity,
Version=8.2.100.6, Culture=neutral, PublicKeyToken=09af7300eec23701" />
```

تکمیلی: اصول کلی دسترسی به اوراکل به شرح بالاست، ولی نکته مهم مقداردهی به خصیصه Version=X.X.X.X با توجه به نسخه اسمبلی Devart.Data.Oracle.Entity می‌باشد.

dotConnect for Oracle

dotConnect for Oracle

عنوان: نوشتن پرس و جو در Entity Framework با استفاده از LINQ To Entity قسمت اول
نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۴/۰۷ ۱۳:۲۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ to Entity

موجودیت‌های زیر را در نظر بگیرید:

```
public class Customer
{
    public Customer()
    {
        Orders = new ObservableCollection<Order>();
    }
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Family { get; set; }

    public string FullName
    {
        get
        {
            return Name + " " + Family;
        }
    }

    public virtual IList<Order> Orders { get; set; }
}
```

```
public class Product
{
    public Product()
    {
    }

    public Guid Id { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }
}
public class OrderDetail
{
    public Guid Id { get; set; }
    public Guid ProductId { get; set; }
    public int Count { get; set; }
    public Guid OrderId { get; set; }
    public int Price { get; set; }

    public virtual Order Order { get; set; }
    public virtual Product Product { get; set; }

    public string ProductName
    {
        get
        {
            return Product != null ? Product.Name : string.Empty;
        }
    }
}
```

```
public class Order
{
    public Order()
    {
        OrderDetail = new ObservableCollection<OrderDetail>();
    }
    public Guid Id { get; set; }
    public DateTime Date { get; set; }

    public Guid CustomerId { get; set; }
    public virtual Customer Customer { get; set; }
    public virtual IList<OrderDetail> OrderDetail { get; set; }

    public string CustomerFullName
    {
        get
```

```
        {
            return Customer == null ? string.Empty : Customer.FullName;
        }
    }

    public int TotalPrice
    {
        get
        {
            if (OrderDetail == null)
                return 0;

            return
                OrderDetail.Where(orderdetail => orderdetail.Product != null)
                .Sum(orderdetail => orderdetail.Price*orderdetail.Count);
        }
    }
}
```

و نگاشت موجودیت ها:

```
public class CustomerConfiguration : EntityTypeConfiguration<Customer>
{
    public CustomerConfiguration()
    {
        HasKey(c => c.Id);
        Property(c => c.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
public class ProductConfiguration : EntityTypeConfiguration<Product>
{
    public ProductConfiguration()
    {
        HasKey(p => p.Id);
        Property(p => p.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
public class OrderDetailConfiguration : EntityTypeConfiguration<OrderDetail>
{
    public OrderDetailConfiguration()
    {
        HasKey(od => od.Id);
        Property(od => od.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
public class OrderConfiguration: EntityTypeConfiguration<Order>
{
    public OrderConfiguration()
    {
        HasKey(o => o.Id);
        Property(o => o.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    }
}
```

و برای معرفی موجودیت‌ها به Entity Framwork کلاس StoreDbContext را به صورت زیر تعریف می‌کنیم:

```
public class StoreDbContext : DbContext
{
    public StoreDbContext()
        : base("name=StoreDb")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Configurations.Add(new CustomerConfiguration());
        modelBuilder.Configurations.Add(new OrderConfiguration());
        modelBuilder.Configurations.Add(new OrderDetailConfiguration());
        modelBuilder.Configurations.Add(new ProductConfiguration());
    }
```

```
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<OrderDetail> OrderDetails { get; set; }
}
```

جهت مقدار دهی اولیه به database تستی یک DataBaseInitializer به صورت زیر تعریف می‌کنیم:

```
public class MyTestDb : DropCreateDatabaseAlways<StoreDbContext>
{
    protected override void Seed(StoreDbContext context)
    {
        var customer1 = new Customer { Name = "Vahid", Family = "Nasiri" };
        var customer2 = new Customer { Name = "Mohsen", Family = "Jamshidi" };
        var customer3 = new Customer { Name = "Mohsen", Family = "Akbari" };

        var product1 = new Product {Name = "CPU", Price = 350000};
        var product2 = new Product {Name = "Monitor", Price = 500000};
        var product3 = new Product {Name = "Keyboard", Price = 30000};
        var product4 = new Product {Name = "Mouse", Price = 20000};
        var product5 = new Product {Name = "Power", Price = 70000};
        var product6 = new Product {Name = "Hard", Price = 250000};

        var order1 = new Order
        {
            Customer = customer1, Date = new DateTime(2013, 1, 1),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product1, Count = 1, Price = product1.Price},
                new OrderDetail {Product = product2, Count = 1, Price = product2.Price},
                new OrderDetail {Product = product3, Count = 1, Price = product3.Price},
            }
        };

        var order2 = new Order
        {
            Customer = customer1,
            Date = new DateTime(2013, 1, 5),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product1, Count = 2, Price = product1.Price},
                new OrderDetail {Product = product3, Count = 4, Price = product3.Price},
            }
        };

        var order3 = new Order
        {
            Customer = customer1,
            Date = new DateTime(2013, 1, 9),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product1, Count = 4, Price = product1.Price},
                new OrderDetail {Product = product3, Count = 5, Price = product3.Price},
                new OrderDetail {Product = product5, Count = 6, Price = product5.Price},
            }
        };

        var order4 = new Order
        {
            Customer = customer2,
            Date = new DateTime(2013, 1, 9),
            OrderDetail = new List<OrderDetail>
            {
                new OrderDetail {Product = product4, Count = 1, Price = product4.Price},
                new OrderDetail {Product = product3, Count = 1, Price = product3.Price},
                new OrderDetail {Product = product6, Count = 1, Price = product6.Price},
            }
        };

        var order5 = new Order
        {
            Customer = customer2,
            Date = new DateTime(2013, 1, 12),
            OrderDetail = new List<OrderDetail>
            {
```

```
                new OrderDetail {Product = product4, Count = 1, Price = product4.Price},
                new OrderDetail {Product = product5, Count = 2, Price = product5.Price},
                new OrderDetail {Product = product6, Count = 5, Price = product6.Price},
            }
        };

        context.Customers.Add(customer3);

        context.Orders.Add(order1);
        context.Orders.Add(order2);
        context.Orders.Add(order3);
        context.Orders.Add(order4);
        context.Orders.Add(order5);

        context.SaveChanges();
    }
```

و در ابتدای برنامه کد زیر را جهت مقداردهی اولیه به Database مان قرار می‌دهیم:

```
Database.SetInitializer(new MyTestDb());
```

در انتها ConnectionString را در App.Config به صورت زیر تعریف می‌کنیم:

```
<connectionStrings>
    <add name="StoreDb" connectionString="Data Source=.\SQLEXPRESS;
Initial Catalog=StoreDBTest;Integrated Security = true" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

بسیار خوب، حالا همه چیز محیاست برای اجرای اولین پرس و جو:

```
using (var context = new StoreDbContext())
{
    var query = context.Customers;

    foreach (var customer in query)
    {
        Console.WriteLine("Customer Name: {0}, Customer Family: {1}",
                                  customer.Name, customer.Family);
    }
}
```

پرس و جوی تعریف شده لیست تمام Customerها را باز می‌گرداند. query فقط یک "عبارت" پرس و جو هست و زمانی اجرا می‌شود که از آن درخواست نتیجه شود. در مثال بالا این درخواست در اجرای حلقه foreach اتفاق می‌افتد و درست در این لحظه است که دستور SQL ساخته شده و به Database فرستاده می‌شود. EF در این حالت تمام داده‌ها را در یک لحظه باز نمی‌گرداند بلکه این ارتباط فعال است تا حلقه به پایان برسد و تمام داده‌ها از database واکشی شود. خروجی به صورت زیر خواهد بود:

```
Customer Name: Vahid, Customer Family: Nasiri
Customer Name: Mohsen, Customer Family: Jamshidi
Customer Name: Mohsen, Customer Family: Akbari
```

نکته : با هر بار درخواست نتیجه از query ، پرس و جوی مربوطه دوباره به database فرستاده می‌شود که ممکن است مطلوب ما نباشد و باعث افت سرعت شود. برای جلوگیری از تکرار این عمل کافیست با استفاده از متد ToList پرس و جو را در لحظه تعریف به اجرا در آوریم

```
var customers = context.Customers.ToList();
```

خط بالا دیگر یک عبارت پرس و جو نخواهد بود بلکه لیست تمام Customer هاست که به یکباره از database بازگشت داده شده است. در ادامه هرجا که از customers استفاده کنیم دیگر پرس و جویی به database فرستاده نخواهد شد.

پرس و جوی زیر مشتریهایی که نام آنها Mohsen هست را باز می‌گرداند:

```
private static void Query3()
{
    using (var context = new StoreDbContext())
    {
        var methodSyntaxquery = context.Customers
                   .Where(c => c.Name == "Mohsen");
        var sqlSyntaxquery = from c in context.Customers
                             where c.Name == "Mohsen"
                             select c;

        foreach (var customer in methodSyntaxquery)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}",
                                  customer.Name, customer.Family);
        }
    }

    // Output:
    // Customer Name: Mohsen, Customer Family: Jamshidi
    // Customer Name: Mohsen, Customer Family: Akbari
}
```

همانطور که مشاهده می‌کنید پرس و جو به دو روش Method Syntax و Sql Syntax نوشته شده است.

روش Method Syntax روشی است که از متدهای الحاقی (Extention Method) و عبارت‌های لامبدا (Lambda Expersion) برای نوشتن پرس و جو استفاده می‌شود. اما #C روش Sql Syntax را که همانند دستورات SQL هست، نیز فراهم کرده است تا کسانیکه آشنایی با این روش دارند، از این روش استفاده کنند. در نهایت این روش به Method Syntax تبدیل خواهد شد بنابراین پیشنهاد می‌شود که از همین روش استفاده شود تا با دست و پنجه نرم کردن با این روش، از مزایای آن در بخشهای دیگر کدنویسی استفاده شود.

اگر به نوع Customers که در DbContext تعریف شده است، دقت کرده باشید، خواهید دید که DbSet می‌باشد. DbSet کلاس و اینترفیس‌های متفاوتی را پیاده سازی کرده است که در ادامه با آنها آشنا خواهیم شد:

IQueryable<TEntity>, IEnumerable<TEntity>, IQueryable, IEnumerable: که امکان استفاده از متدهای نام آشنای LINQ را برای ما فراهم می‌کند. البته فراموش نشود که EF از Provider ای با نام LINQ To Entity برای تفسیر پرس و جوی ما و ساخت دستور SQL متناظر آن استفاده می‌کند. بنابراین تمامی متدهایی که در LINQ To Object استفاده می‌شوند در اینجا قابل استفاده نیستند. بطور مثال اگر در پرس و جو از LastOrDefault روی Customer استفاده شود در زمان اجرا با خطای زیر مواجه خواهیم شد و در نتیجه در استفاده از این متدها به این مسئله باید دقت شود.

LINQ to Entities does not recognize the method 'Store.Model.Customer

LastOrDefault[Customer](System.Linq.IQueryable`1[Store.Model.Customer], System.Linq.Expressions.Expression`1[System.Func`2[Store.Model.Customer,System.Boolean]])' method, and this method cannot be translated into a store expression.

IDbSet<TEntity>: که دارای متدهای Add, Attach, Create, Find, Remove, Local می‌باشد و برای بحث ما Find و Local جهت ساخت پرس و جو استفاده می‌شوند که در ادامه توضیح داده خواهند شد.

DbQuery<TEntity>: که دارای متدهای AsNoTracking و Include می‌باشد و در ادامه توضیح داده خواهند شد.

متد Find: این متد کلید اصلی را به عنوان ورودی گرفته و برای بازگرداندن نتیجه مراحل زیر را طی می‌کند:

داده‌های موجود در حافظه را بررسی می‌کند یعنی آنهایی که Load و یا Attach شده اند.

داده هایی که به DbContext اضافه (Add) ولی هنوز در database درج نشده اند.

داده هایی که در database هستند ولی هنوز Load نشده اند.

Find در صورت پیدا نکردن Exception ای صادر نمی‌کند بلکه مقدار null را بر می‌گرداند.

```
private static void Query4()
{
    using (var context = new StoreDbContext())
    {
        var customer = context.Customers.Find(new Guid("2ee2fd32-e0e9-4955-bace-1995839d4367"));

        if (customer == null)
            Console.WriteLine("Customer not found");
        else
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", customer.Name,
customer.Family);
    }
}
```

با توجه به اینکه Idها توسط Database ساخته می‌شوند. شما باید از Id دیگری که موجود می‌باشد، استفاده کنید تا نتیجه ای برگشت داده شود.
نکته: در صورتیکه کلید اصلی شما از دو یا چند فیلد تشکیل شده بود. می‌بایست این دو یا چند مقدار را به عنوان پارامتر به Find بفرستید.

متد Single: گاهی نیاز هست که داده‌ای پرس و جو شود اما نه با کلید اصلی بلکه با شرط دیگری، در این حالت از Single استفاده می‌شود. این متد یک مقدار را باز می‌گرداند و در صورتی که صفر یا بیش از یک مقدار در شرط صدق کند exception صادر می‌کند. متد SingleOrDefault رفتاری مشابه دارد اما اگر مقداری در شرط صدق نکند مقدار پیش فرض را باز می‌گرداند.
نکته: مقدار پیش فرض بستگی به نوع خروجی دارد که اگر object باشد مقدار null و اگر بطور مثال نوع عددی باشد، صفر می‌باشد.

```
private static void Query5()
{
    using (var context = new StoreDbContext())
    {
        try
        {
            var customer1 = context.Customers.Single(c => c.Name == "Unkown");  // Exception: Sequence
contains no elements
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
```

```
        }

        try
        {
            var customer2 = context.Customers.Single(c => c.Name == "Mohsen");  // Exception: Sequence
contains more than one element
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        var customer3 = context.Customers.SingleOrDefault(c => c.Name == "Unkown");  // customer3 ==
null

        var customer4 = context.Customers.Single(c => c.Name == "Vahid");  // customer4 != null
    }
}
```

متد First: در صورتیکه به اولین نتیجه پرس و جو نیاز هست می‌توان از First استفاده کرد. اگر پرس و جو نتیجه در بر نداشته باشد یعنی null باشد exception صادر خواهد شد اما اگر FirstOrDefault استفاده شود مقدار پیش فرض برگردانده خواهد شد.

```
private static void Query6()
{
    using (var context = new StoreDbContext())
    {
        try
        {
            var customer1 = context.Customers.First(c => c.Name == "Unkown");  // Exception: Sequence
contains no elements
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        var customer2 = context.Customers.FirstOrDefault(c => c.Name == "Unknown");  // customer2 ==
null
        var customer3 = context.Customers.First(c => c.Name == "Mohsen");
    }
}
```

نظرات خوانندگان

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۵

خیلی ممنون مهندس، فقط اینکه در داخل سازنده StoreDbContext چرا به این شکل عمل کرده اید:

```
public StoreDbContext()
: base("name=StoreDb")
{
}
```

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۳۳

StoreDb نام مدخل Connection String ای هست که در AppConfig ایجاد شده

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۱:۴۶

ممنون از پاسخگوییتون ، در واقع اگر اینکارو نمیکردید باید در AppConfig نام Connection را StoredDbContext میگذاشتیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۲:۲۶

چندین روش برای تعریف رشته اتصالی در EF وجود دارد. بیشتر در اینجا

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۴:۳۰

بله البته به همراه namespace اگه اشتباه نکنم

نویسنده: جمشیدی فر
تاریخ: ۱۳۹۳/۰۸/۰۵ ۱۱:۳۷

متد FirstOrDefault باعث اجرای کوئری روی database می‌شه یا از context درون حافظه رکورد مورد نظر را بر میگرداند؟
اگر نیاز باشه که یک رکورد با اجرای کوئری از دیتابیس بازیابی بشه، و نه از context جاری، راه حل چیه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۸/۰۵ ۱۲:۲۹

بجای حدس و گمان، خروجی رو لاگ کنید: نمایش خروجی SQL کدهای Entity framework 6 در کنسول دیباگ ویژوال استودیو

نویسنده: حمیدرضا کبیری
تاریخ: ۱۳۹۳/۰۸/۱۰ ۱۴:۴۷

در پرس و جوهای معمولی ، بدین شکل عمل می‌شود که در نهایت نتیجه با شرط یک Id یا چیزی شبیه این مقایسه می‌شود .

```
var Id=1;
var books = (from b in db.Books
                            where b.bookId == Id
                            select new
                            {
                               //...
```

```
                                }).ToList();
```

حالا اگر شرط من بجای داشتن فقط یک Id لیستی از Id باشد چطور عمل کنم ؟

```
var booksId= new list<int>(){ 1 , 2 , 6 , 7};
var books = (from b in db.Books
                                where b.bookId == ???
                                select new
                                {
                                   //...

                                }).ToList();
```

چطور میتونم لیستی رو که دارم بجای مقایسه با یک Id ، با یک لیستی از Idها مقایسه کنم و نتیجه را بگیرم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۱۰ ۱۴:۵۸

از متد Contains استفاده کنید که به where in ترجمه می‌شود:

```
from book in db.Books
where booksId.Contains(book.bookId)
```

عنوان: **نوشتن پرس و جو در Entity Framework با استفاده از LINQ To Entity قسمت دوم**
نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۴/۱۳ ۱۰:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ to Entity

در قسمت قبل با نحوه اجرای پرس و جو آشنا شدید و همچنین به بررسی متدهای Find و Single و First و تفاوت‌های آنها پرداختیم. در این قسمت با خصوصیت Local و متد Load آشنا خواهیم شد. همانطور که در قسمت قبل دیدید، مقادیر اولیه‌ای برای Database و جداولمان مشخص کردیم. برای جدول Customer این داده‌ها را داشتیم:

ID	Name	Family
یک مقدار Guid	Vahid	Nasiri
یک مقدار Guid	Mohsen	Akbari
یک مقدار Guid	Mohsen	Jamshidi

ID توسط Database تولید می‌شوند به همین دلیل از ذکر مقداری مشخص خودداری شده است.
به کد زیر دقت کنید:

```
private static void Query7()
{
    using (var context = new StoreDbContext())
    {
        // Add
        context.Customers.Add(new Customer { Name = "Ali", Family = "Jamshidi" });

        // change
        var customer1 = context.Customers.Single(c => c.Family == "Jamshidi");
        customer1.Name = "Mohammad";

        // Remove
        var customer2 = context.Customers.Single(c => c.Family == "Akbari");
        context.Customers.Remove(customer2);

        var customers = context.Customers.Where(c => c.Name != "Vahid");

        foreach (var cust in customers)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", cust.Name, cust.Family)
        }
    }
}
```

همانطور که مشاهده می‌کنید عمل اضافه، تغییر و حذف روی Customer انجام شده ولی هنوز هیچ تغییری در Database ذخیره نشده است. آخرین پرس و جو چه نتیجه ای را دربر خواهد داشت؟

بله، فقط تغییر یک موجودیت در نظر گرفته شده است ولی اضافه و حذف نه!
نتیجه مهمی که حاصل می‌شود این است که در پرس و جوهایی که روی Database اجرا می‌شوند سه مورد را باید در نظر داشت:

داده هایی که اخیرا به DbContext اضافه شده‌اند ولی هنوز در Database ذخیره نشده‌اند، درنظر گرفته نخواهند شد.

داده هایی که در DbContext حذف شده‌اند ولی در Database هستند، در نتیجه پرس و جو خواهند بود.

داده هایی که قبلا از database توسط پرس و جوی دیگری گرفته شده و تغییر کرده‌اند، آن تغییرات در نتیجه پرس و جو موثر

خواهند بود.

پس پرس و جوهای LINQ ابتدا روی database انجام می‌شوند و idهای بازگشت داده شده با idهای موجود در DbContext مطابقت داده می‌شوند یا در DbContext وجود دارند که در این صورت آن موجودیت بازگشت داده می‌شود یا وجود ندارند که در این صورت موجودیتی که از Database خوانده شده، بازگشت داده می‌شوند.
برای درک بیشتر کد زیر را در نظر بگیرید:

```
private static void Query7_1()
{
    using (var context = new StoreDbContext())
    {
        // Add
        context.Customers.Add(new Customer { Name = "Ali", Family = "Jamshidi" });

        // change
        var customer1 = context.Customers.Single(c => c.Family == "Jamshidi");
        customer1.Name = "Vahid";

        // Remove
        var customer2 = context.Customers.Single(c => c.Family == "Akbari");
        context.Customers.Remove(customer2);

        var customers = context.Customers.Where(c => c.Name != "Vahid");

        foreach (var cust in customers)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", cust.Name, cust.Family)
        }
    }
}
```

این کد همان کد قبلی است اما نام customer1 در DbContext (که Mohsen بوده در Database) به Vahid تغییر کرده و پرس و جو روی نام هایی زده شده است که Vahid نباشند خروجی به صورت زیر خواهد بود:

Customer Name: Vahid, Customer Family: Jamshidi

Customer Name: Mohsen, Customer Family: Akbari Vahid در خروجی آمده در صورتیکه در شرط صدق نمی‌کند چراکه پرس و جو روی Database زده شده، جاییکه نام این مشتری Mohsen بوده اما موجودیتی بازگشت داده شده که دارای همان Id هست اما در DbContext دستخوش تغییر شده است.

Local: همانطور که قبلا اشاره شد خصوصیتی از DbSet می‌باشد که شامل تمام داده هایی هست که:

اخیرا از database پرس و جو شده است (می‌تواند تغییر کرده یا نکرده باشد)

اخیرا به Context اضافه شده است (توسط متد Add)

دقت شود که Local شامل داده‌هایی که از database خوانده شده و از Context، حذف (Remove) شده‌اند، نمی‌باشد.
نوع این خصوصیت ObservableCollection می‌باشد که می‌توان از آن برای Binding در پروژه‌های ویندوزی استفاده کرد.
به کد زیر دقت کنید:

```
private static void Query8()
{
    using (var context = new StoreDbContext())
    {
        // Add
        context.Customers.Add(new Customer { Name = "Ali", Family = "Jamshidi" });

        // change
        var customer1 = context.Customers.Single(c => c.Family == "Jamshidi");
```

```
        customer1.Name = "Mohammad";

        // Remove
        var customer2 = context.Customers.Single(c => c.Family == "Akbari");
        context.Customers.Remove(customer2);

        var customers = context.Customers.Local;

        foreach (var cust in customers)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", cust.Name, cust.Family);
        }
    }
}
```

کد بالا شبیه به کد قبلی می‌باشد با این تفاوت که در انتها foreach روی Local زده شده است. خروجی به صورت زیر خواهد بود:

همانطور که ملاحظه می‌کنید Local شامل Ali Jamshidi که اخیرا اضافه شده (ولی در Database ذخیره نشده) و Mohammad Jamshidi که از Database خوانده شده و تغییر کرده، می‌باشد اما شامل Mohsen Akbari که از Database خوانده شده اما در Context حذف شده است، نمی‌باشد.
می‌توان روی Local نیز پرس و جوی اجرا کرد. در این صورت از پروایدر LINQ To Object استفاده خواهد شد و درنتیجه دست بازتر هست و تمام امکانات این پروایدر می‌توان استفاده کرد.

Load: یکی دیگر از مواردی که باعث اجرای پرس و جو می‌شود متد Load می‌باشد که یک Extension Method می‌باشد. این متد در حقیقت یک پیمایش روی پرس و جو انجام می‌دهد و باعث بارگذاری داده‌ها در Context می‌شود. مانند استفاده از ToList البته بدون ساختن List که سربار ایجاد می‌کند.

```
private static void Query9()
{
    using (var context = new StoreDbContext())
    {
        var customers = context.Customers.Where(c => c.Name == "Mohsen");
        customers.Load();

        foreach (var cust in context.Customers.Local)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", cust.Name, cust.Family);
        }
    }
    // Output:
    // Customer Name: Mohsen, Customer Family: Akbari
    // Customer Name: Mohsen, Customer Family: Jamshidi
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۱۳ ۱۰:۴۸

این دوجمله رو

«- داده هایی که اخیرا به DbContext اضافه شده‌اند ولی هنوز در Database ذخیره نشده‌اند، درنظر گرفته نخواهند شد.
- داده هایی که در DbContext حذف شده‌اند ولی در Database هستند، در نتیجه پرس و جو خواهند بود.»

میشه خلاصه‌اش کرد به «تا زمانیکه SaveChanges فراخوانی نشه، از اطلاعات تغییر کرده نمیشه کوئری گرفت (کوئری‌ها همیشه روی دیتابیس انجام می‌شن)؛ اما خاصیت Local این تغییرات محلی رو داره یا اینکه در change tracker میشه موارد EntityState.Added | EntityState.Modified | EntityState.Unchanged رو هم کوئری گرفت».

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۴/۱۳ ۱۴:۷

دقیقا!
جهت تاکید بیشتر روی "اجرا شدن پرس و جو در Database نه DbContext" متد Query7_1 به متن اضافه شد

نویسنده: مجید_فاضلی نسب
تاریخ: ۱۳۹۲/۰۸/۲۶ ۲۳:۱۴

DbContext دقیقا چیه ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۲۷ ۰:۵۱

قسمت‌های 11 و 12 سری EF رو مطالعه کنید.

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۶:۵۳

با تشکر از مقاله آموزندتون، خواستم بدونم که مثلا در چه سناریویی بهتر است از متد .Localاستفاده کرد(یا به عبارتی این متد کی به درد کار ما میخورد)، با توجه به اینکه این متد اطلاعاتی که به اصطلاح In-Memory هستند را برای ما میآورد؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۷:۷

بیشتر برای استفاده در WPF و برنامه‌های دسکتاپ است .

نویسنده: پژمان
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۷:۱۰

خیلی ممنون، در مورد Load هم تو نت گشتم چیزی دستگیرم نشد، اگه در این مورد هم مقاله ای باز سراغ دارید ممنون میشم لینکشو بدید ممنوون

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۷:۲۵

در همان مقاله‌ای که لینک دادم، اواسط آن به Load هم پرداخته. کاربرد عملی آن در پروژه طراحی فریم ورک WPF با EF در سایت هست.

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۸:۱۲

یکی از کاربرداش در حذف اشیاء مرتبط

عنوان: نوشتن پرس و جو در Entity Framework با استفاده از LINQ To Entity قسمت سوم
نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۴/۲۰ ۱۸:۵۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ to Entity

اجرای پرس و جو روی داده‌های به هم مرتبط (Related Data)

اگر به موجودیت Customer دقت کنید دارای خصوصیتی با نام Orders می‌باشد که از نوع IList<Order> هست یعنی دارای لیستی از Order هاست بنابراین یک رابطه یک به چند بین Customer و Order وجود دارد. در ادامه به بررسی نحوه پرس و جو کردن روی داده‌های به هم مرتبط خواهیم پرداخت.

ابتدا به کد زیر دقت کنید:

```
private static void Query10()
{
    using (var context = new StoreDbContext())
    {
        var customers = context.Customers;
        foreach (var customer in customers)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", customer.Name,
customer.Family);
            foreach (var order in customer.Orders)
            {
                Console.WriteLine("\t Order Date: {0}", order.Date);
            }
        }
    }
}
```

اگر کد بالا را اجرا کنید هنگام اجرای حلقه داخلی با خطای زیر مواجه خواهید شد:

```
System.InvalidOperationException: There is already an open DataReader associated with this Command
which must be closed first
```

همانطور که قبلا اشاره شد EF با اجرای یک پرس و جو به یکباره داده‌ها را باز نمی‌گرداند بنابراین در حلقه اصلی که روی Customers زده شده است با هر پیمایش یک customer از Database فراخوانی می‌شود درنتیجه DataReader تا پایان یافتن حلقه باز می‌ماند. حال آنکه حلقه داخلی نیز برای خواندن Orderها نیاز به اجرای یک پرس و جو دارد بنابراین DataReader ای جدید باز می‌شود و در نتیجه با خطایی مبنی بر اینکه DataReader دیگری باز است، مواجه می‌شویم. برای حل این مشکل می‌بایست جهت باز بودن چند DataReader همزمان، کد زیر را به ConnectionString اضافه کنیم

```
MultipleActiveResultSets = true
```

که با این تغییر کد بالا به درستی اجرا می‌شود.

در بارگذاری داده‌های به هم مرتبط EF سه روش را در اختیار ما قرار می‌دهد:

Lazy Loading

Eager Loading

Explicit Loading

که در ادامه به بررسی آنها خواهیم پرداخت.

Lazy Loading: در این روش داده‌های مرتبط در صورت نیاز با یک پرس وجوی جدید که به صورت اتوماتیک توسط EF ساخته می‌شود، گرفته خواهند شد. کد زیر را در نظر بگیرید:

```
private static void Query11()
{
    using (var context = new StoreDbContext())
    {
        var customer = context.Customers.First();

        Console.WriteLine("Customer Name: {0}, Customer Family: {1}", customer.Name, customer.Family);
        foreach (var order in customer.Orders)
        {
            Console.WriteLine("\t Order Date: {0}", order.Date);
        }
    }
}
```

اگر این کد را اجرا کنید خواهید دید که یک بار پرس و جویی مبنی بر دریافت اولین Customer روی database زده خواهد شد و پس از چاپ آن در ادامه برای نمایش Orderهای این Customer پرس و جوی دیگری زده خواهد شد. در حقیقت پرس و جوی اول فقط Customer را بازگشت می‌دهد و در ادامه، اول حلقه، جایی که نیاز به Orderهای این Customer می‌شود EF پرس و جو دوم را بصورت هوشمندانه و اتوماتیک اجرا می‌کند. به این روش بارگذاری داده‌های مرتبط Lazy Loading گفته می‌شود که به صورت پیش فرض در EF فعال است.

برای غیرفعال کردن این روش، کد زیر را اجرا کنید:

```
context.Configuration.LazyLoadingEnabled = false;
```

EF از dynamic proxy برای Lazy Loading استفاده می‌کند. به این صورت که در زمان اجرا کلاسی جدید که از کلاس POCO مان ارث برده است، ساخته می‌شود. این کلاس proxy می‌باشد و در آن navigation propertyها بازنویسی شده‌اند و کمی منطق برای خواندن داده‌های وابسته اضافه شده است.

برای ایجاد dynamic proxy شروط زیر لازم است:

- کلاس POCO می‌بایست public بوده و sealed نباشد.
- Navigation propertyها می‌بایست virtual باشد.

در صورتیکه هرکدام از این دو شرط برقرار نباشند کلاس proxy ساخته نمی‌شود و Lazy Loading حتی در صورت فعال بودن انجام نخواهد شد. مثلا اگر پراپرتی Orders در کلاس Customer مان virtual نباشد. در شروع حلقه کد بالا پرس و جوی جدید اجرا نشده و در نتیجه مقدار این پراپرتی null خواهد ماند.

Lazy Loading به ما در عدم بارگذاری داده‌های مرتبط که به آنها نیازی نداریم، کمک می‌کند. اما در صورتیکه به داده‌های مرتبط نیاز داشته باشیم "مسئله Select n+1" پیش خواهد آمد که باید این مسئله را مد نظر داشته باشیم.

مسئله Select n+1: کد زیر را در نظر بگیرد

```
private static void Query12()
{
    using (var context = new StoreDbContext())
    {
        var customers = context.Customers;
        foreach (var customer in customers)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", customer.Name,
customer.Family);
            foreach (var order in customer.Orders)
            {
                Console.WriteLine("\t Order Date: {0}", order.Date);
            }
        }
    }
}
```

هنگام اجرای کد بالا یک پرس و جو برای خواندن Customerها زده خواهد شد و به ازای هر Customer یک پرس و جوی دیگر برای گرفتن Orderها زده خواهد شد. در این صورت پرس و جوی اول ما اگر n مشتری را برگرداند، n پرس و جو نیز برای گرفتن Orderها زده خواهد شد که روهم n+1 دستور Select می‌شود. این تعداد پرس و جو موجب عدم کارایی می‌شود و برای رفع این مسئله نیاز به امکانی جهت بارگذاری هم زمان داده‌های مرتبط مورد نیاز خواهد بود. این امکان با استفاده از Eager Loading

برآورده می‌شود.

روش Eager Loading: هنگامی که در یک پرس و جو نیاز به بارگذاری همزمان داده‌های مرتبط نیز باشد، از این روش استفاده می‌شود. برای این منظور از متد Include استفاده می‌شود که ورودی آن navigation property مربوطه می‌باشد. این پارامتر ورودی را همانطور که در کد زیر مشاهده می‌کنید، می‌توان به صورت string و یا Lambda Expression مشخص کرد.
دقت شود که برای حالت Lambda Expression بایدSystem.Data.Entity به usingها اضافه شود.

```
private static void Query13()
{
    using (var context = new StoreDbContext())
    {
        var customers = context.Customers.Include(c => c.Orders);
        //var customers = context.Customers.Include("Orders");
        foreach (var customer in customers)
        {
            Console.WriteLine("Customer Name: {0}, Customer Family: {1}", customer.Name,
customer.Family);
            foreach (var order in customer.Orders)
            {
                Console.WriteLine("\t Order Date: {0}", order.Date);
            }
        }
}
```

در این صورت یک پرس و جو به صورت join اجرا خواهد شد.
اگر داده‌های مرتبط در چند سطح باشند، می‌توان با دادن مسیر داده‌های مرتبط اقدام به بارگذاری آنها کرد. به مثالهای زیر توجه کنید:

```
context.OrderDetails.Include(o => o.Order.Customer)
```

در پرس و جوی بالا به ازای هر OrderDetail داده‌های مرتبط Order و Customer آن بارگذاری می‌شود.

```
context.Orders.Include(o => o.OrderDetail.Select(od => od.Product))
```

در پرس و جوی بالا به ازای هر Order لیست OrderDetail ها و برای هر OrderDetail داده مرتبط Product آن بارگذاری می‌شود.

```
context.Orders.Include(o => o.Customer).Include(o => o.OrderDetail)
```

در پرس و جوی بالا به ازای هر Order داده‌های مرتبط OrderDetail و Customer آن بارگذاری می‌شود.

روش Explicit Loading: این روش مانند Lazy Loading می‌باشد که می‌توان داده‌های مرتبط را جداگانه فراخوانی کرد اما نه به صورت اتوماتیک توسط EF بلکه به صورت صریح توسط خودمان انجام می‌شود. این روش حتی اگر navigation propertyهای ما virtual نباشند نیز قابل انجام است. برای انجام این روش از متد DbContext.Entry استفاده می‌شود.

```
private static void Query14()
{
    using (var context = new StoreDbContext())
    {
        var customer = context.Customers.First(c => c.Family == "Jamshidi");

        context.Entry(customer).Collection(c => c.Orders).Load();

        foreach (var order in customer.Orders)
        {
            Console.WriteLine(order.Date);
        }
    }
}
```

در پرس و جوی بالا تمام Orderهای یک Customer به صورت جدا گرفته شده است برای این منظور از چون Orders یک لیست می‌باشد، از متد Collection استفاده شده است.

```
private static void Query15()
{
    using (var context = new StoreDbContext())
    {
        var order = context.Orders.First();

        context.Entry(order).Reference(o => o.Customer).Load();

        Console.WriteLine(order.Customer.FullName);
    }
}
```

در پرس و جوی بالا Customer یک Order صراحتا و به صورت جداگانه از database گرفته شده است.
با توجه به دو مثال بالا مشخص است که اگر داده مرتبط ما به صورت لیست است از Collection و درغیر این صورت از Reference استفاده می‌شود.
در صورتیکه بخواهیم ببینیم آیا داده‌ی مرتبط مان بازگذاری شده است یا خیر، از خصوصیت IsLoaded به صورت زیر استفاده می‌کنیم:

```
if (context.Entry(order).Reference(o => o.Customer).IsLoaded)
    context.Entry(order).Reference(o => o.Customer).Load();
```

و در آخر اگر بخواهیم روی داده‌های مرتبط پرس و جو اجرا کنیم نیز این قابلیت وجود دارد. برای این منظور از Query استفاده می‌کنیم.

```
private static void Query16()
{
    using (var context = new StoreDbContext())
    {
        var customer = context.Customers.First(c => c.Family == "Jamshidi");

        IQueryable<Order> query = context.Entry(customer).Collection(c => c.Orders).Query();

        var order = query.First();
    }
}
```

نظرات خوانندگان

نویسنده: مهدی زارعی
تاریخ: ۱۳۹۲/۰۵/۲۹ ۱۱:۱۴

این سری مطالب بسیار خوب و مفید است .از نویسنده محترم خواهش می‌کنم نگارش این مجموعه را متوقف نکند .
در صورتی که برای ایشون امکان پذیر نیست خواهش می‌کنم منبع یا منابعی که از آن‌ها در مورد این سری مقالات به آن رجوع کرده اند را معرفی کنند.
با تشکر از زحماتشان

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۵/۲۹ ۱۲:۸

منبع کتابهایی هست که در اینجا معرفی شده

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۵/۳۱ ۰:۴۰

در خصوص این قسمت:
"... در پرس و جوی بالا به ازای هر OrderDetail داده‌های مرتبط Order و Customer آن بارگذاری می‌شود.

context.Orders.Include(o => o.OrderDetail.Select(od => od.Product))	1

" بهتره برای ابهام زدایی ذکر کنید که OrderDetail یک Collection است و نمیتوان مانند پرس و جوی مثال قبلی از o=> o.OrderDerail.Product استفاده کرد.

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۵/۳۱ ۰:۴۳

در صورتیکه بخواهیم ببینیم آیا داده‌ی مرتبط مان بازگذاری شده است یا خیر، از خصوصیت IsLoaded به صورت زیر استفاده می‌کنیم:

```
if (context.Entry(order).Reference(o => o.Customer).IsLoaded)
```

منظور Not Isloaded بوده که ظاهرا ! جا افتاده

عنوان: **تفاوت بین IQueryable و IEnumerable در حین کار با ORMs**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۰۷ ۱۷:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: LINQ

متد زیر را که یکی از اشتباهات رایج حین استفاده از LINQ خصوصا جهت Binding اطلاعات است، در نظر بگیرید:

```
IQueryable<Customer> GetCustomers()
```

این متد در حقیقت هیچ چیزی را Get نمی‌کند! نام اصلی آن GetQueryableCustomers و یا GetQueryObjectForCustomersاست.
IQueryable قلب LINQ است و تنها بیانگر یک عبارت (expression) از رکوردهایی می‌باشد که مد نظر شما است و نه بیشتر.

```
IQueryable<Customer> youngCustomers = repo.GetCustomers().Where(m => m.Age < 15);
```

برای مثال زمانیکه یک IQueryable را همانند مثال فوق فیلتر می‌کنید نیز هنوز چیزی از بانک اطلاعاتی یا منبع داده‌ای دریافت نشده است. هنوز هیچ اتفاقی رخ نداده است و هنوز رفت و برگشتی به منبع داده‌ای صورت نگرفته است.
به آن باید به شکل یک expression builder نگاه کرد و نه لیستی از اشیاء فیلتر شده‌ی ما. به این مفهوم، deferred execution (اجرای به تاخیر افتاده) نیز گفته می‌شود (باید دقت داشت که IQueryable هم یک نوع IEnumerable است به علاوه expression trees که مهم‌ترین وجه تمایز آن نیز می‌باشد).
برای مثال در عبارت زیر تنها در زمانیکه متد ToList فراخوانی می‌شود، کل عبارت LINQ ساخته شده، به عبارت SQL متناظر با آن ترجمه شده، اطلاعات از دیتابیس اخذ گردیده و حاصل به صورت یک لیست بازگشت داده می‌شود:

```
IList<Competitor> competitorRecords =  competitorRepository
  .Competitors
  .Where(m => !m.Deleted)
  .OrderBy(m => m.countryId)
  .ToList(); //فقط اینجا است که اس کیوال نهایی تولید می‌شود
```

در مورد IEnumerable ها چطور؟

```
IEnumerable<Product> products = repository.GetProducts();
var productsOver25 = products.Where(p => p.Cost >= 25.00);
```

دو سطر فوق به این معنا است:
لطفا ابتدا به بانک اطلاعاتی رجوع کن و تمام رکوردهای محصولات موجود را بازگشت بده. سپس بر روی این حجم بالای اطلاعات، محصولاتی را که قیمت بالای 25 دارند، فیلتر کن.

اگر همین دو سطر را با IQueryable بازنویسی کنیم چطور؟

```
IQueryable<Product> products = repository.GetQueryableProducts();
var productsOver25 = products.Where(p => p.Cost >= 25.00);
```

در سطر اول تنها یک عبارت LINQ ساخته شده است و بس. در سطر دوم نیز به همین صورت. در طی این دو سطر حتی یک رفت و برگشت به بانک اطلاعاتی صورت نخواهد گرفت. در ادامه اگر این اطلاعات به نحوی Select شوند (یا ToList فراخوانی شود، یا در طی یک حلقه برای مثال Iteration ایی روی این حاصل صورت گیرد یا موارد مشابه دیگر)، آنگاه کوئری SQL متناظر با عبارت LINQ فوق ساخته شده و بر روی بانک اطلاعاتی اجرا خواهد شد.

بدیهی است این روش منابع کمتری را نسبت به حالتی که تمام اطلاعات ابتدا دریافت شده و سپس فیلتر می‌شوند، مصرف می‌کند (حالت بازگشت تمام اطلاعات ممکن است شامل 20000 رکورد باشد، اما حالت دوم شاید فقط 5 رکورد را بازگشت دهد).

سؤال: پس IQueryable بسیار عالی است و از این پس کلا از IEnumerable ها دیگر نباید استفاده کرد؟
خیر! توصیه اکید طراحان این است که لطفا تا حد امکان متدهایی که IQueryable بازگشت می‌دهند ایجاد نکنید! IQueryable یعنی اینکه این نقطه‌ی آغازین کوئری در اختیار شما، بعد برو هر کاری که دوست داشتی با آن در طی لایه‌های مختلف انجام بده و هر زمانیکه دوست داشتی از آن یک خروجی تهیه کن. خروجی IQueryable به معنای مشخص نبودن زمان اجرای نهایی کوئری و همچنین مبهم بودن نحوه‌ی استفاده از آن است. به همین جهت متدهایی را طراحی کنید که IEnumerable بازگشت می‌دهند اما در بدنه‌ی آن‌ها به نحو صحیح و مطلوبی از IQueryable استفاده شده است. به این صورت حد و مرز یک متد کاملا مشخص می‌شود. متدی که واقعا همان فیلتر کردن محصولات را انجام می‌دهد، همان 5 رکورد را بازگشت خواهد داد؛ اما با استفاده از یک لیست یا یک IEnumerable و نه یک IQueryable که پس از فراخوانی متد نیز به هر نحو دلخواهی قابل تغییر است.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۰۸/۰۷ ۲۲:۰۱:۴۷

جالب بود. من هم چند وقت پیش به این موضوع برخورد کرده بودم: http://stackoverflow.com/questions/3949823/why-skip-and-take-does-not-work-when-passing-through-a-method

حتی یک مطلب کوچولو هم برای آن آماده کرده و در سیستم اتوماتیک وبلاگم برای انتشار گذاشته‌ام.

نویسنده: سامان نام نیک
تاریخ: ۱۳۸۹/۰۸/۰۸ ۱۱:۱۴:۳۸

مدت ها بود که سوال فوق در ذهنم بود
از توضیح مختصر و مفیدتون ممنون

نویسنده: علی اقدم
تاریخ: ۱۳۸۹/۰۸/۰۹ ۱۲:۱۳:۵۱

کاملا درسته، به خاطر Tricky بودن IQueryable شدیدا توصیه می کنم که اگر معماری چند لایه کار می کنید اصلا لایه Bussiness داده ها رو به صورت IQueryable به UI پاس نکنه و در عوض می تونید از IList استفاده کنید.

آقای نصیری بسیار جالب و آموزنده بود،ممنون

نویسنده: کیان
تاریخ: ۱۳۹۱/۰۶/۲۸ ۱۶:۲۰

آیا می‌شه به نوع **IList** بسنده کرد یا کاملا بسته به جایی که استفاده می‌کنیم ممکنه فرق کنه این قضیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۸ ۱۶:۲۸

بستگی به مکان استفاده داره. اگر قرار است دو یا چند جستجو را انجام دهید، اینکارها باید با IQueryable داخل یک متد انجام شود، اما خروجی متد فقط باید لیست حاصل باشد؛ نه IQueryable ایی که انتهای آن باز است و سبب نشتی لایه سرویس شما در لایه‌های دیگر خواهد شد. IQueryable فقط یک expression است. هنوز اجرا نشده. زمانیکه First، ToList و امثال آن روی این عبارت فراخوانی شود تبدیل به SQL شده و سپس بر روی بانک اطلاعاتی اجرا می‌شود. به این deferred execution یا اجرای به تعویق افتاده گفته می‌شود.
اگر این عبارت را در اختیار لایه‌های دیگر قرار دهید، یعنی انتهای کار را بازگذاشته‌اید و حد و حدود سیستم شما مشخص نیست. شما اگر IQueryable بازگشت دهید، در لایه‌ای دیگر می‌شود یک join روی آن نوشت و اطلاعات چندین جدول دیگر را استخراج کرد؛ درحالیکه نام متد شما GetUsers بوده. بنابراین بهتر است به صورت صریح اطلاعات را به شکل List بازگشت دهید، تا انتهای کار باز نمانده و طراحی شما نشتی نداشته باشد.
طراحی یک لایه سرویس که خروجی IQueryable دارد نشتی دار درنظر گرفته شده و توصیه نمی‌شود. اصطلاحا leaky abstraction هم به آن گفته می‌شود؛ چون طراح نتوانسته حد و مرز سیستم خودش را مشخص کند و همچنین نتوانسته سازوکار درونی آن‌را به خوبی کپسوله سازی و مخفی نماید.

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۱/۰۶/۲۹ ۹:۱۱

تفاوت بازگشت متد از نوع List و IList در اینجا چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۹ ۹:۳۱

این‌ها بیشتر مباحث طراحی API است. اگر از List استفاده کنید، مصرف کننده کتابخانه شما مجبور است فقط از List استفاده کند. List صرفا یک پیاده سازی خاص از IList است.

اگر از اینترفیس و قرارداد IList استفاده شود، آزادی عمل بیشتری را در اختیار مصرف کننده خواهید گذاشت. در اینجا مصرف کننده می‌تواند از هر پیاده سازی دلخواهی از IList برای کار با API شما استفاده کند. حتی مواردی که در زمان طراحی API اصلی وجود خارجی نداشته‌اند و بعدها پیاده سازی خواهند شد.

عنوان: EF Code First #10
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۳ ۱۷:۵۴:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

حین کار با ORMهای پیشرفته، ویژگی‌های جالب توجهی در اختیار برنامه نویس‌ها قرار می‌گیرد که در زمان استفاده از کلاس‌های متداول SQLHelper از آن‌ها خبری نیست؛ مانند:

الف) Deferred execution

ب) Lazy loading

ج) Eager loading

نحوه بررسی SQL نهایی تولیدی توسط EF

برای توضیح موارد فوق، نیاز به مشاهده خروجی SQL نهایی حاصل از ORM است و همچنین شمارش تعداد بار رفت و برگشت به بانک اطلاعاتی. بهترین ابزاری را که برای این منظور می‌توان پیشنهاد داد، برنامه EF Profiler است. برای دریافت آن می‌توانید به این آدرس مراجعه کنید: (^) و (^)

پس از وارد کردن نام و آدرس ایمیل، یک مجوز یک ماهه آزمایشی، به آدرس ایمیل شما ارسال خواهد شد.
زمانیکه این فایل را در ابتدای اجرای برنامه به آن معرفی می‌کنید، محل ذخیره سازی نهایی آن جهت بازبینی بعدی، مسیر MyUserName\Local Settings\Application Data\EntityFramework Profiler خواهد بود.

استفاده از این برنامه هم بسیار ساده است:
الف) در برنامه خود، ارجاعی را به اسمبلی HibernatingRhinos.Profiler.Appender.dll که در پوشه برنامه EFProf موجود است، اضافه کنید.
ب) در نقطه آغاز برنامه، متد زیر را فراخوانی نمائید:

```
HibernatingRhinos.Profiler.Appender.EntityFramework.EntityFrameworkProfiler.Initialize();
```

نقطه آغاز برنامه می‌تواند متد Application_Start برنامه‌های وب، در متد Program.Main برنامه‌های ویندوزی کنسول و WinForms و در سازنده کلاس App برنامه‌های WPF باشد.
ج) برنامه EFProf را اجرا کنید.

مزایای استفاده از این برنامه
1) وابسته به بانک اطلاعاتی مورد استفاده نیست. (برخلاف برای مثال برنامه معروف SQL Server Profiler که فقط به همراه SQL Server ارائه می‌شود)
2) خروجی SQL نمایش داده شده را فرمت کرده و به همراه Syntax highlighting نیز هست.
3) کار این برنامه صرفا به لاگ کردن SQL تولیدی خلاصه نمی‌شود. یک سری از Best practices را نیز به شما گوشزد می‌کند. بنابراین اگر نیاز دارید سیستم خود را بر اساس دیدگاه یک متخصص بررسی کنید (یک Code review ارزشمند)، این ابزار می‌تواند بسیار مفید باشد.
4) می‌تواند کوئری‌های سنگین و سبک را به خوبی تشخیص داده و گزارشات آماری جالبی را به شما ارائه دهد.
5) می‌تواند دقیقا مشخص کند، کوئری را که مشاهده می‌کنید از طریق کدام متد در کدام کلاس صادر شده است و دقیقا از چه سطری.
6) امکان گروه بندی خودکار کوئری‌های صادر شده را بر اساس DbContext مورد استفاده به همراه دارد.
و ...

استفاده از این برنامه حین کار با EF «الزامی» است! (البته نسخه‌های NH و سایر ORMهای دیگر آن نیز موجود است و این مباحث در مورد تمام ORMهای پیشرفته صادق است)
مدام باید بررسی کرد که صفحه جاری چه تعداد کوئری را به بانک اطلاعاتی ارسال کرده و به چه نحوی. همچنین آیا می‌توان با اعمال اصلاحاتی، این وضع را بهبود بخشید. بنابراین عدم استفاده از این برنامه حین کار با ORMs، همانند راه رفتن در خواب است! ممکن است تصور کنید برنامه دارد به خوبی کار می‌کند اما ... در پشت صحنه فقط صفحه جاری برنامه، 100 کوئری را به بانک اطلاعاتی ارسال کرده، در حالیکه شما تنها نیاز به یک کوئری داشته‌اید.

کلاس‌های مدل مثال جاری

کلاس‌های مدل مثال جاری از یک دپارتمان که دارای تعدادی کارمند می‌باشد، تشکیل شده است. ضمنا هر کارمند تنها در یک دپارتمان می‌تواند مشغول به کار باشد و رابطه many-to-many نیست :

```
using System.Collections.Generic;

namespace EF_Sample06.Models
{
    public class Department
    {
        public int DepartmentId { get; set; }
        public string Name { get; set; }

        //Creates Employee navigation property for Lazy Loading (1:many)
        public virtual ICollection<Employee> Employees { get; set; }
    }
}
```

```
namespace EF_Sample06.Models
{
    public class Employee
    {
        public int EmployeeId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        //Creates Department navigation property for Lazy Loading
        public virtual Department Department { get; set; }
    }
}
```

نگاشت دستی این کلاس‌ها هم ضرورتی ندارد، زیرا قراردادهای توکار EF Code first را رعایت کرده و EF در اینجا به سادگی می‌تواند primary key و روابط one-to-many را بر اساس navigation properties تعریف شده، تشخیص دهد.

در اینجا کلاس Context برنامه به شرح زیر است:

```
using System.Data.Entity;
using EF_Sample06.Models;

namespace EF_Sample06.DataLayer
{
    public class Sample06Context : DbContext
    {
        public DbSet<Department> Departments { set; get; }
        public DbSet<Employee> Employees { set; get; }
    }
}
```

و تنظیمات ابتدایی نحوه به روز رسانی و آغاز بانک اطلاعاتی نیز مطابق کدهای زیر می‌باشد:

```
using System.Collections.Generic;
using System.Data.Entity.Migrations;
using EF_Sample06.Models;

namespace EF_Sample06.DataLayer
{
    public class Configuration : DbMigrationsConfiguration<Sample06Context>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(Sample06Context context)
        {
            var employee1 = new Employee { FirstName = "f name1", LastName = "l name1" };
            var employee2 = new Employee { FirstName = "f name2", LastName = "l name2" };
            var employee3 = new Employee { FirstName = "f name3", LastName = "l name3" };
            var employee4 = new Employee { FirstName = "f name4", LastName = "l name4" };

            var dept1 = new Department { Name = "dept 1", Employees = new List<Employee> { employee1,
employee2 } };
            var dept2 = new Department { Name = "dept 2", Employees = new List<Employee> { employee3 }
};
            var dept3 = new Department { Name = "dept 3", Employees = new List<Employee> { employee4 }
};

            context.Departments.Add(dept1);
            context.Departments.Add(dept2);
            context.Departments.Add(dept3);
            base.Seed(context);
        }
    }
}
```

نکته: تهیه خروجی XML از نگاشت‌های خودکار تهیه شده

اگر علاقمند باشید که پشت صحنه نگاشت‌های خودکار EF Code first را در یک فایل XML جهت بررسی بیشتر ذخیره کنید، می‌توان از متد کمکی زیر استفاده کرد:

```
void ExportMappings(DbContext context, string edmxFile)
{
     var settings = new XmlWriterSettings { Indent = true };
     using (XmlWriter writer = XmlWriter.Create(edmxFile, settings))
     {
         System.Data.Entity.Infrastructure.EdmxWriter.WriteEdmx(context, writer);
     }
}
```

بهتر است پسوند فایل XML تولیدی را edmx قید کنید تا بتوان آن‌را با دوبار کلیک بر روی فایل، در ویژوال استودیو نیز مشاهده کرد:

```
using (var db = new Sample06Context())
{
     ExportMappings(db, "mappings.edmx");
}
```

الف) بررسی Deferred execution یا بارگذاری به تاخیر افتاده

برای توضیح مفهوم Deferred loading/execution بهترین مثالی را که می‌توان ارائه داد، صفحات جستجوی ترکیبی در برنامه‌ها است. برای مثال یک صفحه جستجو را طراحی کرده‌اید که حاوی دو تکست باکس دریافت FirstName و LastName کاربر است. کنار هر کدام از این تکست باکس‌ها نیز یک چک‌باکس قرار دارد. به عبارتی کاربر می‌تواند جستجویی ترکیبی را در اینجا انجام دهد. نحوه پیاده سازی صحیح این نوع مثال‌ها در EF Code first به چه نحوی است؟

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using EF_Sample06.DataLayer;
using EF_Sample06.Models;

namespace EF_Sample06
{
    class Program
    {
        static IList<Employee> FindEmployees(string fName, string lName, bool byName, bool byLName)
        {
            using (var db = new Sample06Context())
            {
                IQueryable<Employee> query = db.Employees.AsQueryable();

                if (byLName)
                {
                    query = query.Where(x => x.LastName == lName);
                }

                if (byName)
                {
                    query = query.Where(x => x.FirstName == fName);
                }
```

```
                return query.ToList();
            }
        }

        static void Main(string[] args)
        {
            // note: remove this line if you received : create database is not supported by this
provider.
            HibernatingRhinos.Profiler.Appender.EntityFramework.EntityFrameworkProfiler.Initialize();

            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample06Context,
Configuration>());

            var list = FindEmployees("f name1", "l name1", true, true);
            foreach (var item in list)
            {
                Console.WriteLine(item.FirstName);
            }
        }
    }
}
```

نحوه صحیح این نوع پیاده سازی ترکیبی را در متد FindEmployees مشاهده می‌کنید. نکته مهم آن، استفاده از نوع IQueryable و متد AsQueryable است و امکان ترکیب کوئری‌ها با هم.

به نظر شما با فراخوانی متد FindEmployees به نحو زیر که هر دو شرط آن توسط کاربر انتخاب شده است، چه تعداد کوئری به بانک اطلاعاتی ارسال می‌شود؟

```
var list = FindEmployees("f name1", "l name1", true, true);
```

شاید پاسخ دهید که سه بار : یکبار در متد db.Employees.AsQueryable و دوبار هم در حین ورود به بدنه شرط‌های یاد شده و اینجا است که کسانی که قبلا با رویه‌های ذخیره شده کار کرده باشند، شروع به فریاد و فغان می‌کنند که ما قبلا این مسایل رو با یک SP در یک رفت و برگشت مدیریت می‌کردیم!

پاسخ صحیح: «فقط یکبار»! آن‌هم تنها در زمان فراخوانی متد ToList و نه قبل از آن.

برای اثبات این مدعا نیاز است به خروجی SQL لاگ شده توسط EF Profiler مراجعه کرد:

```
SELECT [Extent1].[EmployeeId]              AS [EmployeeId],
       [Extent1].[FirstName]               AS [FirstName],
       [Extent1].[LastName]                AS [LastName],
       [Extent1].[Department_DepartmentId] AS [Department_DepartmentId]
FROM   [dbo].[Employees] AS [Extent1]
WHERE  ([Extent1].[LastName] = 'l name1' /* @p__linq__0 */)
       AND ([Extent1].[FirstName] = 'f name1' /* @p__linq__1 */)
```

IQueryable قلب LINQ است و تنها بیانگر یک عبارت (expression) از رکوردهایی می‌باشد که مد نظر شما است و نه بیشتر. برای مثال زمانیکه یک IQueryable را همانند مثال فوق فیلتر می‌کنید، هنوز چیزی از بانک اطلاعاتی یا منبع داده‌ای دریافت نشده است. هنوز هیچ اتفاقی رخ نداده است و هنوز رفت و برگشتی به منبع داده‌ای صورت نگرفته است. به آن باید به شکل یک expression builder نگاه کرد و نه لیستی از اشیاء فیلتر شده‌ی ما. به این مفهوم، deferred execution (اجرای به تاخیر افتاده) نیز گفته می‌شود.

کوئری LINQ شما تنها زمانی بر روی بانک اطلاعاتی اجرا می‌شود که کاری بر روی آن صورت گیرد مانند فراخوانی متد ToList، فراخوانی متد First یا FirstOrDefault و امثال آن. تا پیش از این فقط به شکل یک عبارت در برنامه وجود دارد و نه بیشتر.

اطلاعات بیشتر: « تفاوت بین IQueryable و IEnumerable در حین کار با ORMs »

ب) بررسی Lazy Loading یا واکشی در صورت نیاز

در مطلب جاری اگر به کلاس‌های مدل برنامه دقت کنید، تعدادی از خواص به صورت virtual تعریف شده‌اند. چرا؟
تعریف یک خاصیت به صورت virtual، پایه و اساس lazy loading است و به کمک آن، تا به اطلاعات شیءایی نیاز نباشد، وهله سازی نخواهد شد. به این ترتیب می‌توان به کارآیی بیشتری در حین کار با ORMs رسید. برای مثال در کلاس‌های فوق، اگر تنها نیاز به دریافت نام یک دپارتمان هست، نباید حین وهله سازی از شیء دپارتمان، شیء لیست کارمندان مرتبط با آن نیز وهله سازی شده و از بانک اطلاعاتی دریافت شوند. به این وهله سازی با تاخیر، lazy loading گفته می‌شود.
Lazy loading پیاده سازی ساده‌ای نداشته و مبتنی است بر بکارگیری AOP frameworks یا کتابخانه‌هایی که امکان تشکیل اشیاء Proxy پویا را در پشت صحنه فراهم می‌کنند. علت virtual تعریف کردن خواص رابط نیز به همین مساله بر می‌گردد، تا این نوع کتابخانه‌ها بتوانند در نحوه تعریف اینگونه خواص virtual در زمان اجرا، در پشت صحنه دخل و تصرف کنند. البته حین استفاده از EF یا انواع و اقسام ORMs دیگر با این نوع پیچیدگی‌ها روبرو نخواهیم شد و تشکیل اشیاء Proxy در پشت صحنه انجام می‌شوند.

یک مثال: قصد داریم اولین دپارتمان ثبت شده در حین آغاز برنامه را یافته و سپس لیست کارمندان آن‌را نمایش دهیم:

```
using (var db = new Sample06Context())
{
    var dept1 = db.Departments.Find(1);
    if (dept1 != null)
    {
        Console.WriteLine(dept1.Name);
        foreach (var item in dept1.Employees)
        {
             Console.WriteLine(item.FirstName);
        }
    }
}
```

رفتار یک ORM جهت تعیین اینکه آیا نیاز است برای دریافت اطلاعات بین جداول Join صورت گیرد یا خیر، واکشی حریصانه و غیرحریصانه را مشخص می‌سازد.
در حالت واکشی حریصانه به ORM خواهیم گفت که لطفا جهت دریافت اطلاعات فیلدهای جداول مختلف، از همان ابتدای کار در پشت صحنه، Join های لازم را تدارک ببین. در حالت واکشی غیرحریصانه به ORM خواهیم گفت به هیچ عنوان حق نداری Join ایی را تشکیل دهی. هر زمانی که نیاز به اطلاعات فیلدی از جدولی دیگر بود باید به صورت مستقیم به آن مراجعه کرده و آن مقدار را دریافت کنی.
به صورت خلاصه برنامه نویس در حین کار با ORM های پیشرفته نیازی نیست Join بنویسد. تنها باید ORM را طوری تنظیم کند که آیا اینکار را حتما خودش در پشت صحنه انجام دهد (واکشی حریصانه)، یا اینکه خیر، به هیچ عنوان SQL های تولیدی در پشت صحنه نباید حاوی Join باشند (lazy loading).

در مثال فوق به صورت خودکار دو کوئری به بانک اطلاعاتی ارسال می‌گردد:

```
SELECT [Limit1].[DepartmentId] AS [DepartmentId],
       [Limit1].[Name]         AS [Name]
FROM   (SELECT TOP (2) [Extent1].[DepartmentId] AS [DepartmentId],
                       [Extent1].[Name]         AS [Name]
        FROM   [dbo].[Departments] AS [Extent1]
        WHERE  [Extent1].[DepartmentId] = 1 /* @p0 */) AS [Limit1]


SELECT [Extent1].[EmployeeId]              AS [EmployeeId],
       [Extent1].[FirstName]               AS [FirstName],
       [Extent1].[LastName]                AS [LastName],
       [Extent1].[Department_DepartmentId] AS [Department_DepartmentId]
FROM   [dbo].[Employees] AS [Extent1]
```

```
WHERE  ([Extent1].[Department_DepartmentId] IS NOT NULL)
       AND ([Extent1].[Department_DepartmentId] = 1 /* @EntityKeyValue1 */)
```

یکبار زمانیکه قرار است اطلاعات دپارتمان یک (db.Departments.Find) دریافت شود. تا این لحظه خبری از جدول Employees نیست. چون lazy loading فعال است و فقط اطلاعاتی را که نیاز داشته‌ایم فراهم کرده است.
زمانیکه برنامه به حلقه می‌رسد، نیاز است اطلاعات dept1.Employees را دریافت کند. در اینجا است که کوئری دوم، به بانک اطلاعاتی صادر خواهد شد (بارگذاری در صورت نیاز).

ج) بررسی Eager Loading یا واکشی حریصانه

حالت lazy loading بسیار جذاب به نظر می‌رسد؛ برای مثال می‌توان خواص حجیم یک جدول را به جدول مرتبط دیگری منتقل کرد. مثلا فیلدهای متنی طولانی یا اطلاعات باینری فایل‌های ذخیره شده، تصاویر و امثال آن. به این ترتیب تا زمانیکه نیازی به اینگونه اطلاعات نباشد، lazy loading از بارگذاری آن‌ها جلوگیری کرده و سبب افزایش کارآیی برنامه می‌شود.
اما ... همین lazy loading در صورت استفاده نا آگاهانه می‌تواند سرور بانک اطلاعاتی را در یک برنامه چندکاربره از پا درآورد! نیازی هم نیست تا شخصی به سایت شما حمله کند. مهاجم اصلی همان برنامه نویس کم اطلاع است!
اینبار مثال زیر را درنظر بگیرید که بجای دریافت اطلاعات یک شخص، مثلا قصد داریم، اطلاعات کلیه دپارتمان‌ها را توسط یک Grid نمایش دهیم (فرقی نمی‌کند برنامه وب یا ویندوز باشد؛ اصول یکی است):

```
using (var db = new Sample06Context())
{
      foreach (var dept in db.Departments)
      {
           Console.WriteLine(dept.Name);
           foreach (var item in dept.Employees)
           {
                Console.WriteLine(item.FirstName);
            }
        }
}
```

یک نکته: اگر سعی کنیم کد فوق را اجرا کنیم به خطای زیر برخواهیم خورد:

```
There is already an open DataReader associated with this Command which must be closed first
```

برای رفع این مشکل نیاز است گزینه MultipleActiveResultSets=True را به کانکشن استرینگ اضافه کرد:

```
<connectionStrings>
    <clear/>
    <add
       name="Sample06Context"
       connectionString="Data Source=(local);Initial Catalog=testdb2012;Integrated Security =
true;MultipleActiveResultSets=True;"
       providerName="System.Data.SqlClient"
      />
</connectionStrings>
```

سؤال: به نظر شما در دو حلقه تو در توی فوق چندبار رفت و برگشت به بانک اطلاعاتی صورت می‌گیرد؟ با توجه به اینکه در متد Seed ذکر شده در ابتدای مطلب، تعداد رکوردها مشخص است.

پاسخ: 7 بار!

Object context #15

Statements | Object context Usage

Short SQL	Row Count	Duration	Alerts
SELECT ... FROM [dbo].[Departments] AS [Extent1]	6	27 ms / 891 ms	
SELECT ... FROM [dbo].[Employees] AS [Extent1] WHERE...	2	63 ms / 94 ms	
SELECT ... FROM [dbo].[Employees] AS [Extent1] WHERE...	1	44 ms / 47 ms	
SELECT ... FROM [dbo].[Employees] AS [Extent1] WHERE...	1	294 ms / 297 ms	
SELECT ... FROM [dbo].[Employees] AS [Extent1] WHERE...	2	38 ms / 47 ms	
SELECT ... FROM [dbo].[Employees] AS [Extent1] WHERE...	1	162 ms / 172 ms	
SELECT ... FROM [dbo].[Employees] AS [Extent1] WHERE...	1	209 ms / 203 ms	

Details | Alerts | Stack Trace

```
1 SELECT [Extent1].[EmployeeId]              AS [EmployeeId],
2        [Extent1].[FirstName]               AS [FirstName],
3        [Extent1].[LastName]                AS [LastName],
4        [Extent1].[Department_DepartmentId] AS [Department_DepartmentId]
5 FROM   [dbo].[Employees] AS [Extent1]
6 WHERE  ([Extent1].[Department_DepartmentId] IS NOT NULL)
7        AND ([Extent1].[Department_DepartmentId] = 21 /* @EntityKeyValue1
```

Param	Value
@EntityK	21

و اینجا است که عنوان شد استفاده از EF Profiler در حین توسعه برنامه‌های مبتنی بر ORM «الزامی» است! اگر از این نکته اطلاعی نداشتید، بهتر است یکبار تمام صفحات گزارش‌گیری برنامه‌های خود را که حاوی یک Grid هستند، توسط EF Profiler بررسی کنید. اگر در این برنامه پیغام خطای n+1 select را دریافت کردید، یعنی در حال استفاده ناصحیح از امکانات lazy loading می‌باشید.

آیا می‌توان این وضعیت را بهبود بخشید؟ زمانیکه کار ما گزارشگیری از اطلاعات با تعداد رکوردهای بالا است، استفاده ناصحیح از ویژگی Lazy loading می‌تواند به شدت کارآیی بانک اطلاعاتی را پایین بیاورد. برای حل این مساله در زمان‌های قدیم (!) بین جداول join می‌نوشتند؛ الان چطور؟

در EF متدی به نام Include جهت Eager loading اطلاعات موجودیت‌های مرتبط به هم درنظر گرفته شده است که در پشت صحنه همینکار را انجام می‌دهد:

```
using (var db = new Sample06Context())
{
      foreach (var dept in db.Departments.Include(x => x.Employees))
      {
           Console.WriteLine(dept.Name);
           foreach (var item in dept.Employees)
           {
              Console.WriteLine(item.FirstName);
           }
       }
}
```

همانطور که ملاحظه می‌کنید اینبار به کمک متد Include، نسبت به واکشی حریصانه Employees اقدام کرده‌ایم. اکنون اگر برنامه را اجرا کنیم، فقط یک رفت و برگشت به بانک اطلاعاتی انجام خواهد شد و کار Join نویسی به صورت خودکار توسط EF مدیریت می‌گردد:

```
SELECT [Project1].[DepartmentId]            AS [DepartmentId],
       [Project1].[Name]                    AS [Name],
       [Project1].[C1]                      AS [C1],
       [Project1].[EmployeeId]              AS [EmployeeId],
       [Project1].[FirstName]               AS [FirstName],
       [Project1].[LastName]                AS [LastName],
       [Project1].[Department_DepartmentId] AS [Department_DepartmentId]
FROM   (SELECT [Extent1].[DepartmentId]            AS [DepartmentId],
               [Extent1].[Name]                    AS [Name],
               [Extent2].[EmployeeId]              AS [EmployeeId],
               [Extent2].[FirstName]               AS [FirstName],
               [Extent2].[LastName]                AS [LastName],
               [Extent2].[Department_DepartmentId] AS [Department_DepartmentId],
               CASE
                 WHEN ([Extent2].[EmployeeId] IS NULL) THEN CAST(NULL AS int)
                 ELSE 1
               END                                 AS [C1]
        FROM   [dbo].[Departments] AS [Extent1]
               LEFT OUTER JOIN [dbo].[Employees] AS [Extent2]
                 ON [Extent1].[DepartmentId] = [Extent2].[Department_DepartmentId]) AS [Project1]
ORDER  BY [Project1].[DepartmentId] ASC,
          [Project1].[C1] ASC
```

متد Include در نگارش‌های اخیر EF پیشرفت کرده است و همانند مثال فوق، امکان کار با lambda expressions را جهت تعریف خواص مورد نظر به صورت strongly typed ارائه می‌دهد. در نگارش‌های قبلی این متد، تنها امکان استفاده از رشته‌ها برای معرفی خواص وجود داشت.

همچنین توسط متد Include امکان eager loading چندین سطح با هم نیز وجود دارد؛ مثلا x.Employees.Kids و همانند آن.

چند نکته در مورد نحوه خاموش کردن Lazy loading

امکان خاموش کردن Lazy loading در تمام کلاس‌های برنامه با تنظیم خاصیت Configuration.LazyLoadingEnabled کلاس Context برنامه به نحو زیر میسر است:

```
public class Sample06Context : DbContext
{
        public Sample06Context()
        {
            this.Configuration.LazyLoadingEnabled = false;
        }
```

یا اگر تنها در مورد یک کلاس نیاز است این خاموش سازی صورت گیرد، کلمه کلیدی virtual را حذف کنید. برای مثال با نوشتن public ICollection<Employee> Employees بجای public virtual ICollection<Employee> Employees در اولین بار وهله سازی کلاس دپارتمان، لیست کارمندان آن به نال تنظیم می‌شود. البته در این حالت null object pattern را نیز فراموش نکنید (وهله سازی پیش فرض Employees در سازنده کلاس):

```
public class Department
{
     public int DepartmentId { get; set; }
```

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

    public  ICollection<Employee> Employees { get; set; }
    public Department()
    {
        Employees = new HashSet<Employee>();
    }
}
```

به این ترتیب به خطای null reference object بر نخواهیم خورد. همچنین وهله سازی، با مقدار دهی لیست دریافتی از بانک اطلاعاتی متفاوت است. در اینجا نیز باید از متد Include استفاده کرد.

بنابراین در صورت خاموش کردن lazy loading، حتما نیاز است از متد Include استفاده شود. اگرlazy loading فعال است، جهت تبدیل آن به eager loading از متد Include استفاده کنید (اما اجباری نیست).

نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۲/۲۳ ۱۸:۲۶:۲۷

باسلام.متشکرم از مقالات عالی شما. Code First با آموزش شما کاملاً لذت بخشه. همیشه سلامت باشید. یاعلی

نویسنده: mohammad
تاریخ: ۱۳۹۱/۰۲/۲۳ ۱۸:۵۹:۴۱

استاد این Alert ها (دایره های خاکستری رنگ) هم خیلی مهم هستند؟ آلرتش هم Unbound result set هست که تقریباً برای بیشتر Entity هام همینه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۳ ۱۹:۵۶:۲۲

بستگی به تعداد رکورد دارد. اگر کم است مهم نیست. اگر زیاد است می‌تونه به مصرف بالای حافظه سرور منتهی بشه. این مورد رو میشه با متد الحاقی take کنترل کرد که ترجمه می‌شود به select top n.

نویسنده: Naser Tahery
تاریخ: ۱۳۹۱/۰۲/۲۴ ۱۹:۰۴:۱۳

لایک و رای جوابگوی تشکر از شما نیست
بسیار ممنون

نویسنده: بهزاد
تاریخ: ۱۳۹۱/۰۴/۱۸ ۱۸:۴۲

سلام
ابزار efprof ابزار بسیار خوبی است ولی متاسفانه ما در ایران مشکل خرید داریم و مجبوریم از کرک استفاده کنیم، آیا کرک این نرم افزار هم موجود است؟
بنده نتوانستم چیزی پیدا کنم، ممنون میشم اگر می‌تونین کمک کنین

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۸ ۱۸:۵۵

نیازی به کرک ندارد. در متن توضیح دادم. ایمیل خودتون رو در سایت آن وارد کرده و یک مجوز یک ماهه دریافت کنید. این مجوز رسمی، دسترسی کامل استفاده از برنامه رو به شما می‌ده.

نویسنده: فربد صالحی
تاریخ: ۱۳۹۱/۰۴/۱۹ ۸:۵۵

در رابطه با lazy loading سئوالی داشتم. در روش db first ، خود به خود navigation propertyها در مدل ساخته میشه.از اونجایی که lazy loading به طور پیش فرض فعال هست ، اینطور که شما اینجا توضیح دادید هیچکدام از navigation propertyها به جداول موردنظر رجوع نمی‌کنند. اگه تا اینجا رو درست گفته باشم سئوال اصلی من اینه:
وقتی جداول بزرگ باشند و تعداد navigation propertyها زیاد، مخصوصا وقتی مراجعه به یک جدول چندبار اتفاق بیفتد (مثلا فیلدهایی مثل InsertUserId و DeleteUserId داشته باشیم که هردو به جدول user مراجعه می‌کنند) EF نام‌های نامناسبی تولید میکنه که هنگام استفاده نمیشه تشخیص داد کدوم یکی مثلا به InsertUserId و کدوم یکی به DeleteUserId مربوط میشه. اگر هم بخوایم دستی نامگذاری‌ها رو تغییر بدیم، علاوه بر وقتگیر بودن، با هربار تغییر مدل، دوباره باید اینکار رو تکرار کنیم.
راه حلی که به ذهن من میرسه اینه که توی یه partial class، یه همچین property هایی اضافه کنم.(کد زیر) در واقع موقع

نمایش در گرید، از InsertUsername به عنوان نام کاربری درج کننده استفاده می‌کنم. امیدوارم تونسته باشم درست توضیح بدم. می‌خوام بدونم این روش تا چه حد درسته.

```
public string InsertUsername
{
    get { return DB.Users.Where(x=>x.Id == InsertUserId).Select(x=>x.Username).FirstOr Default(); }
```

```
private set {}
```

```
};
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۹ ۹:۱۰

- شما می‌تونید با استفاده از fluent api کنترل کاملی بر روی نام‌های خودکار تولیدی داشته باشید. یک سری پیش فرض در ابتدای امر هست؛ اما تمام این‌ها با fluent api قابل بازنویسی است.
- اینکه چه نامی در بانک اطلاعاتی تولید شده در EF Code first اهمیتی ندارد. شما با اشیاء سروکار دارید. قرار نیست مستقیما از فیلدی کوئری بگیرید یا قرار نیست مستقیما SQL خام بنویسید. زمانیکه از LINQ استفاده می‌کنید تمام ترجمه‌ها خودکار است صرفنظر از اینکه نام‌ها در سمت دیتابیس الان چه چیزی هست.
- تمام navigation propertyها به جداول مورد نظر مراجعه می‌کنند. lazy loading به معنای عدم بارگذاری اطلاعات اشیاء مرتبط در بار اول فراخوانی شیء پایه است و تنها بارگذاری اطلاعات اشیاء وابسته در زمان نیاز. دقیقا در زمانیکه خاصیتی از آن شیء مرتبط فراخوانی شود و نه قبل از آن.
- زمانیکه primary key یک جدول رو دارید بهتر است از متد Find استفاده کنید بجای کوئری LINQ فوق. به این ترتیب از سطح اول کش برخوردار خواهید شد (تعداد کمتر رفت و برگشت به بانک اطلاعاتی).
- شما بدون مشکل می‌تونید مستقیما از خواص اشیاء مرتبط استفاده کنید و اگر می‌خواهید lazy loading را متوقف کنید (خصوصا برای نمایش اطلاعات در یک گرید) فقط کافی است از متد Include یاد شده استفاده کنید.

نویسنده: فربد صالحی
تاریخ: ۱۳۹۱/۰۴/۱۹ ۹:۵۲

1- اهمیت نام تولید شده اونجاست که توی navigation property ، برای insertUserId و deleteUserId مقادیر user1 و user2 تولید میشه و به فرض وقتی بخوایم نام کاربر درج کننده (user1.username) را نمایش بدیم، ممکنه به اشتباه نام کاربر حذف کننده (user2.username)
رو نمایش بدیم. چون user1 و user2 نامهای شفافی نیستن.
2- fluent api که فرمودین رو بررسی می‌کنم، ولی راستش متوجه نشدم که روشی که استفاده کردم درست هست یا نه. در واقع به جای استفاده از navigation propertyهای خودکار، خودم با اضافه کردن یه property مثلا با نام InsertUsername، نام کاربر درج کننده رو برمیگردونم. (همون که تو کد کامنت قبلی نوشتم).
اینجا هم به نظرم تا موقعی که از InsertUsername استفاده نشه، مراجعه به دیتابیس انجام نمیشه، درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۰:۱۶

- روش دیگر تعیین نام صریح کلیدهای خارجی تشکیل شده در سمت دیتابیس در EF Code first استفاده از ویژگی ForeignKey بر روی یک navigation property است. در قسمت سوم این سری به آن اشاره شده است (مورد هشتم متادیتای بررسی شده در آن).
- بله. ولی اگر در این حالت اطلاعات شما در یک گرید نمایش داده شود n هزار بار رفت و برگشت به بانک اطلاعاتی خواهید داشت. در مبحث جاری به آن با ذکر ریزجزئیات و روش حل خاص آن، پرداخته شده است.

نویسنده: فربد صالحی
تاریخ: ۱۳۹۱/۰۴/۲۰ ۹:۲۵

هنگام استفاده از EF 4 بوسیله WCF، ما خطایی در یافت می‌کنیم با این عنوان:

```
underlying connection was closed. the connection was closed unexpectedly
```

جستجو شد و جاهای مختلف اینطور گفته شده که در هنگام استفاده از EF با WCF، باید lazy loading غیرفعال شه ، در غیر اینصورت حلقه ایجاد میشه! مثلا اینجا
حالا این سوال پیش میاد که علت این مسال چیه، آیا راه دیگه ای وجود نداره. و مهمتر اینکه غیر فعال کردن lazy loading کارایی برنامه رو پایین نمیاره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۰ ۹:۵۵

دنیای WCF ایی که از طریق وب در دسترس است، یک دنیای اصطلاحا detached است. در این حالت زمانیکه ctx.Bills.ToList را فراخوانی می‌کنید، لیست صورتحساب‌ها از سرور دریافت شده و اتصال خاتمه پیدا می‌کند. اینجا دیگر lazy loading معنایی ندارد چون context جاری در سمت سرور بسته شده.
شما زمانی می‌تونید از lazy loading برای بارگذاری اشیاء مرتبط مانند حلقه زیر استفاده کنید:

```
foreach (var dept in db.Departments)
{
     Console.WriteLine(dept.Name);
      foreach (var item in dept.Employees)
      {
           Console.WriteLine(item.FirstName);
       }
}
```

که در یک context و در یک اتصال باز به سرور قرار داشته باشید. در این حالت EF تمام اتصالات و رفت و برگشت‌های مورد نیاز را بدون کوئری نوشتن خاصی مدیریت می‌کند.
در WCF یکبار اطلاعات serialize شده و اتصال بسته می‌شود (البته WCF فراتر است از حالت http binding ساده؛ ولی عموما این مورد در برنامه‌های وب مدنظر است). بنابراین اینبار اگر dept.Employees را روی لیست تهیه شده فراخوانی کنید، پیغام بسته بودن اتصال رو دریافت می‌کنید. به همین جهت اگر نیاز به اطلاعات کارمندان هم هست، همه را باید به یکباره از سرور دریافت کرد.

نویسنده: فربد صالحی
تاریخ: ۱۳۹۱/۰۴/۲۰ ۱۰:۵

متشکر از پاسخ شما. پس میشه اینطور نتیجه گرفت که :

موقع استفاده از WCF، با غیرفعال کردن lazy loading فقط entityهای اصلی بارگذاری می‌شوند و تاثیری روی کارایی برنامه نداره. و در مثال شما ، اگه بخوایم به dept.Employees دسترسی داشته باشیم، باید صریحا اونها رو دریافت کرد.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۷/۰۱ ۱۰:۲۴

آقای نصیری سلام.
روشی برای خرید یا استفاده دائمی از Ef profiler نیست؟ من تابحال چندین ایمیل ساختم و اونو دانلود کردم تا برام کار کنه.
ممنون میشم اگه راهنمایی کنید. یاعلی.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۱ ۱۰:۴۲

- سازنده Ef profiler یک اسرائیلی است (همان سازنده RavenDB). من بعید می‌دونم بتونید خرید کنید.
- پیشنهاد من استفاده از mini-profiler سورس باز است که با EF Code first هم کار می‌کند.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۷/۰۱ ۱۰:۵۵

متشکرم. فراوان

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۸/۰۴ ۱۰:۳۷

آیا include کار join را انجام می‌دهد؟چه تفاوتی بین include و join وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۴ ۱۱:۱۵

در EF یک سری متد وجود دارند که حجم کوئری‌های LINQ نوشته شده را کاهش می‌دهند. یک نمونه آن Include است، نمونه دیگر استفاده از خواص راهبری است: (^ و ^)

نویسنده: bluesky
تاریخ: ۱۳۹۱/۱۰/۲۱ ۱۵:۳۶

DynamicProxies چیه دقیقا؟ و چرا استفاده می‌شه توسط ef؟ (منظور اینکه چرا در runtime بجای یک آبجکت واقعی از کلاس مورد نظرم با یک dynamicProxy از اون کلاس روبرو می‌شم؟! چرا این رو هم ef نبرده پشت صحنه؟)
من بعضی جاهای برنامم که می‌خوام ابجکتی رو cast کنم بخاطر اینکه اون ابجکت یک dynamicProxy هست (در واقع یک wrapper روی کلاس مورد نظرم هست) نمی‌تونم و باید underlying type اش رو بگیرم. و اینکه دقیقا نمی‌دونم (در runtime) جایی که بخوام cast انجام بدم آیا با یک dynamicProxy روبرو هستم یا آبجکتی که مد نظرم هست!

با استفاده از DbContext.Configuration.ProxyCreationEnabled=false می‌شه غیرفعالش کرد. ولی نمی‌دونم چه side effect هایی داره! و کجاها چه تغییری می‌کنه!

می تونین کمکم کنین آقای نصیری و سایر دوستان؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۱ ۱۸:۳۸

در تمام ORMها استفاده میشه. NHibernate هم نوع خاص خودش را دارد. در EF از کلاس‌های پروکسی به دو منظور Lazy loading و change tracking استفاده میشه. به همین جهت خواصی که در lazy loading مورد استفاده قرار می‌گیرند باید virtual تعریف شوند. به این ترتیب ORM مورد استفاده می‌تونه این خواص رو جهت کاربردهای خودش، به صورت پویا بازنویسی و override کنه. با تنظیم DbContext.Configuration.ProxyCreationEnabled=false ، دو کاربرد یاد شده از کار خواهند افتاد. اطلاعات بیشتر رو می‌تونید در بلاگ یکی از اعضای تیم EF بخونید .

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۱:۲۵

وقتی کلاینت Java Script ای یا Silverlight ای در سمت کلاینت داره Change Tracking رو انجام می‌ده و برای Update داره Current و Original رو با هم برام ارسال می‌کنه، و تغییرات سمت سرور واقعا ناچیزه، واقعا می‌طلبه که نه تنها Proxy Creation رو غیر فعال کرد، که بنده حتی Automatic Change Tracking رو هم غیر فعال می‌کنم، فقط در Override کردن Save Changes در DbContext یک بار دستی
Change Tracker.Detect Changes رو فراخونی می‌کنم
موقع Load کردن اطلاعات برای ارسال به سمت کلاینت نیز همیشه از As No Tracking استفاده می‌کنم، و واقعا تا این لحظه به هیچ وجه حس نکردم که چیزی رو از دست دادم، Lazy Loading هم واقعا آیتم حیاتی ای نیست.

ولی از اون طرف من نه ساخته شدن Proxy رو دارم، نه فراخونی Detect Changes رو به صورت پشت صحنه ای در اکثر متدهای EF و نه سربار ساخته شدن Entryها هنگام Load (البته در بازگشت از کلاینت به سرور، Attach می‌کنم، که راه در رویی هم نداره)
چون این موارد همه در Repository قرار داده شدند، عملا کدنویسی خودم رو هم تحت الشعاع قرار ندادم
به همه افراد توصیه می‌کنم که این کار رو انجام بدند

نویسنده: bluesky
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۳:۳۳

بله مرسی اینا رو و چنجای دیگه رو هم خوندم مطالب رو اما مشکلم رو نتونستم براش راه حل اصولی پیدا کنم متاسفانه
مشکل اینه که من در runtime در بعضی شرایط وقتی entity خودم رو (چون بصورت ارث بری کار شده یک entity پایه هست که فقط یک Id داره) می خوام (مثلا) cast کنم به یک entity دیگه (که می‌دونم نوعش دقیقا چیه) ، type اون، **DynamicProxy** هست بجای اینکه از نوع مورد نظر من که انتظارشو دارم باشه (گرچه underlyingType اون همون نوعی هست که مد نظر بنده می‌باشد) اما مشکل بدتره زمانی که من نمی‌دونم **دقیقا کِی** نوع این entity از جنس DynamicProxyهای ef هست و کی از جنس مورد نظر خودم *(احتمالا پشت صحنه ef وقتی که entity بنده درگیر سیستم changetracking می‌شه ef براش همچین wrapper ای می‌سازه تا کارای خودشو بکنه ولی کاش این dynamicProxy رو هم مثل خیلی چیزای دیگه تو این DbContextApi می‌بردن پشت صحنه تا زندگیمونو بکنیم:( )*

امیدوارم منظورم رو رسونده باشم
مثلا من کلاس پایه‌ی زیر رو دارم برای تمام موجودیت هام:

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

و یک کلاس معمولی (که در واقع جدولی در بانک هست مثلا:

```
public class Student : MyBaseEntity
{
  public string Name {get;set;}
 //...
}
```

حالا من یک کلاس جنریک دارم مثل زیر (که صرفا جهت ساده سازی سناریوی مورد اشاره اینجوری نوشتمش و وجود خارجی نداره) :

```
public class SomeGenericWorker <TEntity> where TEntity : BaseEntity
{
 //...

  public void DoSth(TEntity entity)
{
  if (entity is Student)
```

```
{
  // ...
}

}

}
```

مشخصه که من تو تابع DoSth می‌خوام چیکار کنم. حالا مشکل اینه که در بعضی جاها این روال درسته (یعنی type موجودیت entity ، واقعا Student هست) اما بعضی شرایط type موجودیت هام از جنس **DynamicProxies.Student_23323123124546454576646** هستن و باید **underlyingType** شون رو بگیرم اونوقت می‌تونم با نوع مد نظر خودم کار کنم.

در برنامه ای که نوشتم همه چی درست کار می‌کنه و برنامه داره کارشو می‌کنه و فعلا برای مشکل بالا که عرض کردم من راه حل درستی پیدا نکردم و خیلی دست و پا شکسته کار کردم جاهایی که اون مسئله وجود داره واسه همین دنبال جواب درست می‌گردم کماکان..

در ضمن با disable کردن امکان DynamicProxy در Config مربوط به DbContext خودم، از مزایای خوبی مثل ChangeTracking , LazyLoading بی بهره می‌شم و اصلا جالب نیست. پس باید این گزینه true باشه. اما می‌خوام که type واقعی یک entity رو هر لحظه سرراست بتونم بگیرم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۳:۴۱

از متد ObjectContext.GetObjectType استفاده کنید. برای نمونه در پیاده سازی سطح دوم کش ازش استفاده شده:

```
var changedEntityNames = ctx.ChangeTracker
                                    .Entries()
                                    .Where(x => x.State == EntityState.Added ||
                                                x.State == EntityState.Modified ||
                                                x.State == EntityState.Deleted)
                                    .Select(x =>
ObjectContext.GetObjectType(x.Entity.GetType()).FullName)
                                    .Distinct()
                                    .ToList();
```

نویسنده: bluesky
تاریخ: ۱۳۹۱/۱۰/۲۱ ۲۳:۵۱

مرسی
می رم رو این که گفتین کار کنم
با تشکر

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۱۰/۲۲ ۱۰:۴۱

دوست عزیز، من که Proxy Creation رو غیر فعال کردم، و اضافه بر اون Automatic Change Tracking رو هم غیر فعال کردم، با سناریویی که گفتم کماکان Change Tracking رو دارم، و فکر کنم صحبت من رو شما اشتباه متوجه شدید، مگه می‌شه آدم Change Tracking رو کاملا بذاره کنار، من فقط روش رو عوض کردم

مسلما تنها راه Change Tracking با استفاده از Dynamic Proxy نیست، در NHibernate هم من به جای استفاده از Dynamic Proxy اومدم از IL Injection استفاده کردم با استفاده از Post Sharp، چون تو اون برنامه واقعا تغییرات سمت سرور زیاد بود و تغییرات به غیر از زمان Save نیز به صورت آنی نیاز بود.

برای درک این که چرا من این کارها رو انجام می‌دم، به این صفحه بروید و شماره 3.1.1 را مطالعه کنید.

با این حال، با فرض این که شما بنا به هر علتی، Dynamic Proxy رو بخواهید، **کدی که اینجا نوشتید** باید در هر حالتی کار کنه، چون در هر حال اون Dynamic Proxy از Student ارث بری کرده، پس is شما True بر می‌گردونه. اگه باز کدی رو که کار نمی‌کنه رو اینجا بنویسید، شاید مشکل چیز دیگه ای هستش Program.cs

فایلی که دارد کار می‌کند و حاوی شرط شما است.
علاوه بر این من یک Extension Method نوشتم، که Real Type مد نظر شما رو بر می‌گردونه، حال چه تو NHibernate، چه تو Entity Framework، چه هر جای دیگه ای

```
public static class ReflectionExt
    {
        public static Type GetRealType(this Type type)
        {
            if (Object.ReferenceEquals(type, null))
                throw new ArgumentNullException("type");

            if (type.Assembly.IsDynamic)
                return GetRealType(type.BaseType);
            else
                return type;
        }
    }
```

نویسنده: نوید
تاریخ: ۱۳۹۱/۱۲/۰۹ ۱۳:۴

سلام
من در حین استفاده از EF-Profiler با خطای Exception has been thrown by the target of an invocation مواجه میشم.
EF-profiler رو بعد از دانلود کردن اجرا میکنم، Dll مربوطه رو به برنامه اضافه کردم و برنامه رو Run میکنم . این خطا روی خط

```
HibernatingRhinos.Profiler.Appender.EntityFramework.EntityFrameworkProfiler.Initialize();
```

اتفاق می‌افتد.
روی کد پروژه خودتون هم تست کردم ، همین خطا رو میداد.
ممنون میشم راهنماییم کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۰۹ ۱۳:۴۵

- بله. همینطوره. به همین جهت یک قسمت در کد فوق کامنت شده نوشته شده:

```
// note: remove this line if you received : CreateDatabase is not supported by the provider.
```

EF Prof با سیستم به روز رسانی بانک اطلاعاتی EF Code first تداخل ایجاد می‌کنه. اول دیتابیس رو به روز کنید. بعد تنظیمات EF Prof رو اضافه کنید و بعد هم آغاز دیتابیس رو با null مقدار دهی کنید.

- ضمنا گروه مرتبط با EF Prof و محصولات مشابه اینجا است:
http://groups.google.com/group/efprof
بهتر است این نوع مشکلات را با خودشون مطرح کنید.

نویسنده: محمود داوری
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۷:۲۹

با سلام
من مشکلی که با متد Include معرفی شده دارم عدم شناخت اون داخل کدهاست.
من از EF5 و NET 4.5. استفاده میکنم ولی زمان استفاده از این متد ، به رنگ قرمز درمیاد و البته هیچ خطایی توسط ویژوال استدیو هم دریافت نمیکنم و تعجب اینکه به خوبی هم کار میکنه. ولی به اصطلاح باید مانند نوشتن در یک فایل txt کار کنم چون هیچ Intellisense وجود نداره.

در تصویر بهتر میتونید ببینید :

نویسنده: محمود داوری
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۷:۳۶

البته فضای نام System.Data.Entity به رنگ خاکستری در اومده ، یعنی عملا هیچ استفاده ای از اون نمیشه.
آیا این فضای نام ارتباطی با ورژن EF و NET 4.5. نداره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۸:۸

- فضای نام System.Data.Entity مربوط به متد الحاقی جدید Include، در اسمبلی EntityFramework.dll وجود دارد. بنابراین برای استفاده از آن نیاز است اسمبلی EntityFramework.dll را هم به پروژه کتابخانه‌ای جدید خود، الحاق کنید.
- اگر مورد فوق برقرار است و Intellisense کار نمی‌کند، اما برنامه کامپایل می‌شود، احتمالا مشکل از ReSharper ایی است که دارید استفاده می‌کنید. VS.NET را با دستور خط فرمان ذیل، در حالت safe mode اجرا کنید:

```
"c:\__path to__\IDE\devenv.exe" /safemode /nosplash /log
```

در این حالت افزونه‌ها بارگذاری نمی‌شوند. اگر مشکلی نبود، یعنی باید ReSharper را به روز یا مجددا نصب کنید.

نویسنده: محمود داوری
تاریخ: ۱۳۹۲/۰۵/۲۰ ۱۸:۲۱

بسیار ممنون. همینطور که شما فرمودید بدون بارگزاری افزونه‌ها کار کرد و مشکلی نبود.

نویسنده: ناظم
تاریخ: ۱۳۹۲/۱۱/۰۳ ۱۲:۲۳

سلام
من EF profiler رو از طریق NuGet نصب کردم ، اسمبلی‌ها به برنامه الحاق شد،تابع استارت اون به صورت خودکار به برنامه اضافه شد... همه درست. ولی وقتی برنامه رو اجرا میکنم هیچ اتفاقی نمیوفته و EF Profiler هیچ چیزی رو نمیتونه لاگ کنه. خیلی در مورد این مشکل گشتم ولی چیزی پیدا نمیکنم در ضمن وقتی EF Profiler رو قبل از ایجاد بانک اطلاعاتی به برنامه اضافه میکنم

خطای زیر رو میده.

```
An unhandled exception of type 'System.NotSupportedException' occurred in EntityFramework.dll

Additional information: Unable to determine the DbProviderFactory type for connection of type
'System.Data.SqlClient.SqlConnection'. Make sure that the ADO.NET provider is installed or registered
in the application config.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۳ ۱۳:۲۴

- در EF 6 نیاز است یک سری تعاریف را به فایل کانفیگ اضافه کنید : ارتقاء به EF 6 و استفاده از EF 6
- در EF 6 اصلا نیازی به پروفایلر خاصی ندارید : نحوه‌ی لاگ کردن توکار با EF 6
- برنامه‌های دیگری هم برای لاگ کردن وجود دارند (سورس باز و رایگان). مثلا mini profiler

نویسنده: ناظم
تاریخ: ۱۳۹۲/۱۱/۰۵ ۸:۵۴

سلام؛ من از sql server 2008 استفاده میکنم(روی یک سرور دیگر هست)، از اول هم EF6 رو نصب کردم.
باز با این حال همین خطا پا برجاست، همچنین چیزی که متوجه نمیشم اینکه وقتی EFProfiler رو از پروژه حذف میکنم (حذف App_Start.EntityFrameworkProfilerBootstrapper.PreStart();) این خطا رفع میشه.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۰۵ ۱۳:۳۳

به نظر EF Profiler با نگارش 6 مشکل داره . بهتره از روش‌های دیگری که گفتند مثل mini profiler استفاده کنید.

عنوان: **روش‌هایی برای بهبود سرعت برنامه‌های مبتنی بر Entity framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۴ ۱۱:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در این مطلب تعدادی از شایع‌ترین مشکلات حین کار با Entity framework که نهایتا به تولید برنامه‌هایی کند منجر می‌شوند، بررسی خواهند شد.

مدل مورد بررسی

```
public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<BlogPost> BlogPosts { get; set; }
    }

    public class BlogPost
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

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

کوئری‌هایی که در ادامه بررسی خواهند شد، بر روی رابطه‌ی one-to-many فوق تعریف شده‌اند؛ یک کاربر به همراه تعدادی مطلب منتشر شده.

مشکل 1: بارگذاری تعداد زیادی ردیف

```
var data = context.BlogPosts.ToList();
```

در بسیاری از اوقات، در برنامه‌های خود تنها نیاز به مشاهده‌ی قسمت خاصی از یک سری از اطلاعات، وجود دارند. به همین جهت بکارگیری متد ToList بدون محدود سازی تعداد ردیف‌های بازگشت داده شده، سبب بالا رفتن مصرف حافظه‌ی سرور و همچنین بالا رفتن میزان داده‌ای که هر بار باید بین سرور و کلاینت منتقل شوند، خواهد شد. یک چنین برنامه‌هایی بسیار مستعد به استثناهایی از نوع out of memory هستند.
راه حل: با استفاده از Skip و Take، مباحث صفحه‌ی بندی را اعمال کنید.

مشکل 2: بازگرداندن تعداد زیادی ستون

```
var data = context.BlogPosts.ToList();
```

فرض کنید View برنامه، در حال نمایش عناوین مطالب ارسالی است. کوئری فوق، علاوه بر عناوین، شامل تمام خواص تعریف شده‌ی دیگر نیز هست. یک چنین کوئری‌هایی نیز هربار سبب هدر رفتن منابع سرور می‌شوند.
راه حل: اگر تنها نیاز به خاصیت Content است، از Select و سپس ToList استفاده کنید؛ البته به همراه نکته 1.

```
var list = context.BlogPosts.Select(x => x.Content).Skip(15).Take(15).ToList();
```

مشکل 3: گزارشگیری‌هایی که بی‌شباهت به حمله‌ی به دیتابیس نیستند

```
foreach (var post in context.BlogPosts)
{
    Console.WriteLine(post.User.Name);
}
```

فرض کنید قرار است رکوردهای مطالب را نمایش دهید. در حین نمایش این مطالب، در قسمتی از آن باید نام نویسنده نیز درج شود. با توجه به رابطه‌ی تعریف شده، نوشتن post.User.Name به ازای هر مطلب، بسیار ساده به نظر می‌رسد و بدون مشکل هم کار می‌کند. اما ... اگر خروجی SQL این گزارش را مشاهده کنیم، به ازای هر ردیف نمایش داده شده، یکبار رفت و برگشت به بانک اطلاعاتی، جهت دریافت نام نویسنده یک مطلب وجود دارد.
این مورد به lazy loading مشهور است و در مواردی که قرار است با یک مطلب و یک نویسنده کار شود، شاید اهمیتی نداشته باشد. اما در حین نمایش لیستی از اطلاعات، بی‌شباهت به یک حمله‌ی شدید به بانک اطلاعاتی نیست.
راه حل: در گزارشگیری‌ها اگر نیاز به نمایش اطلاعات روابط یک موجودیت وجود دارد، از متد Include استفاده کنید تا Lazy loading لغو شود.

```
foreach (var post in context.BlogPosts.Include(x=>x.User))
```

مشکل 4: فعال بودن بی‌جهت مباحث ردیابی اطلاعات

```
var data = context.BlogPosts.ToList();
```

در اینجا ما فقط قصد داریم که لیستی از اطلاعات را دریافت و سپس نمایش دهیم. در این بین، هدف، ویرایش یا حذف اطلاعات این لیست نیست. یک چنین کوئری‌هایی مساوی هستند با تشکیل dynamic proxies مخصوص EF جهت ردیابی تغییرات اطلاعات (مباحث AOP توکار). EF توسط این dynamic proxies، محصور کننده‌هایی را برای تک تک آیتم‌های بازگشت داده شده از لیست تهیه می‌کند. در این حالت اگر خاصیتی را تغییر دهید، ابتدا وارد این محصور کننده (غشاء نامرئی) می‌شود، در سیستم ردیابی EF ذخیره شده و سپس به شیء اصلی اعمال می‌گردد. به عبارتی شیء در حال استفاده، هر چند به ظاهر post.User است اما در واقعیت یک User دارای روکشی نامرئی از جنس dynamic proxyهای EF است. تهیه این روکش‌ها، هزینه‌بر هستند؛ چه از لحاظ میزان مصرف حافظه و چه از نظر سرعت کار.
راه حل: در گزاشگیری‌ها، dynamic proxies را توسط متد AsNoTracking غیرفعال کنید:

```
var data = context.BlogPosts.AsNoTracking().Skip(15).Take(15).ToList();
```

مشکل 5: باز کردن تعداد اتصالات زیاد به بانک اطلاعاتی در طول یک درخواست

هر Context دارای اتصال منحصربفرد خود به بانک اطلاعاتی است. اگر در طول یک درخواست، بیش از یک Context مورد استفاده قرار گیرد، بدیهی است به همین تعداد اتصال باز شده به بانک اطلاعاتی، خواهیم داشت. نتیجه‌ی آن فشار بیشتر بر بانک اطلاعاتی و همچنین کاهش سرعت برنامه است؛ از این لحاظ که اتصالات TCP برقرار شده، هزینه‌ی بالایی را به همراه دارند.
روش تشخیص:

```
private void problem5MoreThan1ConnectionPerRequest()
        {
            using (var context = new MyContext())
            {
                var count = context.BlogPosts.ToList();
            }
        }
```

داشتن متدهایی که در آن‌ها کار وهله سازی و dispose زمینه‌ی EF انجام می‌شود (متدهایی که در آن‌ها new Context وجود دارد).

راه حل: برای حل این مساله باید از روش‌های تزریق وابستگی‌ها استفاده کرد. یک Context وهله سازی شده‌ی در طول عمر یک درخواست، باید بین وهله‌های مختلف اشیایی که نیاز به Context دارند، زنده نگه داشته شده و به اشتراک گذاشته شود.

مشکل 6: فرق است بین IList و IEnumerable

```
DataContext = from user in context.Users
                      where user.Id>10
                      select user;
```

خروجی کوئری LINQ نوشته شده از نوع IEnumerable است. در EF، هربار مراجعه‌ی مجدد به یک کوئری که خروجی IEnumerable دارد، مساوی است با ارزیابی مجدد آن کوئری. به عبارتی، یکبار دیگر این کوئری بر روی بانک اطلاعاتی اجرا خواهد شد و رفت و برگشت مجددی صورت می‌گیرد.
زمانیکه در حال تهیه‌ی گزارشی هستید، ابزارهای گزارشگیر ممکن است چندین بار از نتیجه‌ی کوئری شما در حین تهیه‌ی گزارش استفاده کنند. بنابراین برخلاف تصور، data binding انجام شده، تنها یکبار سبب اجرای این کوئری نمی‌شود؛ بسته به ساز و کار درونی گزارشگیر، چندین بار ممکن است این کوئری فراخوانی شود.
راه حل: یک ToList را به انتهای این کوئری اضافه کنید. به این ترتیب از نتیجه‌ی کوئری، بجای اصل کوئری استفاده خواهد شد و در این حالت تنها یکبار رفت و برگشت به بانک اطلاعاتی را شاهد خواهید بود.

مشکل 7: فرق است بین IQueryable و IEnumerable

خروجی IEnumerable، یعنی این عبارت را محاسبه کن. خروجی IQueryable یعنی این عبارت را درنظر داشته باش. اگر نیاز است نتایج کوئری‌ها با هم ترکیب شوند، مثلا بر اساس رابط کاربری برنامه، کاربر بتواند شرط‌های مختلف را با هم ترکیب کند، باید از ترکیب IQueryableها استفاده کرد تا سبب رفت و برگشت اضافی به بانک اطلاعاتی نشویم.

مشکل 8: استفاده از کوئری‌های Like دار

```
var list = context.BlogPosts.Where(x => x.Content.Contains("test"))
```

این نوع کوئری‌ها که در نهایت به Like در SQL ترجمه می‌شوند، سبب full table scan خواهند شد که کارآیی بسیار پایینی دارند. در این نوع موارد توصیه شده‌است که از روش‌های full text search استفاده کنید.

مشکل 9: استفاده از Count بجای Any

اگر نیاز است بررسی کنید مجموعه‌ای دارای مقداری است یا خیر، از Count>0 استفاده نکنید. کارآیی Any و کوئری SQL ایی که تولید می‌کند، به مراتب بیشتر و بهینه‌تر است از Count>0.

مشکل 10: سرعت insert پایین است

ردیابی تغییرات را خاموش کرده و از متد جدید AddRange استفاده کنید. همچنین افزونه‌هایی برای Bulk insert نیز موجود هستند.

مشکل 11: شروع برنامه کند است

می‌توان تمام مباحث نگاشت‌های پویای کلاس‌های برنامه به جداول و روابط بانک اطلاعاتی را به صورت کامپایل شده در برنامه ذخیره کرد . این مورد سبب بالا رفتن سرعت شروع برنامه خصوصا در حالتیکه تعداد جداول بالا است می‌شود.

نظرات خوانندگان

نویسنده: محمد شیران
تاریخ: ۱۳۹۳/۰۴/۰۴ ۱۷:۳۰

مطلب فوق العاده آموزنده ای بود. لطفا در خصوص نحوه استفاده از ساز و کار full text search در EF هم اگه روشی هست توضیح بفرمایید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۴ ۱۷:۳۵

- در کنفرانس techEd 2014 در جلسه « Entity Framework: Building Applications with Entity Framework 6 » کار با Full text search در EF 6، جزو مثال‌های مطرح شده‌است.
- روش دوم هم در اینجا « Full text search in Microsoft's Entity Framework ».
- روش سوم با استفاده از IDbCommandInterceptor در اینجا « Microsoft's Full Text Search in Entity Framework 6 »

نویسنده: فواد عبداللهی
تاریخ: ۱۳۹۳/۰۴/۰۵ ۱۹:۳۳

ممنون از مطلب مفید تون
من با راه حل شماره 5 و استفاده همزمان از addrange یا modify (update) و ... (بجز select, add, delete) همزمان مشکل دارم! اگر ممکنه یه sample در این رابطه معرفی کنید.
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۵ ۲۰:۳۹

```
((DbSet<Category>)_categories).AddRange(...);

// or in the Sample07Context

public IEnumerable<TEntity> AddThisRange<TEntity>(IEnumerable<TEntity> entities) where TEntity : class
{
    return ((DbSet<TEntity>)this.Set<TEntity>()).AddRange(entities);
}
```

نویسنده: ناظم
تاریخ: ۱۳۹۳/۰۵/۲۰ ۱۷:۴۷

سلام؛ **خروجی IEnumerable، یعنی این عبارت را محاسبه کن**
وقتی خروجی query مثلا در نوع IEnumerable ذخیره میشه تا وقتی مورد استفاده قرار نگرفته رفت و برگشتی به بانک صورت نگرفته مثل IQueryable :

```
private IEnumerable<Entity1> ienumerableEntites;
        private IQueryable<Entity1> queryablelEntities;

            ienumerableEntites = context.Entity1.Where(x=>x.EntityID>50);
            queryablelEntities = context.Entity1.Where(x => x.EntityID > 50);
```

منظورتون از جمله بالا چیست؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۳/۰۵/۲۰ ۱۸:۲۹

« بررسی Deferred execution یا بارگذاری به تاخیر افتاده »

نویسنده: Elham
تاریخ: ۱۳۹۳/۰۸/۳۰ ۱۷:۱۹

به نظر شما کوئری‌های پایین رو چطور میشه بهینه نوشت؟

```
List<Stat> allQuestion = (from a in TempClass.Stats
         where a.Person.PersonID == TempClass.ActiveUser.PersonID &&
               a.Subject.SubjectID == tileNumber
         select a).AsParallel().ToList();

int allQuestionCount = allQuestion.Count;

int correctCount = (from a in allQuestion
                    where a.Person.PersonID == TempClass.ActiveUser.PersonID &&
                          a.Subject.SubjectID == tileNumber
                    select a.CorrectQuestionCount).Sum();

int totalTime = (from a in allQuestion
                 where a.Person.PersonID == TempClass.ActiveUser.PersonID &&
                       a.Subject.SubjectID == tileNumber
                 select a.TotalTime).Sum();

double score = (from a in allQuestion
                where a.Person.PersonID == TempClass.ActiveUser.PersonID &&
                      a.Subject.SubjectID == tileNumber
                select a.Score).Sum();
```

50 بار دستورات بالا اجرا میشه و یک مکث حدودا 20 ثانیه‌ای داره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۳۰ ۱۷:۴۹

چندبار رفت و برگشت به بانک اطلاعاتی را می‌شود تبدیل کرد به یکبار رفت و برگشت: « اعمال توابع تجمعی بر روی چند ستون در Entity framework »

نویسنده: علیرضا م
تاریخ: ۱۳۹۳/۰۹/۰۴ ۱۵:۱۳

سلام
ایشان یکبار در ابتدا از ToList استفاده نموده اند. اعمال تجمعی که بعد از کویری اول نوشته شده است در حافظه محاسبه میشود یا در سمت بانک اطلاعاتی؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۰۴ ۱۵:۲۸

- در حافظه.
- ولی در کل روش محاسبه‌ی sum این نیست که رکوردها را به همراه تمام ستون‌های جدول از بانک اطلاعاتی واکشی کرد و بعد در برنامه چند ستون انتخابی آن‌ها را جمع زد؛ زمانیکه خود بانک اطلاعاتی این توانایی را به نحو بهینه‌تری دارد.

عنوان: **نکاتی در مورد استفاده از توابع تجمعی در Entity framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۰:۳۸
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

استفاده از Aggregate functions یا توابع تجمعی چه در زمان SQL نویسی مستقیم و یا در حالت استفاده از LINQ to Entities نیاز به ملاحظات خاصی دارد که عدم رعایت آن‌ها سبب کرش برنامه در زمان موعد خواهد شد. در ادامه تعدادی از این موارد را مرور خواهیم کرد.

ابتدا مدل‌های برنامه را در نظر بگیرید که از یک صورتحساب، به همراه ریز قیمت‌های آیتم‌های مرتبط با آن تشکیل شده است:

```
public class Bill
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<Transaction> Transactions { set; get; }
    }

    public class Transaction
    {
        public int Id { set; get; }
        public DateTime AddDate { set; get; }
        public int Amount { set; get; }

        [ForeignKey("BillId")]
        public virtual Bill Bill { set; get; }
        public int BillId { set; get; }
    }
```

در ادامه این کلاس‌ها را در معرض دید EF Code first قرار می‌دهیم:

```
public class MyContext : DbContext
    {
        public DbSet<Bill> Bills { get; set; }
        public DbSet<Transaction> Transactions { get; set; }
    }
```

همچنین تعدادی رکورد اولیه را نیز جهت انجام آزمایشات به بانک اطلاعاتی متناظر، اضافه خواهیم کرد:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            var bill1 = new Bill { Name = "bill-1" };
            context.Bills.Add(bill1);

            for (int i = 0; i < 11; i++)
            {
                context.Transactions.Add(new Transaction
                {
                    AddDate = DateTime.Now.AddDays(-i),
                    Amount = 1000000000 + i,
                    Bill = bill1
                });
            }
            base.Seed(context);
        }
    }
```

در اینجا به عمد از ارقام بزرگ استفاده شده است تا نمایانگر عملکرد یک سیستم واقعی در طول زمان باشد.

اولین مثال: یک جمع ساده

```
public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var context = new MyContext())
            {
                var sum = context.Transactions.Sum(x => x.Amount);
                Console.WriteLine(sum);
            }
        }
    }
```

ساده‌ترین نیازی را که در اینجا می‌توان مدنظر داشت، جمع کل تراکنش‌های سیستم است. به نظر شما خروجی کوئری فوق چیست؟

خروجی SQL کوئری فوق به نحو زیر است:

```
SELECT
         [GroupBy1].[A1] AS [C1]
         FROM ( SELECT
                    SUM([Extent1].[Amount]) AS [A1]
                    FROM [dbo].[Transactions] AS [Extent1]
                    )  AS [GroupBy1]
```

و خروجی واقعی آن استثنای زیر می‌باشد:

```
Arithmetic overflow error converting expression to data type int.
```

بله. محاسبه ممکن نیست؛ چون جمع حاصل از بازه اعداد صحیح خارج شده است.

راه حل:

نیاز است جمع را بر روی Int64 بجای Int32 انجام دهیم:

```
var sum2 = context.Transactions.Sum(x => (Int64)x.Amount);
```

```
SELECT
      [GroupBy1].[A1] AS [C1]
         FROM ( SELECT
                    SUM( CAST( [Extent1].[Amount] AS bigint)) AS [A1]
                    FROM [dbo].[Transactions] AS [Extent1]
               )  AS [GroupBy1]
```

مثال دوم: سیستم باید بتواند با نبود رکوردها نیز صحیح کار کند

برای نمونه کوئری زیر را بر روی بازه‌ای که سیستم عملکرد نداشته است، در نظر بگیرید:

```
var date = DateTime.Now.AddDays(10);
var sum3 = context.Transactions
                  .Where(x => x.AddDate > date)
                  .Sum(x => (Int64)x.Amount);
```

یک چنین خروجی SQL ایی دارد:

```
SELECT
     [GroupBy1].[A1] AS [C1]
        FROM ( SELECT
                    SUM( CAST( [Extent1].[Amount] AS bigint)) AS [A1]
                    FROM [dbo].[Transactions] AS [Extent1]
                    WHERE [Extent1].[AddDate] > @p__linq__0
              )  AS [GroupBy1]
```

اما در سمت کدهای ما با خطای زیر متوقف می‌شود:

```
The cast to value type 'Int64' failed because the materialized value is null.
Either the result type's generic parameter or the query must use a nullable type.
```

راه حل: استفاده از نوع‌های nullable در اینجا ضروری است:

```
var date = DateTime.Now.AddDays(10);
var sum3 = context.Transactions
                  .Where(x => x.AddDate > date)
                  .Sum(x => (Int64?)x.Amount) ?? 0;
```

به این ترتیب، خروجی صفر را بدون مشکل، دریافت خواهیم کرد.

مثال سوم: حالت‌های خاص استفاده از خواص راهبری

کوئری زیر را در نظر بگیرید:

```
var sum4 = context.Bills.First().Transactions.Sum(x => (Int64?)x.Amount) ?? 0;
```

در اینجا قصد داریم جمع تراکنش‌های صورتحساب اول را بدست بیاوریم که از طریق استفاده از خاصیت راهبری Transactions کلاس Bill، به نحو فوق میسر شده است. به نظر شما خروجی SQL آن به چه صورتی است؟

```
SELECT
     [Extent1].[Id] AS [Id],
     [Extent1].[AddDate] AS [AddDate],
     [Extent1].[Amount] AS [Amount],
     [Extent1].[BillId] AS [BillId]
   FROM [dbo].[Transactions] AS [Extent1]
   WHERE [Extent1].[BillId] = @EntityKeyValue1
```

بله! در اینجا خبری از Sum نیست. ابتدا کل اطلاعات دریافت شده و سپس جمع و منهای نهایی در سمت کلاینت بر روی آن‌ها انجام می‌شود؛ که بسیار ناکارآمد است. (قرار است این مورد ویژه، در نگارش‌های بعدی بهبود یابد)

راه حل کنونی:

```
var entry = context.Bills.First();
var sum5 = context.Entry(entry).Collection(x => x.Transactions).Query().Sum(x => (Int64?)x.Amount) ??
0;
```

در اینجا باید از روش خاصی که مشاهده می‌کنید جهت کار با خواص راهبری استفاده کرد و نکته اصلی آن استفاده از متد Query است. حاصل کوئری LINQ فوق اینبار SQL مطلوب زیر است که سمت سرور عملیات جمع را انجام می‌دهد و نه سمت کلاینت:

```
SELECT
    [GroupBy1].[A1] AS [C1]
     FROM ( SELECT
                SUM( CAST( [Extent1].[Amount] AS bigint)) AS [A1]
                    FROM [dbo].[Transactions] AS [Extent1]
                     WHERE [Extent1].[BillId] = @EntityKeyValue1
            )  AS [GroupBy1]
```

نکاتی که در اینجا ذکر شدند در مورد تمام توابع تجمعی مانند Sum، Count، Max و Min و غیره صادق هستند و باید به آن‌ها نیز دقت داشت.

نظرات خوانندگان

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۸:۵۴

ممنون از این سری پست‌ها. عالین.
و خیلی جالب نشون دادید که استفاده از پروفایلر چقدر اهمیت داره. مرسی.

عنوان: **کوئری هایی با قابلیت استفاده ی مجدد**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۸ ۲۰:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, ORM

با توجه به اصل Dry تا می‌توان باید از نوشتن کدهای تکراری خودداری کرد و کدها را تا جایی که ممکن است به قسمت هایی با قابلیت استفاده‌ی مجدد تبدیل کرد. حین کار کردن با ORMهای معروف مثل NHibernate و EntityFramework زمان زیادی نوشتن کوئری‌ها جهت واکشی داده‌ها از دیتابیس صرف می‌شود. اگر بتوان کوئری هایی با قابلیت استفاده‌ی مجدد نوشت علاوه بر کاهش زمان توسعه قابلیت هایی قدرتمندی مانند زنجیر کردن کوئری‌ها به دنبال هم به دست می‌آید.
با یک مثال نحوه‌ی نوشتن و مزایای کوئری با قابلیت استفاده‌ی مجدد را بررسی می‌کنیم :
برای مثال دو جدول شهرها و دانش آموزان را درنظر بگیرید:

```
namespace ReUsableQueries.Model
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
    [ForeignKey("BornInCityId")]
        public virtual City BornInCity { get; set; }
        public int BornInCityId { get; set; }
    }

    public class City
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<Student> Students { get; set; }
    }
}
```

در ادامه این کلاس‌ها را در معرض دید EF Code first قرار داده:

```
using System.Data.Entity;
using ReUsableQueries.Model;

namespace ReUsableQueries.DAL
{
    public class MyContext : DbContext
    {
        public DbSet<City> Cities { get; set; }
        public DbSet<Student> Students { get; set; }
    }
}
```

و همچنین تعدادی رکورد آغازین را نیز به جداول مرتبط اضافه می‌کنیم:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
        protected override void Seed(MyContext context)
        {
            var city1 = new City { Name = "city-1" };
            var city2 = new City { Name = "city-2" };
            context.Cities.Add(city1);
            context.Cities.Add(city2);
            var student1 = new Student() {Name = "Shaahin",LastName = "Kiassat",Age=22,BornInCity =
city1};
            var student2 = new Student() { Name = "Mehdi", LastName = "Farzad", Age = 31, BornInCity =
```

```
city1 };
            var student3 = new Student() { Name = "James", LastName = "Hetfield", Age = 49, BornInCity
= city2 };
            context.Students.Add(student1);
            context.Students.Add(student2);
            context.Students.Add(student3);
            base.Seed(context);
        }
    }
```

فرض کنید قرار است یک کوئری نوشته شود که در جدول دانش آموزان بر اساس نام ، نام خانوادگی و سن جستجو کند :

```
var context = new MyContext();
          var query= context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
              x => x.Age == age);
```

احتمالا هنوز کسانی هستند که فکر می‌کنند کوئری‌های LINQ همان لحظه که تعریف می‌شوند اجرا می‌شوند اما اینگونه نیست . در واقع این کوئری فقط یک Expression از رکوردهای جستجو شده است و تا زمانی که متد ToList یا ToArray روی آن اجرا نشود هیچ داده ای برگردانده نمی‌شود.

در یک برنامه‌ی واقعی داده‌های باید به صورت صفحه بندی شده و مرتب شده برگردانده شود پس کوئری به این صورت خواهد بود :

```
var query= context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
              x => x.Age == age).OrderBy(x=>x.LastName).Skip(skip).Take(take);
```

ممکن است بخواهیم در متد دیگری در لیست دانش آموزان بر اساس نام ، نام خانوادگی ، سن و شهر جستجو کنیم و سپس خروجی را اینبار بر اساس سن مرتب کرده و صفحه بندی نکنیم:

```
var query = context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where
              (
                  x => x.Age == age).Where(x => x.BornInCityId == 1).OrderBy(x => x.Age);
```

همانطور که می‌بینید قسمت هایی از این کوئری با کوئری هایی که قبلا نوشتیم یکی است ، همچنین حتی ممکن است در قسمت دیگری از برنامه نتیجه‌ی همین کوئری را به صورت صفحه بندی شده لازم داشته باشیم.

اکنون نوشتن این کوئری‌ها میان کد های Business Logic باعث شده هیچ استفاده‌ی مجددی نتوانیم از این کوئری‌ها داشته باشیم. حال بررسی می‌کنیم که چگونه می‌توان کوئری هایی با قابلیت استفاده‌ی مجدد نوشت :

```
namespace ReUsableQueries.Quries
{
    public static class StudentQueryExtension
    {
        public static IQueryable<Student> FindStudentsByName(this IQueryable<Student> students,string
name)
        {
            return students.Where(x => x.Name.Contains(name));
        }
        public static IQueryable<Student> FindStudentsByLastName(this IQueryable<Student> students,
string lastName)
        {
            return students.Where(x => x.LastName.Contains(lastName));
        }
        public static IQueryable<Student> SkipAndTake(this IQueryable<Student> students, int skip , int
take)
        {
            return students.Skip(skip).Take(take);
        }
        public static IQueryable<Student> OrderByAge(this IQueryable<Student> students)
```

```
        {
            return students.OrderBy(x=>x.Age);
        }
    }
}
```

همان طور که مشاهده می‌کنید به کمک متدهای الحاقی برای شیء IQueryable<Student> چند کوئری نوشته ایم . اکنون در محل استفاده از کوئری‌ها می‌توان این کوئری‌ها را به راحتی به هم زنجیر کرد. همچنین اگر روزی قرار شد منطق یکی از کوئری‌ها عوض شود با عوض کردن آن در یک قسمت برنامه همه جا اعمال می‌شود. نحوه‌ی استفاده از این متدهای الحاقی به این صورت خواهد بود :

```
var query =
context.Students.FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(skip,take);
```

فرض کنید قرار است یک سیستم جستجوی پیشرفته به برنامه اضافه شود که بر اساس شرط‌های مختلف باید یک شرط در کوئری اعمال شود یا نشود ، به کمک این طراحی جدید به راحتی می‌توان بر اساس شرط‌های مختلف یک کوئری را اعمال کرد یا نکرد :

```
var query = context.Students.AsQueryable();
          if (searchByName)
          {
              query= query.FindStudentsByName(name);
          }
          if (orderByAge)
          {
              query = query.OrderByAge();
          }
          if (paging)
          {
             query =  query.SkipAndTake(skip, take);
          }
          return query.ToList();
```

همچنین این کوئری‌ها وابسته به ORM خاصی نیستند البته این نکته هم مد نظر است که LINQ Provider بعضی ORMها ممکن است بعضی کوئری‌ها را پشتیبانی نکند.

نظرات خوانندگان

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۸/۱۸ ۲۲:۲۳

ممنون از مطلب خوبتون... می‌خواستم بدونم اگه بخوام این متدها رو (در کلاس StudentQueryExtension) جوری بنویسم که با Anonymous Type هم قابل استفاده باشه چه راه حلی وجود داره؟ (یعنی تمام ستون‌ها رو برنگردونم و فقط اونهایی رو که نیاز دارم نمایش بدم و این اعلام نیاز بتونه داینامیک باشه و از طریق پارامتر به تابع پاس داده بشه یا چیزی شبیه این !) . نوع خروجی متدها بهتره چجوری نوشته بشن؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۸ ۲۲:۴۱

خواهش می‌کنم.
با توجه به این که متدهای الحاقی برای

```
IQueryable<Entity>
```

نوشته شده اند پس نوع خروجی هم باید از همین نوع باشد ، راه حلی که به نظرم می‌آید اینه که برای برگداندن چند ستون نوع برگشتی را از نوع یک CustomObject بگذارید مثلا StudentDTO
در مورد داینامیک بودن نمی‌دانم چه کار باید کرد اما برای خودم هم جالب هست که آیا میشه این کار رو کرد یا خیر .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱:۲۵

- هیچ تغییری را در متدهای الحاقی همه منظوره ایجاد نکنید. این متدها رکوردی رو بر نمی‌گردونند (در متن لینک داده شده). فقط یک سری عبارت هستند. Select نهایی ویژه را پیش از ToList آخر کار انجام بدید.
- برای پویا کردن LINQ امکان استفاده از رشته‌ها وجود داره: (^)
- نوع خروجی متد در این حالت خاص می‌تونه object یا IEnumerable خالی باشد.

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱:۴۷

اول تشکر می‌کنم بابت مطلب خوبتون ..
اگر سوال جناب سیف اللهی رو درست متوجه شده باشم ، ایشون می‌خوان که فیلدهایی رو که از یک تابع برگشت داده میشه خودشون انتخاب کنن و محدود به مقدار بازگشتی از نوع Student برای مثال نباشن .
ایده ای که به ذهن من رسید (بر اساس برداشتی که از سوال داشتم) استفاده از قابلیت بسیار کاربردی Func هستش . یک Func با ورودی از نوع Entity و مقدار بازگشتی از نوع anonymous Type . در هنگام فراخوانی هم میشه از نوع dynamic برای دریافت نتیجه استفاده کرد . یک نمونه از پیاده سازی همچین چیزی رو اینجا قرار دادم .

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۹ ۲:۱۸

ممنونم.
نمونه کد خیلی خوبی بود تشکر.

نویسنده: ابراهیم
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۱:۴۶

سلام. ممنون از مطلب خوبتان. می‌خواستم نظرتان را در رابطهٔ با الگوی Repository بدانم، به نظر من این الگو با اینکه محبوبیت زیادی هم پیدا کرده ولی به پیچیدگی نالازمی نسبت به روش شما دارد. سوالی نیز داشتم، امکان نداشت به شیوه‌ای از IEnumerable به جای IQueryable استفاده شود؟ به نظر من مزیت آن در این است که بتوان خارج از چارچوب ORM از این کوئری‌ها استفاده شود و برای آن‌ها تست ایجاد نمود.
باز هم ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۲:۱

- مطلب جاری نفی کننده وجود لایه سرویس در برنامه نیست و مکمل آن است.
- پیاده سازی الگوی مخزنی را که لینک دادید اشتباه است. دلایل اشتباه بودن آن‌را در این مطلب مطالعه کنید: (^)

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۲:۲

سلام ؛ استفاده از الگوی Repository اضافی در EF Code first؛ آری یا خیر؟!

لطفا مطلب تفاوت‌های IQueryable و IEnumerable را مطالعه بفرمایید.
اگر از IEnumerable استفاده شود دیگر نمی‌توان کوئری‌ها را به هم زنجیر کرد .

نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۰۸/۲۰ ۹:۴

بسیار ممنون از تمام دوستان...

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۷:۹

در قسمت زنجیر کردن کوئری‌ها نباید

```
var query =
context.Students.FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(skip,take);
```

به

```
var query =
context.Students.AsQueryable().FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(sk
ip,take);
```

تغییر کند؟! اگر جواب منفی است چرا؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۸:۴۷

نیازی نیست چون DbSet از یک سری کلاس منجمله IQueryable مشتق می‌شود.

نویسنده: کیا
تاریخ: ۱۳۹۱/۰۸/۲۱ ۹:۱۲

برای حالتی که بخواین بصورت داینامیک و Anonoymous ستونها رو پاس بدین می‌تونین بصورت زیر عمل کنین. در سمت سرویس :

```
public IEnumerable<dynamic> GetCustomColumnsDynamic(Func<Student, dynamic> pColumns)
        {
            return _entities.Select(pColumns).ToList();
        }
```

و برای استفاده :

```
var resultDynamic = _serviceStudent.GetCustomColumnsDynamic
                (
                    x=> new { x.Id, x.LastName, x.Age  }
                );

            MessageBox.Show(resultDynamic.LastName);
```

و البته همونطور که می‌دونین چون نتیجه بصورت dynamic در اختیار شما قرار می‌گیره از امکانات کامپایلر بی نصیب هستید

نویسنده: محمد جواد تواضعی
تاریخ: ۱۳۹۱/۰۸/۲۹ ۱۷:۳۰

سلام
شاهین جان بابت مطلب بسیار عالی بود.
می خواستم نظرت در مورد اینکه برای گرفتن کوئری با قابلیت مجدد از این روش استفاده بشود چیست ؟ Expression tree

و برای کوئری با قابلیت مجدد کدام روش بهینه‌تر می‌باشد ؟

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۸ ۱۳:۱۷

با توجه به مطلبی که در مبحث « تفاوت بین IQueryable و IEnumerable در حین کار با ORMs » بیان شده, خروجی متد یا باید List و یا باید IEnumerable باشد ؟
اگه مثالی هم بیان بشه این مهم بیشتر قابل درک است و یا لینکی که با مثال این رو توضیح داده باشه.
متشکر.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۲/۲۸ ۱۳:۳۴

بستگی داره در چه لایه‌ای کار می‌کنید و این خروجی قراره در چه لایه‌ای استفاده بشه. خروجی لایه سرویس قراره در لایه UI نمایش داده بشه؟ خروجی لایه سرویس نباید IQueryable باشه. داخل لایه سرویس می‌خواهید کوئری‌ها را با هم ترکیب کنید؟ باید IQueryable باشه.

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۸ ۱۵:۱۳

با توجه به موارد و بستگی هایی که بیان کردین, فقط در لایه سرویس(بیزینس) باید IQueryable بودن یا نبودن خروجی متد رو مشخص کنیم و یا همچنین در لایه Repository یا همون DAL هم باید این موارد رو در نظر بگیریم ؟
با تشکر.

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۹ ۱۶:۵۷

اگر منبع معتبری هم باشه که این موارد رو در قبال مثال توضیح داده باشه, میتونه خیلی بیشتر مثمر ثمر واقع بشه.
من خیلی گوگل کردم ولی روش‌ها بسیار متنوع بود و آدم سردرگم میشه بیشتر.
متشکرم.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۳/۰۲/۲۹ ۱۷:۳۰

من یک دور بازخوردهای شما را خواندم اما متوجه موردی که برای شما ابهام ایجاد کرده نشدم.
آیا شما از Entity Framework استفاده می‌کنید؟ اگر پاسخ مثبت است، خود EF لایه‌ی Repository را پیاده سازی کرده است، و این پیاده سازی یک IQueryable جهت انجام Queryهای متفاوت در اختیار شما قرار می‌دهد. شما می‌توانید مستقیما از DbContext سمت لایه‌ی سرویس استفاده کنید و داده‌ها را جهت استفاده برای استفاده کننده‌ی لایه‌ی سرویس فراهم کنید.
لایه‌ی سرویس باید داده‌ها را درون حافظه برگرداند، نه اینکه یک IQueryable برگرداند که استفاده کننده آن را اجرا کند.
از Repository در لایه‌ی سرویس استفاده کنید.

نویسنده: احمدعلی شفیعی
تاریخ: ۱۳۹۳/۰۹/۱۲ ۱۸:۱۴

آیا با این روش ORM هم می‌فهمه که فقط اون اطلاعات رو باید از دیتابیس بگیره؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۹/۱۲ ۱۸:۳۶

همیشه میشه خروجی دقیق ORM رو بدون حدس و گمان لاگ و بررسی کرد: نمایش خروجی SQL کدهای Entity framework 6 در کنسول دیباگ ویژوال استودیو

عنوان: **شبیه سازی outer Join در entity framework**
نویسنده: شاهد محمودی
تاریخ: ۱۳۹۲/۰۸/۲۱ ۹:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, join, left outer join, right outer join

فرض کنید دو جدول پرسنل و شهر را در دیتا بیس خود دارید و 2 بار کد شهر در جدول پرسنل ارتباط داده شده که یکی برای محل تولد و دیگری برای محل صدور
و میخواهید توسط outer join لیست تمامی پرسنل و محل تولد و محل صدور را (در صورت وجود) داشته باشید.

در sql گرفتن نتیجه ذکر شده بصورت زیر به راحتی قابل انجام است

```
SELECT
   dbo.tbl_Personnel.ID,
   dbo.tbl_Personnel.FirstName,
   dbo.tbl_Personnel.LastName,
   dbo.tbl_GeographyPosition.Title AS IssuanceLocation,
   tbl_GeographyPosition_1.Title AS BirthLocation
FROM  dbo.tbl_Personnel LEFT OUTER JOIN
          dbo.tbl_GeographyPosition AS tbl_GeographyPosition_1 ON
          dbo.tbl_Personnel.BirthLocationGeographyPositionID = tbl_GeographyPosition_1.ID LEFT OUTER
JOIN
          dbo.tbl_GeographyPosition ON dbo.tbl_Personnel.IssuanceLocationGeographyPositionID =
dbo.tbl_GeographyPosition.ID
```

اما در ef با توجه به خواص راهبری کمتر از join استفاده میکنیم در ضمن به هیچ وجه از Left Outer Join و Right Outer Join استفاده نمی‌شود و باید کوئری فوق را با کد زیر شبیه سازی کرد

```
var contex = new PersonnelEntities();
var Query = from Personnel in contex.tbl_Personnel
                        join IssuanceLocation in contex.tbl_GeographyPosition on
                              Personnel.IssuanceLocationGeographyPositionID equals IssuanceLocation.ID
into AIssuanceLocation
                        from IL in AIssuanceLocation.DefaultIfEmpty()
                        join BirthLocation in contex.tbl_GeographyPosition on
                              Personnel.BirthLocationGeographyPositionID equals BirthLocation.ID into
ABirthLocation
                        from BL in ABirthLocation.DefaultIfEmpty()
                        //where
                        select new
                        {
                            Personnel.ID,
```

```
                                Personnel.FirstName,
                                Personnel.LastName,
                                IssuanceLocation = IL.Title,
                                BirthLocation = BL.Title
                            };
```

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۳:۲۱

جهت تکمیل بحث، اگر مدل‌های برنامه به این صورت باشند (محل تولد اجباری است و Id کلید خارجی آن نال پذیر نیست؛ به همراه محل صدور اختیاری، که Id نال پذیر دارد):

```
public class Place
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<Person> Personnel { set; get; }
    }

    public class Person
    {
        public int Id { set; get; }
        public string FirstName { set; get; }
        public string LastName { set; get; }

        [ForeignKey("BirthPlaceId")]
        public virtual Place BirthPlace { set; get; }
        public int BirthPlaceId { set; get; }

        [ForeignKey("IssuanceLocationId")]
        public virtual Place IssuanceLocation { set; get; }
        public int? IssuanceLocationId { set; get; }
    }
```

با این Context :

```
public class MyContext : DbContext
    {
        public DbSet<Place> Places { get; set; }
        public DbSet<Person> Personnel { get; set; }

        public MyContext()
        {
            this.Database.Log = sql => Console.WriteLine(sql);
        }
    }
```

آنگاه خروجی کوئری ذیل (که یک include دارد روی خاصیت راهبری که مقدار Id کلید خارجی آن ممکن است نال باشد (محل صدور) و نه مورد دومی که Id غیرنال پذیر دارد (محل تولد))

```
context.Personnel.Include(x => x.IssuanceLocation)
```

معادل خواهد بود با (left outer join به صورت خودکار تشکیل شده)

```
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[LastName] AS [LastName],
    [Extent1].[BirthPlaceId] AS [BirthPlaceId],
    [Extent1].[IssuanceLocationId] AS [IssuanceLocationId],
    [Extent2].[Id] AS [Id1],
    [Extent2].[Name] AS [Name],
    [Extent1].[Place_Id] AS [Place_Id]
    FROM  [dbo].[People] AS [Extent1]
    LEFT OUTER JOIN [dbo].[Places] AS [Extent2] ON [Extent1].[IssuanceLocationId] = [Extent2].[Id]
```

و خروجی کوئری زیر که DefaultIfEmpty را هم لحاظ کرده و join نویسی صریحی هم دارد (مطابق مقاله فوق):

```
var query = from personnel in context.Personnel
                            join issuanceLocation in context.Places on
                                  personnel.IssuanceLocationId equals issuanceLocation.Id into
aIssuanceLocation
                            from IL in aIssuanceLocation.DefaultIfEmpty()
                            join birthLocation in context.Places on
                                  personnel.BirthPlaceId equals birthLocation.Id into aBirthLocation
                            from BL in aBirthLocation.DefaultIfEmpty()
                            select new
                               {
                                   personnel.Id,
                                   personnel.FirstName,
                                   personnel.LastName,
                                   IssuanceLocation = IL.Name,
                                   BirthLocation = BL.Name
                               };
```

معادل است با:

```
SELECT
                        [Extent1].[Id] AS [Id],
                        [Extent1].[FirstName] AS [FirstName],
                        [Extent1].[LastName] AS [LastName],
                        [Extent2].[Name] AS [Name],
                        [Extent3].[Name] AS [Name1]
                        FROM [dbo].[People] AS [Extent1]
                        LEFT OUTER JOIN [dbo].[Places] AS [Extent2] ON [Extent1].[IssuanceLocationId] =
[Extent2].[Id]
                        INNER JOIN [dbo].[Places] AS [Extent3] ON [Extent1].[BirthPlaceId] =
[Extent3].[Id]
```

و البته این خروجی دوم فقط در صورتی تشکیل می‌شود که قسمت select new ذکر شود. در غیراینصورت مشکل select n+1 را پیدا می‌کند و اصلا چنین join ایی تشکیل نخواهد شد (در یک حلقه، به ازای هر شخص، یکبار کوئری select به جدول مکان‌ها تشکیل می‌شود). همچنین یک inner join هم علاوه بر left outer join تشکیل شده (برای فیلد غیرنال پذیر).

حتی همین حالت دوم را هم با کوئری ذیل که از خواص راهبری استفاده کرده، می‌توان تولید کرد:

```
var query = context.Personnel.Select(x => new
             {
              x.Id,
              x.FirstName,
              x.LastName,
              BirthPlaceName = x.BirthPlace.Name,
              IssuanceLocationName = x.IssuanceLocation == null ? "" : x.IssuanceLocation.Name
             });
```

با این خروجی SQL (به صورت خودکار برای فیلد نال پذیر، left outer join و برای غیر نال پذیر inner join تشکیل داده)

```
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[LastName] AS [LastName],
    [Extent2].[Name] AS [Name],
    CASE WHEN ([Extent3].[Id] IS NULL) THEN N'' ELSE [Extent3].[Name] END AS [C1]
    FROM   [dbo].[People] AS [Extent1]
    INNER JOIN [dbo].[Places] AS [Extent2] ON [Extent1].[BirthPlaceId] = [Extent2].[Id]
    LEFT OUTER JOIN [dbo].[Places] AS [Extent3] ON [Extent1].[IssuanceLocationId] = [Extent3].[Id]
```

نویسنده: شاهد محمودی
تاریخ: ۱۳۹۲/۰۸/۲۲ ۲:۲۴

با تشکر اما جناب نصیری برای این مشکل من ، که هر 2 جدول اختیاری است و نال پذیر است فکر نکنم بشه از این روش استفاده کرد

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۲۲ ۹:۱۸

فکر کنم خلاصه مطلبی که عنوان شده اینه که اگر در طراحی، فیلد نال پذیر *داشته باشید* ، در صورت استفاده از خواص راهبری متناظر با این فیلدها، به صورت خودکار left outer join درست میشه. بررسی‌اش هم نیازی به حدس و گمان نداره. مطلب لاگ کردن خروجی SQL می‌تونه کمک کنه .

عنوان: **اعمال توابع تجمعی بر روی چند ستون در Entity framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۷ ۸:۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

فرض کنید که می‌خواهیم معادل کوئری زیر را که اعمال توابع تجمعی به چند ستون است،

```
SELECT sum([Rating_TotalRating]), sum([Rating_TotalRaters]), sum([Rating_AverageRating]) FROM
[BlogPosts]
```

در Entity framwork به کمک LINQ to Entities تهیه کنیم.
نکته‌ای که در اینجا وجود دارد، نبود گروه بندی (حداقل به ظاهر) در کوئری نوشته شده است. اما واقعیت این است که یک بانک اطلاعاتی به صورت ضمنی در مورد یک چنین کوئری‌هایی نیز گروه بندی را انجام می‌دهد. برای اینکار، کل رکوردهای مدنظر را یک گروه تصور می‌کند.
اگر سعی کنیم چنین کوئری را توسط عبارات LINQ ایجاد کنیم، در سعی اول به چنین کوئری خواهیم رسید که اصلا کامپایل نمی‌شود:

```
context.BlogPost.Select(r =>
                                        new
                                        {
                                            Sum1 = r.Sum(x => x.RatingTotalRating),
                                            Sum2 = r.Sum(x => x.RatingTotalRaters),
                                            Sum3 = r.Sum(x => x.RatingAverageRating)
                                        }).FirstOrDefault();
```

بنابراین به نظر می‌رسد که شاید بهتر باشد از روش ذیل استفاده کنیم:

```
var sum1 = context.BlogPost.Sum(x => x.RatingTotalRating);
var sum2 = context.BlogPost.Sum(x => x.RatingTotalRaters);
var sum2 = context.BlogPost.Sum(x => x.RatingAverageRating);
```

این روش کار می‌کند و نهایتا معادل نتایج کوئری اول را نیز حاصل خواهد کرد؛ اما با سه بار رفت و برگشت به بانک اطلاعاتی که اصلا بهینه نیست.

راه حل: ایجاد گروه بندی ضمنی SQL به صورت صریح در عبارات LINQ

```
context.BlogPost
                        .GroupBy(dummyNumber => 0)
                        .Select(r =>
                                        new
                                        {
                                            Sum1 = r.Sum(x => x.RatingTotalRating),
                                            Sum2 = r.Sum(x => x.RatingTotalRaters),
                                            Sum3 = r.Sum(x => x.RatingAverageRating)
                                        }).FirstOrDefault();
```

در این کوئری جدید که بر اساس عدد ثابت صفر گروه بندی شده است، یک چنین SQL ایی تولید می‌شود:

```
SELECT TOP (1)
                         [Extent1].[K1] AS [K1],
                         Sum([Extent1].[A1]) AS [A1],
                         Sum([Extent1].[A2]) AS [A2],
                         Sum([Extent1].[A3]) AS [A3]
                         FROM ( SELECT
                             0 AS [K1],
                             [Extent1].[RatingTotalRating] AS [A1],
                             [Extent1].[RatingTotalRaters] AS [A2],
       [Extent1].[RatingAverageRating] AS [A3]
```

```
                    FROM [dbo].[BlogPosts] AS [Extent1]
                )  AS [Extent1]
                GROUP BY [K1]
```

ابتدا یک ستون فرضی با مقدار ثابت صفر به رکوردها اضافه می‌شود. سپس بر اساس این ستون فرضی، کلیه ردیف‌ها گروه بندی شده و در ادامه توابع تجمعی بر روی آن‌ها اعمال می‌گردند. به این ترتیب تعداد رفت و برگشت‌ها به بانک اطلاعاتی به همان یک مورد کاهش خواهد یافت.

عنوان: **متد LastOrDefault در EF**
نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۲/۰۵/۰۱ ۸:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, LINQ to Entity

اگر بخواهیم اولین رکورد از یک جدول را توسط EF درخواست نماییم از متد FirstOrDefaultیا First استفاده می‌شود. برای مثال واکشی اولین رکورد از جدول Student به صورت زیر است:

```
var student=context.Students.FirstOrDefault();
```

در این حالت اولین رکورد از جدول student واکشی می‌شود و اگر رکوردی موجود نباشد یک مقدار null بازگشت داده می‌شود. حال اگر بخواهید به جای اولین رکورد آخرین رکورد را واکشی نمایید چطور؟ برای یافتن آخرین رکورد در لیست‌ها ی Generic و کلا لیست‌های Enumerable از متد LastOrDefault استفاده می‌شود. با این حال این متد توسط Entity Framework پشتیبانی نمی‌شود و در صورتی که از کد زیر استفاده کنید برنامه با خطا متوقف خواهد شد:

```
var student=context.Students.LastOrDefault();
```

دو راه حل برای رفع این مشکل به ذهن می‌رسد:
روش اول: می‌توان خروجی را ابتدا به یک نوع Enumerable مانند List تبدیل کرد و سپس از متد LastOrDefault استفاده کرد. کد زیر را در نظر بگیرید:

```
var student=context.Students.ToList().LastOrDefault();
```

در کد بالا ابتدا رکوردهای جدول Student از درون بانک اطلاعات به صورت کامل واکشی شده و سپس رکورد آخر از میان آنها جدا می‌شود. این حالت در حالی که رکوردهای کمی در جدول وجود داشته باشد روش بدی به حساب نمی‌آید ولی اگر تعداد رکوردها زیاد باشد (اکثر مواقع نیز به همین شکل است) روش مناسبی نمی‌باشد و باعث کندی برنامه می‌شود.

روش دوم: با توجه به اینکه تنها به یک رکورد (آخرین رکورد) نیاز داریم بهتر است یک رکورد هم واکشی شود. در این روش برای اینکه بتوان به آخرین رکورد رسید ابتدا رکوردهای جدول را به صورت نزولی مرتب می‌کنیم و سپس از متد FirstOrDefault برای واکشی آخرین رکورد استفاده می‌نمایید. برای مثال:

```
var student=context.Students.OrderByDescending(s=>s.Id).FirstOrDefault();
```

در کد بالا ابتدا رکوردها را بر اساس فیلد مورد نظر به صورت نزولی مرتب کرده ایم(در نظر داشته باشید عملیات مرتب سازی را می‌توان بر اساس فیلدی که مورد نظر است انجام داد) و پس از آن با توجه به اینکه رکوردها به صورت نزولی مرتب شده اند و رکورد آخر به اول منتقل شده است از متد FirstOrDefault جهت دسترسی به آخرین رکورد که در حال حاضر اول لیست رکوردها است استفاده شده است. سرعت این روش به مراتب از روش اول بیشتر می‌باشد. برای بالا رفتن سرعت مرتب سازی در جداول بزرگ نیز می‌توان از تدابیری همچون Index گذاری بر روی فیلدها در DataBase استفاده کرد(با توجه به فیلدی که قرار است مرتب سازی بر اساس آن انجام شود).

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۰۱ ۸:۳۸

ممنون. روش دوم به select top 1 در حین استفاده از SQL Server ترجمه میشه.

نویسنده: سامان هاشمی
تاریخ: ۱۳۹۲/۰۵/۰۱ ۱۱:۵۴

مورد اول اصلا توصیه نکنید بعدها به دلیل مشکل کارآیی که داره خیلی اذیت میکنه همون مورد دوم تنها گزینه و بهترین گزینه است!

نویسنده: ابوالفضل
تاریخ: ۱۳۹۲/۰۵/۰۲ ۱۲:۳۴

یک نکته اینکه : زمانی که قصد داریم آخرین رکورد افزوده شده رو به این طریق و بر اساس فیلدی غیر از کلید واکشی کنیم (به فرض تاریخ فاکتور و ...) حتما باید برای آن فیلد در صورت کلید نبودنش ، ایندکس ایجاد کنیم تا واکشی در کوتاهترین زمان ممکن در حجم بالای اطلاعات صورت گیرد .

نویسنده: باغبان
تاریخ: ۱۳۹۲/۰۵/۰۳ ۱۵:۳۹

ممنون از آموزش خوبتون می‌خواستم بپرسم در حالت دوم فقط یک رکورد از دیتابیس واکشی میشه؟ یا اینکه از میون رکوردهای واکشی شده یک رکورد را انتخاب می‌کنه؟

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۲/۰۵/۰۳ ۱۷:۸

در روش دوم فقط یک رکورد واکشی می‌شود.

عنوان: Func یا Expression Func در EF
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۶ ۸:۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, LINQ

با بررسی کدهای مختلف Entity framework گاهی از اوقات در امضای توابع کمکی نوشته شده، Func<> مشاهده می‌شود و در بعضی از موارد Expression<Func<>> و ... به نظر استفاده کنندگان دقیقا نمی‌دانند که تفاوت این دو در چیست و کدامیک را باید/یا بهتر است بکار برد.

ابتدا مثال کامل ذیل را در نظر بگیرید:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;
using System.Linq.Expressions;

namespace Sample
{
    public abstract class BaseEntity
    {
        public int Id { set; get; }
    }

    public class Receipt : BaseEntity
    {
        public int TotalPrice { set; get; }
    }

    public class MyContext : DbContext
    {
        public DbSet<Receipt> Receipts { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            if (!context.Receipts.Any())
            {
                for (int i = 0; i < 20; i++)
                {
                    context.Receipts.Add(new Receipt { TotalPrice = i });
                }
            }
            base.Seed(context);
        }
    }

    public static class EFUtils
    {
        public static IList<T> LoadEntities<T>(this DbContext ctx, Expression<Func<T, bool>> predicate)
where T : class
        {
            return ctx.Set<T>().Where(predicate).ToList();
        }

        public static IList<T> LoadData<T>(this DbContext ctx, Func<T, bool> predicate) where T : class
        {
            return ctx.Set<T>().Where(predicate).ToList();
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
```

```
            startDB();

            using (var context = new MyContext())
            {
                var list1 = context.LoadEntities<Receipt>(x => x.TotalPrice == 10);
                var list2 = context.LoadData<Receipt>(x => x.TotalPrice == 20);
            }
        }

        private static void startDB()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            // Forces initialization of database on model changes.
            using (var context = new MyContext())
            {
                context.Database.Initialize(force: true);
            }
        }
    }
}
```

در این مثال ابتدا کلاس Receipt تعریف شده و سپس توسط کلاس MyContext در معرض دید EF قرار گرفته است. در ادامه توسط کلاس Configuration نحوه آغاز بانک اطلاعاتی مشخص گردیده است؛ به همراه ثبت تعدادی رکورد نمونه.

نکته اصلی مورد بحث، کلاس کمکی EFUtils است که در آن دو متد الحاقی LoadEntities و LoadData تعریف شده‌اند. در متد LoadEntities، امضای متد شامل Expression Func است و در متد LoadData فقط Func ذکر شده است.

در ادامه اگر برنامه را توسط فراخوانی متد RunTests اجرا کنیم، به نظر شما خروجی SQL حاصل از list1 و list2 چیست؟

احتمالا شاید عنوان کنید که هر دو یک خروجی SQL دارند (با توجه به اینکه بدنه متدهای LoadEntities و LoadData دقیقا/یا به نظر یکی هستند) اما یکی از پارامتر 10 استفاده می‌کند و دیگری از پارامتر 20. تفاوت دیگری ندارند.

اما ... اینطور نیست!

خروجی SQL متد LoadEntities در متد RunTests به صورت زیر است:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[TotalPrice] AS [TotalPrice]
FROM [dbo].[Receipts] AS [Extent1]
WHERE 10 = [Extent1].[TotalPrice]
```

و ... خروجی متد LoadData به نحو زیر:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[TotalPrice] AS [TotalPrice]
FROM [dbo].[Receipts] AS [Extent1]
```

بله. در لیست دوم هیچ فیلتری انجام نشده (در حالت استفاده از Func خالی) و کل اطلاعات موجود در جدول Receipts، بازگشت داده شده است.

چرا؟

Func اشاره‌گری است به یک متد و Expression Func بیانگر ساختار درختی عبارت lambda نوشته شده است. این ساختار درختی صرفا بیان می‌کند که عبارت lambda منتسب، چه کاری را قرار است یا می‌تواند انجام دهد؛ بجای انجام واقعی آن.

```
public static IQueryable<TSource> Where<TSource>(this IQueryable<TSource> source,
Expression<Func<TSource, bool>> predicate);
public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool>
predicate);
```

اگر از Expression Func استفاده شود، از متد Where ایی استفاده خواهد شد که خروجی IQueryable دارد. اگر از Func استفاده شود، از overload دیگری که خروجی و ورودی IEnumerable دارد به صورت خودکار استفاده می‌گردد.

بنابراین هرچند بدنه دو متد LoadEntities و LoadData به ظاهر یکی هستند، اما بر اساس نوع ورودی Where ایی که دریافت می‌کنند، اگر Expression Func باشد، EF فرصت آنالیز و ترجمه عبارت ورودی را خواهد یافت اما اگر Func باشد، ابتدا باید کل

اطلاعات را به صورت یک لیست IEnumerable دریافت و سپس سمت کلاینت، خروجی نهایی را فیلتر کند.
اگر برنامه را اجرا کنید نهایتا هر دو لیست یک و دو، بر اساس شرط عنوان شده عمل خواهند کرد و فیلتر خواهند شد. اما در حالت اول این فیلتر شدن سمت بانک اطلاعاتی است و در حالت دوم کل اطلاعات بارگذاری شده و سپس سمت کاربر فیلتر می‌شود (که کارآیی پایینی دارد).

نتیجه گیری

به امضای متد Where ایی که در حال استفاده است دقت کنید. همینطور در مورد Count ، Sum و یا موارد مشابه دیگری که predicate قبول می‌کنند.

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۲/۰۴/۲۶ ۲۰:۱

اوایل که از Entity Framework استفاده میکردم دچار همین مشکل شدم. در یک برنامه تمام متدهای دارای شرط لایه سرویسم رو با استفاده از Func پیاده سازی کرده بودم و بعد از مدتی متوجه شدم که برای دریافت یک رکورد از جدول هم برنامه خیلی کند عمل میکنه. کد زیر رو نگاه کنید:

```
public virtual TEntity Find(Func<TEntity, bool> predicate)
{
    return _tEntities.Where(predicate).FirstOrDefault();
}
```

در این حالت همه رکوردها از جدول مورد نظر واکشی میشه و بعد فقط یکی از آنها (اولین رکورد) در سمت کلاینت جدا و بازگشت داده میشه.

نویسنده: Saleh
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۰:۴۲

با تشکر از شما
جهت اطلاع دوستان:
کتاب Functional Programming In CS نوشته Oliver Sturm به طور کامل مبحث Genericها را پوشش داده و موضوعات Func و Expressionها را مفصل تشریح کرده است.

نویسنده: Saleh
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۲:۰

استفاده از Predicate<> چه تفاوتی با استفاده شما از Func دارد؟

حتی آقای نصیری هم به همین صورت استفاده کرده اند.

```
public virtual TEntity Find(Expression<Func<TEntity, bool>> predicate)
```

```
public virtual TEntity Find(Expression<Predicate<TEntity>> predicate)
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۰ ۱۲:۳۸

استفاده نشده. فرق است بین یک پارامتر با *نام* predicate و یک delegate به نام Predicate . ضمنا Func هم یک Delegate است .

عنوان: **درج یک باره چندین رکورد بصورت همزمان هنگام استفاده از ORMها**
نویسنده: محسن اسماعیل پور
تاریخ: ۱۳۹۲/۰۳/۰۵ ۱۹:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, SQLBulkCopy

همونطور که میدونیم درج یکباره چندین رکورد هنگام استفاده از Entity Framework فعلا امکان پذیر نیست و باید از یک حلقه استفاده کرد و آنها رو یک به یک وارد کرد که هنگامی تعداد رکوردها زیاد باشن زمان اجرا یکم زیاد میشه. برای رفع این مشکل در EF Code First میتونین خاصیت AutoDetectChangesEnabled رو برای Context غیرفعال کنید که استفاده از این روش قبلا در این مقاله توضیح داده شده است. راه دیگه استفاده از SqlBulkCopy هست که میتوانید هنگام استفاده از ORM ها ازش استفاده کنید. اگه قبلا از ADO.NET استفاده کرده باشید و خواسته باشید تعداد زیادی رکورد رو بصورت همزمان وارد دیتابیس کنید حتما با SqlBulkCopy آشنایی دارید.

فرض کنید دارید در پروژه، از Entity Framework استفاده میکنید و یک مدل با نام Person دارید که تعریفش به صورت زیر است

```
public class Person
{
     public int PersonId { get; set; }

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

حالا میخوایم تعداد ۵۰۰۰ رکورد از Person رو یکجا وارد دیتابیس کنیم. برای استفاده از SqlBulkCopy ، روش به این شکل هست که ابتدا یک DataTable ایجاد میکنیم. سپس ستونهای متناظر با جدول Person رو با استفاده از DataColumn ایجاد میکنیم و DataColumn های ایجاد شده رو به DataTable اضافه میکنیم و سپس اطلاعات رو وارد DataTable میکنیم و اون رو با استفاده از SqlBulkCopy وارد دیتابیس میکنیم که این روش یکم وقتگیر و خسته کننده است.

راه آسانتر استفاده از یک کتابخانه با نام EntityDataReader هست که توسط مایکروسافت نوشته شده که دیگه نیازی به ساختن DataTable نیست و این کتابخانه کارهای لازم رو خودش انجام میده. در پروژتون یک کلاس با نام EntityDataReader ایجاد کنید و سورس مربوط این کلاس رو از اینجا copy و در داخل کلاس paste کنید.

حالا یک لیست از Pesron با نام personList ایجاد مینماییم و با استفاده از یک حلقه تعداد ۵۰۰۰ تا نمونه از Person ایجاد و به لیست اضافه میکنیم.

```
var personList = new List<Person>();
for (var i = 0; i < 5000; i++)
{
    var person = new Person
        {
            Name = "John Doe",
        };
}
```

در ادامه برای استفاده از SqlBulkCopy نیاز به ConnectionString و نام جدول متناظر با کلاس Person در دیتابیس داریم .

اگر از پروژ وب استفاده میکنید میتونید با این خط کد ConnectionString رو که در فایل web.config ذخیره شده است بروگردونید که در اینجا DataConnection نام ConnectionString ذخیره شده در web.config هست.

```
var connectionString = ConfigurationManager.ConnectionStrings["DataConnection"].ConnectionString;
```

اگر از EF Code First استفاده میکنید و در تنظیمات Context خاصیت PluralizingTableNameConvention رو حذف کردیداید نام جدول dbo.Person هست و در غیر اینصورت db.People هست .

و در ادامه داریم:

```
var connectionString = ConfigurationManager.ConnectionStrings["DataConnection"].ConnectionString;
var bulkCopy = new SqlBulkCopy(connectionString) { DestinationTableName = "dbo.Person" };
bulkCopy.WriteToServer(personList.AsDataReader()  );
```

سرعت این روش بسیار بالاست و برای درجهای با تعداد بالا بهینه است.

برای ویرایش و حذف چندین رکورد بصورت همزمان متیونید از کتابخانه Entity Framework Extended Library استفاده کنید که امکانات دیگری هم داره و از طریق nuget هم قابل نصب است.

نظرات خوانندگان

نویسنده: مصطفی
تاریخ: ۱۳۹۲/۰۳/۰۵ ۲۰:۱۰

این روش یک مشکل خیلی بد داره که اگه تو کلاس مدل از complextype استفاده کرده باشین دیگه نمیشه از کلاس EntityDataReader استفاده کرد
مشکل دوم اینه که اگه پراپرتی‌های مدلتون رو تو فایل کانفیگ به یک ستون غیر هم نام مپ کرده باشین بازم نمیشه از این روش استفاده کرد

نویسنده: محسن اسماعیل پور
تاریخ: ۱۳۹۲/۰۳/۰۵ ۲۲:۱۲

برای رفع مشکل دوم میتونید از DataTable استفاده کنید و نام خاصیتی که ستون متناظرش در جدول فرق داره رو هنگام تعریف DataColumn عوض کنید. روال کار همینه فقط اضافه کاری داره.

نویسنده: حسنین
تاریخ: ۱۳۹۲/۰۳/۰۶ ۱۸:۴۵

جسارتا در انتهای مقاله به گمانم غلط تایپی یا اشتباه لبی باشد:
"و در غیر اینصورت db.Persons هست . "

باید باشد: " و در غیر اینصورت db.People هست

نویسنده: محسن اسماعیل پور
تاریخ: ۱۳۹۲/۰۳/۰۶ ۲۳:۱۵

با تشکر از توجه شما، اصلاح گردید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۴ ۸:۲۴

یک خبر خوب:
در EF 6 نسخه بتا، متدی اضافه شده به نام AddRange که افزودن تعداد زیادی رکورد رو سهولت بخشیده
INSERTing many rows with Entity Framework 6 beta 1 and SQL Server Compact

نویسنده: علی
تاریخ: ۱۳۹۲/۰۳/۲۳ ۱۱:۵۰

سلام
من خطای زیر رو دارم. برای ویرایش چند رکورد به صورت یکجا از Entity Framework Extended Library استفاده کردم. اما مشکلی دارم این است من یک جدول در پایگاه دارم که می‌خوام به صورت یکجا ویرایش کنم به این صورت که ابتدا شماره فیلد هایی که را قرار است تغییر دهم(فیلد ID) داخل یک لیست ریختم و سپس از جدول فقط اون فیلدها رو ویرایش کنم
چند روش برای join جدول با این لیست امتحان کردم یا join خطا می‌دهد اگر join نیز مشکلی نداشت مقدار برگشتی از نوع enumerable هست و زمانی که از update کلاس Entity Framework Extended Libraryاستفاده میکنم
خطا می‌دهد که لیستی که قرار است update شود باید از نوع dbquery باشد

Unable to create a constant value of type 'extendeexample.MyClass'. Only primitive types or enumeration types are supported in this context.

سورس پروژه به همراه پایگاه داده

project.rar

عنوان: **استفاده از خواص راهبری در Entity framework بجای Join نویسی**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۲:۲۴
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

یکی از مزایای مهم استفاده از Entity framework، خواص راهبری (navigation properties) آن هستند که امکان تهیه کوئری‌های بین جداول را به سادگی و به نحوی منطقی فراهم می‌کنند.
برای مثال دو جدول شهرها و افراد را درنظر بگیرید. مقصود از تعریف جدول شهرها در اینجا، مشخص سازی محل تولد افراد است:

```
public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }

        [ForeignKey("BornInCityId")]
        public virtual City BornInCity { get; set; }
        public int BornInCityId { get; set; }
    }

    public class City
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<Person> People { get; set; }
    }
```

در ادامه این کلاس‌ها را در معرض دید EF Code first قرار داده:

```
public class MyContext : DbContext
    {
        public DbSet<City> Cities { get; set; }
        public DbSet<Person> People { get; set; }
    }
```

و همچنین تعدادی رکورد آغازین را نیز به جداول مرتبط اضافه می‌کنیم:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            var city1 = new City { Name = "city-1" };
            var city2 = new City { Name = "city-2" };
            context.Cities.Add(city1);
            context.Cities.Add(city2);

            var person1 = new Person { Name = "user-1", BornInCity = city1 };
            var person2 = new Person { Name = "user-2", BornInCity = city1 };
            context.People.Add(person1);
            context.People.Add(person2);

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

در این حالت برای نمایش لیست نام افراد به همراه محل تولد آن‌ها، بنابر روال سابق SQL نویسی، نوشتن کوئری LINQ زیر بسیار متداول است:

```
public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
```

```
            using (var context = new MyContext())
            {
                var peopleAndCitiesList = from person in context.People
                           join city in context.Cities
                           on person.BornInCityId equals city.Id
                           select new
                           {
                              PersonName = person.Name,
                              CityName = city.Name
                           };

                foreach (var item in peopleAndCitiesList)
                {
                    Console.WriteLine("{0}:{1}", item.PersonName, item.CityName);
                }
            }
        }
    }
```

که حاصل آن اجرای کوئری ذیل بر روی بانک اطلاعاتی خواهد بود:

```
SELECT
          [Extent1].[BornInCityId] AS [BornInCityId],
          [Extent1].[Name] AS [Name],
          [Extent2].[Name] AS [Name1]
FROM  [dbo].[People] AS [Extent1]
INNER JOIN [dbo].[Cities] AS [Extent2] ON [Extent1].[BornInCityId] = [Extent2].[Id]
```

این نوع کوئری‌های join دار را به نحو ساده‌تری نیز می‌توان در EF با استفاده از خواص راهبری و بدون join نویسی مستقیم تهیه کرد:

```
var peopleAndCitiesList = context.People
                                 .Select(person => new
                                                   {
                                                       PersonName = person.Name,
                                                       CityName = person.BornInCity.Name
                                                   });
```

که دقیقا همان خروجی SQL یاد شده را تولید می‌کند.

مثال دوم:

می‌خواهیم لیست شهرها را بر اساس تعداد کاربر متناظر به صورت نزولی مرتب کنیم:

```
var citiesList = context.Cities.OrderByDescending(x => x.People.Count());
foreach (var item in citiesList)
{
    Console.WriteLine("{0}", item.Name);
}
```

همانطور که مشاهده می‌کنید از خواص راهبری در قسمت order by هم می‌شود استفاده کرد. خروجی SQL کوئری فوق به صورت زیر است:

```
SELECT
[Project1].[Id] AS [Id],
[Project1].[Name] AS [Name]
FROM ( SELECT
        [Extent1].[Id] AS [Id],
        [Extent1].[Name] AS [Name],
        (SELECT
                COUNT(1) AS [A1]
                FROM [dbo].[People] AS [Extent2]
                WHERE [Extent1].[Id] = [Extent2].[BornInCityId]) AS [C1]
        FROM [dbo].[Cities] AS [Extent1]
)  AS [Project1]
ORDER BY [Project1].[C1] DESC
```

مثال سوم:

در ادامه قصد داریم لیست شهرها را به همراه تعداد نفرات متناظر با آن‌ها نمایش دهیم:

```
var peopleAndCitiesList = context.Cities
                                    .Select(city => new
                                                {
                                                    InUseCount = city.People.Count(),
                                                    CityName = city.Name
                                                });

foreach (var item in peopleAndCitiesList)
{
     Console.WriteLine("{0}:{1}", item.CityName, item.InUseCount);
}
```

در اینجا از خاصیت راهبری People برای شمارش تعداد اعضای متناظر با هر شهر استفاده شده است.

خروجی SQL کوئری فوق به نحو ذیل است:

```
SELECT
[Extent1].[Id] AS [Id],
(SELECT
        COUNT(1) AS [A1]
        FROM [dbo].[People] AS [Extent2]
        WHERE [Extent1].[Id] = [Extent2].[BornInCityId]) AS [C1],
[Extent1].[Name] AS [Name]
FROM [dbo].[Cities] AS [Extent1]
```

نظرات خوانندگان

نویسنده: ایلیا
تاریخ: ۱۳۹۱/۰۷/۰۸ ۲۰:۳۲

مختصر و مفید. عالی. سپاس.

نویسنده: میرزایی
تاریخ: ۱۳۹۱/۰۷/۱۰ ۱۲:۵۸

با سلام
به موضوع جالب و کاربردی ای اشاره فرمودید.

لطفا روش کار در هنگامی که ارتباط دو جدول به صورت یک به چند باشد و قصد بازیابی رکورهایی را از جدول اول در حالتی که حداقل یک رکورد در جدول دوم با شرط ما وجود داشته باشد را بیان فرمایید.

مثلا جدول کارمندان یک شرکت با شرکت هایی که هر فرد قبلا در آن سابقه کار داشته است.
میخواهیم کارمندانی را که در شرکت x کار کرده اند را به دست آوریم.

با تشکر از مطالب مفید شما.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۰ ۱۳:۹

معادل سؤال شما با توجه به مدل‌های فوق به صورت زیر است:
می‌خواهیم لیست افرادی را بدست بیاوریم که در شهر x متولد شده‌اند.
روش اول: اگر شماره شهر را داریم:

```
var cityId = 1;
var list = context.People.Where(x => x.BornInCityId == cityId).ToList();
```

روش دوم: اگر نام شهر را داریم:

```
var cityName = "city-1";
var list2 = context.People.Where(x => x.BornInCity.Name == cityName).ToList();
```

در روش اول از نکته تعریف کلید خارجی استفاده شده.
در روش دوم از نکته استفاده از خواص راهبری، استفاده شده.

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۱۶ ۱:۵۷

ممنون از مطلب مفیدتون . جالب اینه که بدون هیچ دردسری از خواص راهبری میشه برای به روز رسانی و افزودن رکوردهای مرتبط در صورت وجود رابطه‌های صحیح و نرمال سازی دقیق پایگاه داده بهترین استفاده رو کرد ... واقعا ORMها برنامه نویسارو از شر کد نویسی تکراری و خسته کننده‌ی بانک اطلاعاتی تا حد زیادی راحت کردن ...

نویسنده: debuger
تاریخ: ۱۳۹۲/۰۴/۲۳ ۰:۲۹

با سلام؛ اگه حالتی که برای کاربر میرزایی پاسخ دادید برعکس بشه کوئری به چه صورت میشه ، یعنی اگر بخوایم فهرست شهرهایی که در اون فردی به اسم خاصی متولد شده رو بدست بیاریم (خروجی کوئری از جنس لیستی از شهر باشه) کوئری رو به صورت زیر نوشتم اگه راهنمایی کنید در صورتی که بخوایم از طریق cities به خروجی مورد نظر برسیم ممنون میشم .

```
string personName = "user-1";
            var result = context.People.Where(p => p.Name == personName).Select(c =>
c.BornInCity).ToList();
```

ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۳ ۰:۵۷

از Any استفاده کنید:

```
var citiesContainPerson = context.Cities.Where(city => city.People.Any(person => person.Name == "user-
1")).ToList();
```

با این خروجی SQL:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name]
FROM [dbo].[Cities] AS [Extent1]
WHERE  EXISTS (SELECT
1 AS [C1]
FROM [dbo].[People] AS [Extent2]
WHERE ([Extent1].[Id] = [Extent2].[BornInCityId]) AND (N'user-1' = [Extent2].[Name])
)
```

نویسنده: میثم
تاریخ: ۱۳۹۲/۰۹/۱۱ ۱۹:۵۴

سلام واقعا ممنون بابت این مطالب کلی مسائل جدید یاد گرفتم از سایتتون.
یه سوال داشتم اگر براتون مقدوره راهنماییم کنید ممنون : تو این خاصیت راهبری کوئری ایجاد شده به صورت inner join درمیاد حالا ما اگه بخوایم left - right outer join یا حتی full join بشه کوئریمون به چه صورت باید عمل کنیم؟ اصلا با خاصیت راهبری EF میشه همچین کاری رو انجام داد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۱ ۲۱:۶

اینجا بحث شده: « شبیه سازی outer Join در entity framework »

نویسنده: saeed
تاریخ: ۱۳۹۲/۱۰/۰۹ ۱۷:۵۲

سلام؛ میشه منظورتون رو در مورد خواص راهبری بگید ؟ یعنی به چی میگن خواص راهبری ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۹ ۱۷:۵۹

خاصیتی که یک entity را به entity دیگر وصل می‌کند؛ navigation property
در مثال بالا خاصیت‌هایی که به صورت virtual تعریف شدند، از این دست هستند.

نویسنده: سعید
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۴:۴

سلام؛ در یک رابطه many-to-many چطور میشه اطلاعات رو واکشی کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۰ ۱۴:۸

« بررسی تفصیلی رابطه Many-to-Many در EF Code first »

نویسنده: فخاری
تاریخ: ۱۳۹۳/۰۴/۰۵ ۱۸:۴۸

با سلام
اگه یک کلاس مخاطب با کد زیر باشه:

```
public class Contact
    {
        public int ContactId { get; set; }
        public string FName { get; set; }
        public string LName { get; set; }
        public string FatherName { get; set; }
        public string Email { get; set; }
       public virtual ICollection<Phone> Phones { get; set; }
}
```

و یک کلاس هم برای شماره تلفن‌ها با کد زیر:

```
public class Phone
    {
        public int PhoneId { get; set; }
        public string PhoneNumber { get; set; }
        public string PhoneNote { get; set; }
        public string PhoneAddress { get; set; }
        public int PhoneTypeId { get; set; }
        public virtual PhoneType PhoneType { get; set; }

        [ForeignKey("ContactId")]
        public virtual Contact Contact { get; set; }
        public int ContactId { get; set; }
    }
```

حالا در زمان جستجو من از کد زیر استفاده نموده ام :

```
var listContacts = db.Contacts.Include(p => p.Phones).AsQueryable();
            if (searchContact.ByName)
                listContacts = listContacts.Where(c => c.LName.Contains(searchContact.Name));
            if (searchContact.ByNumber)
            {
                listContacts = listContacts.Where(c=>c.);

            }
            var phonelistmodel = await
                 listContacts.OrderBy(p => p.ContactId)
                     .Skip(page * count)
                     .Take(count)
                     .Select(c => new ListPhoneNumberViewmodel()
                     {
                         ContactId = c.ContactId,
                         Email = c.Email,
                         Name = c.FName + " " + c.LName,
                         Phones = c.Phones
                     }).ToListAsync();
```

ولی اصلا به اطلاعات جدول phone دسترسی ندارم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۰۵ ۲۰:۴

- از Any استفاده کنید، برای رسیدن به لیست اشخاص:

```
listContacts = listContacts.Where(c => c.Phones.Any(x => x.PhoneNumber == "1234....."));
```

- قبل از where یک SelectMany قرار دهید، برای رسیدن به لیست تلفن‌ها:

```
listContacts.SelectMany(c=>c.Phones).Where(c=>c.PhoneNumber=="123....")
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۳ ۰:۵۹

-1برای این کوئیری‌ها چطور از سطح دوم کش استفاده کنیم؟
-2 برای تبدیل به ویو مدل مورد نظر در کدام لایه تبدیلات انجام شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۳ ۱:۲۳

- مانند قبل
- در همان لایه سرویس

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۳ ۲۱:۲۰

منم دقیقا همین کارو کردم اما به این خطا برخورد کردم. پس از رفع خطا با روش معرفی شده، این دفعه با این خطا مواجه میشم:

```
The entity or complex type 'PWS.DataLayer.Context.Tag' cannot be constructed in a LINQ to Entities
query.
```

کوئری منم اینه

```
return tags.Cacheable(x => x.Select(item => new Tag
            {
                Id = item.Id,
                ArticlesCount = item.Articles.Count(),
                Name = item.Name,
                CreatedBy = item.CreatedBy,
                CreatedOn = item.CreatedOn,
                ModifiedBy = item.ModifiedBy,
                ModifiedOn = item.ModifiedOn
            })).ToList();
```

که در اون خصیصه ArticlesCount با NotMapped مزین شده و قراره تعداد مقالات اون تگ توش قرار بگیره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۳ ۲۱:۳۷

این خطای خود EF هست (^). به این معنا که در LINQ to Entities مجاز نیستید در حین projection، از کلاس‌هایی که به جداول بانک اطلاعاتی نگاشت شده‌اند استفاده کنید. از یک ViewModel یا یک DTO استفاده کنید تا مشکل برطرف شود. اطلاعات بیشتر

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۰/۰۴ ۱۵:۲۹

سلام
این روش استفاده کردم با استفاده از یک ویو مدل اما اشکالی که پیش میامد این بود که در صورت تغییر در مدل‌های اصلی حافظه کش خالی نمی‌شد. پس از بررسی به این نتیجه رسیدم چون ویو مدل در زمان ثبت در حافظه کش rootKey متفاوتی نسبت به DBSET ایجاد میکرد و در زمان تغییرات حافظه کش پاک نمی‌شد. در پیاده سازی کش سطح دوم یک فیلد RootKey اضافه کردم به صورت اپشنال، در صورتی که می‌خواستیم روت کی دستی تعیین کنیم به مشکل بر نخوریم در نتیجه مشکل نا معتبر کردن کش هم حل شد.

عنوان: مرتب سازی رکوردها به صورت اتفاقی در Entity framework
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۷ ۹:۵۸
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

یکی از انواع روش‌هایی که در SQL Server و مشتقات آن برای نمایش رکوردها به صورت اتفاقی مورد استفاده قرار می‌گیرد، استفاده از کوئری زیر است:

```
SELECT * FROM table
ORDER BY NEWID()
```

سؤال: ترجمه و معادل کوئری فوق در Entity framework به چه صورتی است؟

پاسخ:

یک مثال کامل را در این زمینه در ادامه ملاحظه می‌کنید:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;

namespace Sample
{
    public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }

    public class MyContext : DbContext
    {
        public DbSet<User> Users { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            context.Users.Add(new User { Name = "User 1", Age = 20 });
            context.Users.Add(new User { Name = "User 2", Age = 25 });
            context.Users.Add(new User { Name = "User 3", Age = 30 });
            context.Users.Add(new User { Name = "User 4", Age = 35 });
            context.Users.Add(new User { Name = "User 5", Age = 40 });
            base.Seed(context);
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());

            using (var context = new MyContext())
            {
               var randomListOfUsers =
                        context.Users
                               .Where(person => person.Age >= 25 && person.Age < 40)
                               .OrderBy(person => Guid.NewGuid())
                               .ToList();

               foreach (var person in randomListOfUsers)
                   Console.WriteLine("{0}:{1}", person.Name, person.Age);
            }
        }
    }
```

```
}
```

تنها نکته مهم آن سطر ذیل است که برای مرتب سازی اتفاقی استفاده شده است:

```
.OrderBy(person => Guid.NewGuid())
```

که معادل

```
ORDER BY NEWID()
```

در SQL Server است.

خروجی SQL تولیدی کوئری LINQ فوق را نیز در ادامه مشاهده می‌کنید:

```
SELECT
[Project1].[Id] AS [Id],
[Project1].[Name] AS [Name],
[Project1].[Age] AS [Age]
FROM ( SELECT
NEWID() AS [C1], ------ Guid created here
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
[Extent1].[Age] AS [Age]
FROM [dbo].[Users] AS [Extent1]
WHERE ([Extent1].[Age] >= 25) AND ([Extent1].[Age] < 40)
)  AS [Project1]
ORDER BY [Project1].[C1] ASC  ------ Used for sorting here
```

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۳:۳۰

مرسی بابت مطلب خوبتون
اما کاربرد این روش مرتب سازی در کجاست؟
به شخصه اولین بار است که این روش مرتب سازی را میبینم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۳:۳۲

یک نمونه از کاربرد آن

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۳:۳۶

مرسی
جالب بود
دقت نکرده بودم

نویسنده: Mohsen
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۸:۱۵

تشکر از مطالب زیباتون

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۱/۰۷/۰۷ ۱۸:۳۸

مرسی واقعا مفید بود.

نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۱:۴۳

این روش رو جداول حجیم سرعت رو میاره پایین تو محیط sql میشه از TABLESAMPLE استفاده کرد که متاسفانه معادلی براش تو linq نیست و...
آقای نصیری یه روش هم به این صورته

```
SELECT * FROM Table1
  WHERE (ABS(CAST(
  (BINARY_CHECKSUM(*) *
  RAND()) as int)) % 100) < 10
```

میخواستم بدونم این روش رو میشه به linq تبدیل کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۲:۴۴

خیر. کوئری‌های EF طوری طراحی شدن که قابل انتقال به بانک‌های اطلاعاتی دیگر هم باشند و بتونید با تغییر کانکشن استرینگ بدون نگرانی آنچنانی، واقعا از یک بانک اطلاعاتی دیگر استفاده کنید.
در EF می‌شود raw sql هم نوشت: (^). در این حالت برنامه شما صرفا به بانک اطلاعاتی مورد استفاده گره خواهد خورد و اگر

مثلا این BINARY_CHECKSUM در SQLite وجود خارجی نداشت، این برنامه و سیستم دیگر قابل انتقال نخواهند بود.

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۶:۰

یه سری دستورات هستند که EF معادلی براشون توی SQL نداره به طور مثال EF نمی‌دونه که متد Parse رو در SQL به چی تبدیل کنه،به طور مثال کد زیر :

```
var query = (from list in dbContext.Packages
                            where list.Id == Int32.Parse(Request["Id"].ToString())
                            select list).FirstOrDefault();
```

باید به صورت زیر تغییر بدیم :

```
Int32 ID = Int32.Parse(Request["Id"].ToString());
 var query = (from list in dbContext.Packages
                            where list.Id == ID
                            select list).FirstOrDefault();
```

به نظرتون توی نسخه‌های بعد این مشکلات رو برطرف می‌کنن؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۶:۳۹

لطفا از موضوع بحث که مرتب سازی اتفاقی است خارج نشید.
در مورد پیشنهادها و انتقادهای مرتبط با EF می‌تونید به اینجا مراجعه کنید: (^)

نویسنده: احسان آرک
تاریخ: ۱۳۹۲/۱۰/۲۱ ۱۹:۲

با سلام
آیا میشود جدولی که هر آیتم آن میتواند لیستی از همین جدول را داشته باشد در entity مرتب کرد؟ مانند یک لیست تو در تو.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۱ ۱۹:۳۰

در مطلب « مباحث تکمیلی مدل‌های خود ارجاع دهنده در EF Code first » بعد از Where (که نوشته for TreeViewHelper) یک OrderBy قرار بدید. نمونه‌اش همین مرتب سازی کامنت‌های تو در توی سایت جاری است بر اساس Id آن‌ها از بالا به پایین ذیل هر مطلب.

عنوان: **استفاده از خواص راهبری در EF Code first جهت ساده سازی کوئری‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۴ ۲۳:۴۷
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

گاهی از اوقات یافتن معادل LINQ کوئری‌های SQLایی که پیشتر به سادگی و بر اساس ممارست، در کسری از دقیقه نوشته می‌شدند، آنچنان ساده نیست. برای مثال فرض کنید یک سری پروژه وجود دارند که به ازای هر پروژه، تعدادی بازخورد ثبت شده است. هر بازخورد نیز دارای وضعیت‌هایی مانند «در حال انجام» و «انجام شد» است. می‌خواهیم کوئری LINQ سازگار با EF ایی را تهیه کنیم که تعداد موارد «در حال انجام» را نمایش دهد.

بر این اساس، کلاس‌های مدل دومین مساله به صورت زیر خواهند بود:

```
public class Project
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<ProjectIssue> ProjectIssues { set; get; }
    }

    public class ProjectIssue
    {
        public int Id { set; get; }
        public string Body { set; get; }

        [ForeignKey("ProjectStatusId")]
        public virtual ProjectIssueStatus ProjectIssueStatus { set; get; }
        public int ProjectStatusId { set; get; }

        [ForeignKey("ProjectId")]
        public virtual Project Project { set; get; }
        public int ProjectId { set; get; }
    }

    public class ProjectIssueStatus
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<ProjectIssue> ProjectIssues { set; get; }
    }
```

یک پروژه می‌تواند تعدادی Issue ثبت شده داشته باشد. هر Issue نیز دارای وضعیتی مشخص است.

اگر EF Code first را وادار به تهیه جداول و روابط معادل کلاس‌های فوق کنیم:

```
public class MyContext : DbContext
    {
        public DbSet<ProjectIssueStatus> ProjectStatus { get; set; }
        public DbSet<ProjectIssue> ProjectIssues { get; set; }
        public DbSet<Project> Projects { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            var project1 = new Project { Name = "پروژه جدید" };
            context.Projects.Add(project1);

            var stat1 = new ProjectIssueStatus { Name = "درحال انجام" };
            var stat2 = new ProjectIssueStatus { Name = "انجام شد" };
            context.ProjectStatus.Add(stat1);
            context.ProjectStatus.Add(stat2);

            var issue1 = new ProjectIssue
            {
```

```
                Body = "تغییر قلم گزارش",
                ProjectIssueStatus = stat1,
                Project = project1
            };
            var issue2 = new ProjectIssue
            {
                Body = "تغییر لوگوی گزارش",
                ProjectIssueStatus = stat1,
                Project = project1
            };
            context.ProjectIssues.Add(issue1);
            context.ProjectIssues.Add(issue2);

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

به شکل زیر خواهیم رسید:

سابقا برای یافتن تعداد متناظر با هر IssueStatus خیلی سریع می‌شد چنین کوئری را نوشت:

```
1  SELECT [Id],
2         [Name],
3         InUseCount = (
4             SELECT COUNT(*)
5             FROM   ProjectIssues
6             WHERE  ProjectStatusId = [ProjectIssueStatus].id
7         )
8  FROM   [ProjectIssueStatus]
```

Results | Messages

	Id	Name	InUseCount
1	1	درحال انجام	2
2	2	انجام شد	0

اما اکنون معادل آن با EF Code first چیست؟

```
public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());

            using (var ctx = new MyContext())
            {
                var projectId = 1;
                var list = ctx.ProjectStatus.Select(x => new
                                                    {
                                                        Id = x.Id,
                                                        Name = x.Name,
                                                        Count = x.ProjectIssues.Count(p => p.ProjectId
== projectId)
                                                    }).ToList();
                foreach (var item in list)
                    Console.WriteLine("{0}:{1}",item.Name, item.Count);
            }
        }
    }
```

بله. همانطور که ملاحظه می‌کنید در اینجا به کوئری بسیار ساده و واضحی با کمک استفاده از navigation properties (خواص راهبری مانند ProjectIssues) تعریف شده رسیده‌ایم. خروجی SQL تولید شده توسط EF نیز به صورت زیر است:

```
SELECT [Project1].[Id] AS [Id],
       [Project1].[Name] AS [Name],
       [Project1].[C1] AS [C1]
FROM   (
           SELECT [Extent1].[Id] AS [Id],
                  [Extent1].[Name] AS [Name],
                  (
                      SELECT COUNT(1) AS [A1]
                      FROM   [dbo].[ProjectIssues] AS [Extent2]
                      WHERE  ([Extent1].[Id] = [Extent2].[ProjectStatusId])
```

```
                                AND ([Extent2].[ProjectId] = 1 /*@p__linq__0*/)
                        ) AS [C1]
                FROM   [dbo].[ProjectIssueStatus] AS [Extent1]
            ) AS [Project1]
```

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۷/۰۵ ۷:۲۱

وای، منظورتون Navigation Property بود؟ نیم ساعته دارم فکر می‌کنم خواص راهبری دیگه چیه؟

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۰۵ ۱۶:۸

در بسیاری از مثالهای این سایت از IList استفاده کردید و در این مثال از ICollection . فرق اینها دقیقا در چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۵ ۱۷:۴۳

IEnumerable فقط خواندنی است.
ICollection یک IEnumerable است که قابلیت Add و Remove به آن اضافه شده.
IList یک ICollection است که به اعضای آن از طریق ایندکس‌ها می‌توان دسترسی اتفاقی داشت.
در تمام مثال‌های EF Code first این سایت از ICollection برای معرفی خواص راهبری استفاده شده چون EF برای انجام اعمال داخلی خودش به مجموعه‌هایی که قابلیت افزودن یا حذف عناصر را داشته باشند، نیاز دارد. به علاوه اگر به سورس EF هم مراجعه کنید برای تشخیص روابط بین کلاس‌ها به دنبال ICollection می‌گردد.
همچنین رسم است حین انتخاب اینترفیس‌هایی از این دست که از هم مشتق می‌شوند، روال انتخاب «the least specific abstraction» رعایت شود. یعنی انتخاب کوچکترین پیاده سازی با حداقل نیازهایی که کاربرد مورد نظر را برآورده می‌کند.

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۷/۰۶ ۰:۱۴

مرسی
خیلی کامل بود.

نویسنده: Alex
تاریخ: ۱۳۹۱/۰۷/۰۶ ۰:۲۳

میتونم بپرسم چرا از toList استفاده کردید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۶ ۰:۴۵

این یک عادت خوب در EF است. زمانیکه خروجی کار شما IEnumerable باشد، هر بار دسترسی به نتیجه آن، یکبار رفت و برگشت به بانک اطلاعاتی را سبب خواهد شد. برای نمونه در مثال زیر دوبار رفت و برگشت به بانک اطلاعاتی خواهیم داشت (یکبار در حلقه اول و یکبار در حلقه دوم). اما با استفاده از ToList فقط یکبار رفت و برگشت صورت گرفته و اطلاعات اصطلاحا materialized خواهند شد.

```
var list = ctx.ProjectStatus.Select(...);

foreach (var item in list)
{...}

foreach (var item in list)
{...}
```

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۱/۰۷/۰۶ ۱۸:۲۳

شمای edmxی برای درک بهتر.

نویسنده: alireza
تاریخ: ۱۳۹۱/۰۸/۰۲ ۰:۲۸

لطفا راهنمایی کنید برای اینکه ببینیم ef برای یک کوئری linq چه عبارت sql ایی تولید میکند و آن را اجرا میکند، چه کاری باید انجام داد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۲ ۰:۵۱

نحوه‌ی مشاهده‌ی خروجی SQL تولید شده توسط WCF RIA Services

نویسنده: ایمان باقری
تاریخ: ۱۳۹۳/۰۱/۲۷ ۱۰:۴۴

من وقتی از IList استفاده میکنم برای تعریف خواص راهبری زمانی که اون رو به یک گرید بایند میکنم AddNewRow گرید کار

نمیکنه(devExpress)با جستجو و تعریف bindingList به جای لیست مشکل حل شد.
میخواستم ببینم تعریف خواص راهبری از نوع bindingList مشکلی ندارد؟
فقط binginList متد AddRange ندارد و برای اضافه کردن چند لیست به آن باید از foreach استفاده کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۷ ۱۱:۱۱

این‌ها بیشتر مباحث binding دو طرفه است. تیم EF هم یک ObservableListSource را برای برنامه‌های WinForm تدارک دیده: اینجا . برای WPF هم اینجا

نویسنده: آحمد
تاریخ: ۱۳۹۳/۰۶/۳۱ ۱۹:۴۳

با سلام
من یه سوال داشتم، این سوالم مربوط میشه به جدول هایی که توی اینجا ساختید
میخواستم بدونم بهینه‌ترین نوع دستور توی EF برای گرفتن خروجی زیر چی میتونه باشه :
گرفتن FirstName و LastName کاستومر هایی که در نقش با Name ادمین هستند!
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۳۱ ۲۰:۴۵

در انتهای مطلب « بررسی تفصیلی رابطه Many-to-Many در EF Code first » در این مورد بحث شده.

عنوان: **یافتن مقادیر نال در Entity framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۶ ۲۱:۲۱
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

کلاس شخص زیر را درنظر بگیرید

```
public class Person
{
        public int Id { get; set; }
        public string Name { get; set; }
        public int? Age { get; set; }
}
```

در اینجا با توجه به اینکه Name از نوع string است، خودبخود به فیلدی نال‌پذیر نگاشت خواهد شد و همچنین Age عددی نیز در سمت کدهای ما Nullable است، بنابراین خاصیت سن هم به فیلدی نال‌پذیر نگاشت می‌شود.
اگر تمام مراحل متداول ایجاد Context را طی کنیم، به نظر شما خروجی SQL عبارت زیر چه خواهد بود؟

```
string name = null;
var list1 = ctx.Users.Where(x => x.Name == name).ToList();
```

در این عبارت، name به صورت یک متغیر ارسال شده است و نه یک مقدار ثابت (فرض کنید یک متد را تعریف کرده‌اید که name را به صورت پارامتر دریافت می‌کند).
خروجی SQL آن به نحو زیر است:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
[Extent1].[Age] AS [Age]
FROM [dbo].[People] AS [Extent1]
WHERE [Extent1].[Name] = @p__linq__0
-- p__linq__0 (dbtype=String, size=-1, direction=Input) = null
```

به عبارتی خروجی مورد انتظار name **is** null را تولید نکرده است و کوئری ما حداقل با SQL Server نتیجه‌ای را به همراه نخواهد داشت. در مورد Age نیز به همین صورت است.

راه حل:

برای حالت Age، روش زیر خروجی age is null را تولید می‌کند:

```
var list2 = ctx.Users.Where(x => !x.Age.HasValue).ToList();
```

و یا استفاده از object.Equals نیز مشکل را برطرف خواهد کرد:

```
int? age = null;
var list2 = ctx.Users.Where(x => object.Equals(x.Age, age)).ToList();
```

برای حالت Name رشته‌ای می‌توان از روش زیر استفاده کرد:

```
var list1 = ctx.Users.Where(x => string.IsNullOrEmpty(x.Name)).ToList();
```

و یا روش کلی‌تر زیر نیز جواب می‌دهد:

```
string name = null;
```

```
var list1 = ctx.Users.Where(x => name == null ? x.Name == null : x.Name == name).ToList();
```

کاری که در اینجا انجام شده استفاده از x.Name == null در حالت نال بودن name است. از این جهت که EF با کوئری ذیل به علت عدم استفاده از پارامتر برای معرفی مقداری نال، مشکلی ندارد :

```
var list1 = ctx.Users.Where(x => x.Name == null).ToList();
```

نظرات خوانندگان

نویسنده: نیما
تاریخ: ۱۳۹۱/۰۶/۲۸ ۹:۴۷

سلام آقای نصیری

ممنون از مطلب مفیدتون راستش من به این نکته ای که فرمودین توجه نکرده بودم. من دستوراتی را روی دیتابیس northwind اجرا کردم که اولین دستور به این صورت بود:

```
rg = ent.regions.where(x => x.regionid != null).tolist();
```

این دستور در sql server به اینصورت تبدیل میشود :

```
select
[extent1].[regionid] as [regionid],
[extent1].[regiondescription] as [regiondescription]
from [dbo].[region] as [extent1]
```

آیا اینکه اصلا در دستور sql ما چک کردن برای null نداریم به این خاطر است که ef بطور اتوماتیک چک میکند که فیلد regionid از نوع nullable هست و چنانچه نباشه اصلا شرط رو دخالت نمیده؟

من در گام بعدی این دستور را اجرا کردم :

```
rg = ent.regions.where(x => x.regiondescription == null).tolist();
```

که خروجی آن به این صورت بود:

```
select
cast(null as int) as [c1],
cast(null as varchar(1)) as [c2]
from  ( select 1 as x ) as [singlerowtable1]
where 1 = 0
```

میخواستم اگر ممکنه کمی راجع به این دستور توضیح بفرمایید آیا این دستور همون کار is null رو انجام میده یا خیر.

باز هم سپاسگزارم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۸ ۹:۵۹

خروجی SQL شما منطبق با خروجی SQL حاصل از EF نیست. روش کار را اینجا توضیح دادم که چگونه می‌شود این خروجی را دقیقا به دست آورد.
در حالت

```
var list1 = ctx.Users.Where(x => x.Name != null).ToList();
```

این خروجی حاصل می‌شود:

```
SELECT
```

```
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
[Extent1].[Age] AS [Age]
FROM [dbo].[People] AS [Extent1]
WHERE [Extent1].[Name] IS NOT NULL
```

در حالت

```
var list2 = ctx.Users.Where(x => x.Name == null).ToList();
```

دقیقا این خروجی را خواهیم داشت:

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
[Extent1].[Age] AS [Age]
FROM [dbo].[People] AS [Extent1]
WHERE [Extent1].[Name] IS NULL
```

نویسنده: lp
تاریخ: ۱۳۹۱/۰۶/۲۹ ۱۴:۲۱

ممنون از توضیحتون

نویسنده: debuger
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۱:۰

با سلام
ببخشید آیا امکان پیاده سازی تابع isnull هم توسط EF هست ؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۱:۲۵

- از عملگر ?? استفاده کنید تا با تمام بانک‌های اطلاعاتی سازگار باشد.
+ یک سری متد SQL خاص هم در EF وجود دارند که البته وابسته‌اند به بانک اطلاعاتی مورد استفاده و قابل استفاده در عبارات LINQ.

عنوان: **واکشی اطلاعات به صورت chunk chunk (تکه تکه) و نمایش در ListView**
نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۳:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, Entity framework

یکی از مسائلی که در هنگام کار با کنترلهای داده‌ای نظیر GridView , ListView و .. با آن روبرو هستیم مسئله صفحه بندی می‌باشد و در بسیاری از موارد، کل اطلاعات در هر درخواست، بارگذاری میشود. در حالیکه روش بهینه به این صورت است که با توجه به PageSize و Index رکورد، می‌توان تعداد رکورد مورد نظر در همان صفحه را بارگذاری کرد، نه کل رکوردها را.
در این مثال که از Ef Code First و الگوی Unit Of Work استفاده کرده ام ، قصد نمایش اطلاعات در یک ListView و صفحه بندی آن را بصورت chunk chunk دارم.
برای آشنایی بیشتر با الگوی Unit Of Work می‌توانید به مقاله UOW در همین سایت مراجعه کنید.

ابتدا یک کنترل ListView روی صفحه ایجاد می‌کنیم. برای آشنایی بیشتر با این کنترل و بررسی قابلیتهای آن میتوان از این مقاله معرفی کنترل‌های ListView و DataPager استفاده کنید.

سپس با توجه به الگوی Unit Of Work از یک مدل ساده استفاده می‌کنیم. همچنین برای بارگذاری اطلاعات به صورت صفحه به صفحه، نیاز به داشتن Index رکورد و PageSize، جهت محاسبه تعداد رکورد مورد نیاز داریم.

```
public class User
    {
        public Int64 UserId { get; set; }
        public String UserName { get; set; }
     }
```

```
public interface IUserService
    {
        int GetCustomerCount();
        List<User> GetCustomers(int StartIndex, int PageSize);
    }
```

```
public class ImplUserService : IUserService
    {
        IUnitOfWork _uow;
        IDbSet<User> _user;

        public ImplUserService(IUnitOfWork uow)
        {
            _uow = uow;
            _user = uow.Set<User>();
        }

        public int GetCustomerCount()
        {
            int totalCount = _user.ToList().Count;
            return totalCount;
        }

        public List<User> GetCustomers(int StartIndex, int PageSize)
        {
            return _user.OrderBy(i => i.UserId).Skip(StartIndex).Take(PageSize).ToList();
        }
  }
```

در کدهای بالا متد GetCustomerCount تعداد کل رکوردهایی که باید واکشی شوند را مشخص می‌کند. همچنین می‌توان این تعداد را در cache یا viewState ذخیره کرد تا در دفعات بعدی در صورت خالی نبودن مقدار Cacheیا viewState ، نیاز به محاسبه مجدد count نداشته باشیم.

متد GetCustomer که کار اصلی را انجام میدهد از دو پارامتر استفاده می‌کند: StartIndex نقطه شروع و PageSize تعداد رکورد مورد نظر. در اینجا از دستورات Linq استفاده شده و دستور Skip مشخص میکند از کدام شماره رکورد به بعد شروع به واکشی

نماید و دستور Take مشخص می‌کند که چه تعداد رکورد را واکشی نماید.

حالا به سراغ کدهای HTML می‌رویم. در آنجا علاوه بر ListView نیاز به DataPager جهت صفحه بندی و Object DataSource جهت کنترل بارگذاری اطلاعات به صورت chunk chunk داریم.

```
<asp:ListView ID="ListView1" runat="server" DataSourceID="ObjectDataSource1">
            <ItemTemplate>
                <tr style="background-color: #DCDCDC; color: #000000;">
                    <td>
                        <asp:Label ID="UserIdLabel" runat="server" Text='<%# Eval("UserId") %>' />
                    </td>
                    <td>
                        <asp:Label ID="UserNameLabel" runat="server" Text='<%# Eval("UserName") %>' />
                    </td>
                </tr>
            </ItemTemplate>


            <LayoutTemplate>
                <table runat="server">
                    <tr runat="server">
                        <td runat="server">
                            <table id="itemPlaceholderContainer" runat="server" border="1"
style="background-color: #FFFFFF;
                                border-collapse: collapse; border-color: #999999; border-style: none;
border-width: 1px;">
                                <tr runat="server" style="background-color: #DCDCDC; color: #000000;">
                                    <th runat="server">
                                        UserId
                                    </th>
                                    <th runat="server">
                                        UserName
                                    </th>
                                </tr>
                                <tr id="itemPlaceholder" runat="server">
                                </tr>
                            </table>
                        </td>
                    </tr>
                    <tr runat="server">
                        <td runat="server" style="text-align: center; background-color: #CCCCCC;
                            color: #000000;">
                            <asp:DataPager ID="DataPager1" runat="server" PageSize="2">
                                <Fields>
                                    <asp:NextPreviousPagerField ButtonType="Button"
ShowFirstPageButton="True" ShowNextPageButton="False"
                                        ShowPreviousPageButton="False" />
                                    <asp:NumericPagerField />
                                    <asp:NextPreviousPagerField ButtonType="Button"
ShowLastPageButton="True" ShowNextPageButton="False"
                                        ShowPreviousPageButton="False" />
                                </Fields>
                            </asp:DataPager>
                        </td>
                    </tr>
                </table>
            </LayoutTemplate>
        </asp:ListView>
```

همانطور که مشاهده میکنید در DataPager ، مقدار PageSize مشخص شده است.اما Object DataSource

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" EnablePaging="True"
SelectCountMethod="GetCustomerCount"
            SelectMethod="GetCustomers" TypeName="UserService.ImplUserService" EnableViewState="False"
            MaximumRowsParameterName="PageSize" StartRowIndexParameterName="StartIndex">
```

که فکر می‌کنم با توجه به متدهای تعریف شده در بالا ، تعریف Object DataSource کاملاً گویا میباشد.

نکته مهم:

اگر الان برنامه را اجرا کنید با خطای No parameterless constructor defined for this object ر وبرو خواهید شد که جهت

حل این مشکل از کد زیر استفاده میکنیم.

```
protected void ObjectDataSource1_ObjectCreating(object sender, ObjectDataSourceEventArgs e)
        {
            e.ObjectInstance = ObjectFactory.GetInstance<IUserService>();
        }
```

در واقع نیاز است تا یک وهله از کلاس مورد نظر را به Object DataSource معرفی کنیم.
حال با اجرای برنامه و Trace آن متوجه خواهید شد که با کلیک بر روی شماره صفحه، تنها به تعداد رکوردهای همان صفحه، واکشی خواهیم داشت.

با تشکر از راهنمایی‌های آقای نصیری، امیدوارم این مقاله برای دوستان مفید باشه. منتظر نظرات دوستان هستم و اینکه چه جوری بتونیم اینکار رو با jquery ajax و به صورت سبکتر انجام بدیم.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۴:۲۵

یک نکته تکمیلی: در اینجا بهتر است جهت محاسبه تعداد رکوردها از متد Count مستقیما بر روی dbSet استفاده کنید (نیازی به ToList ندارد).

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۴:۳۴

بله کاملاً درسته
ممنون مهندس.

نویسنده: علیرضا اسم‌رام
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۵:۵۵

سلام و تشکر از شما بخاطر این مطلب. پیشنهاد می‌کنم جهت توسعه این ترفند (همانطور که اشاره کردید) نگاهی به + و + و + بیاندازید.
موفق باشید.

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۴/۳۰ ۱۶:۳۰

با سلام و تشکر از شما بابت نظرتون
scrolling هم راه خوبیه برای اینکار که کاربردهای زیادی داره
ممنونم.

عنوان: **امنیت در LINQ to SQL**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۱۵ ۱۳:۰۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Security

سؤال: LINQ to SQL تا چه میزان در برابر حملات تزریق SQL امن است؟
جواب کوتاه: بسیار زیاد!

توضیحات:

```
string query = @"SELECT * FROM USER_PROFILE
              WHERE LOGIN_ID = '"+loginId+@"' AND PASSWORD = '"+password+@"'";
```

گاهی از اوقات هر چقدر هم در مورد خطرات کوئری‌هایی از نوع فوق مقاله نوشته شود کافی نیست و باز هم شاهد این نوع جمع زدن‌ها و نوشتن کوئری‌هایی به شدت آسیب پذیر در حالت استفاده از ADO.Net کلاسیک هستیم. مثال فوق یک نمونه کلاسیک از نمایش آسیب پذیری در مورد تزریق اس کیوال است. یا نمونه‌ی بسیار متداول دیگری از این دست که با ورودی خطرناک می‌تواند تا نمایش کلیه اطلاعات تمامی جداول موجود هم پیش برود:

```
protected void btnSearch_Click(object sender, EventArgs e)
{
 String cmd = @"SELECT [CustomerID], [CompanyName], [ContactName]
  FROM [Customers] WHERE CompanyName ='" + txtCompanyName.Text
  + @"'";

 SqlDataSource1.SelectCommand = cmd;

 GridView1.Visible = true;
}
```

در اینجا فقط کافی است مهاجم با تزریق عبارت SQL مورد نظر خود، کوئری اولیه را کاملا غیرمعتبر کرده و از یک جدول دیگر در سیستم کوئری تهیه کند!
راه حلی که برای مقابله با آن در دات نت ارائه شده نوشتن کوئری‌های پارامتری است و در این حالت کار encoding اطلاعات ورودی به صورت خودکار توسط فریم ورک مورد استفاده انجام خواهد شد؛ همچنین برای مثال اس کیوال سرور، execution plan این نوع کوئری‌های پارامتری را همانند رویه‌های ذخیره شده، کش کرده و در دفعات آتی فراخوانی آن‌ها به شدت سریعتر عمل خواهد کرد. برای مثال:

```
SqlCommand cmd = new SqlCommand("SELECT UserID FROM Users WHERE UserName=@UserName AND
Password=@Password");
cmd.Parameters.Add(new SqlParameter("@UserName", System.Data.SqlDbType.NVarChar, 255, UserName));
cmd.Parameters.Add(new SqlParameter("@Password", System.Data.SqlDbType.NVarChar, 255, Password));
dr = cmd.ExecuteReader();
if (dr.Read()) userId = dr.GetInt32(dr.GetOrdinal("UserID"));
```

زمانیکه از کوئری پارامتری استفاده شود، مقدار پارامتر، هیچگاه فرصت و قدرت اجرا پیدا نمی‌کند. در این حالت صرفا به آن به عنوان یک مقدار معمولی نگاه خواهد شد و نه جزء قابل تغییر بدنه کوئری وارد شده که در حالت جمع زدن رشته‌ها همانند اولین کوئری معرفی شده، تا حد انحراف کوئری به یک کوئری دلخواه مهاجم قابل تغییر است.

اما در مورد LINQ to SQL چطور؟
این سیستم به صورت پیش فرض طوری طراحی شده است که تمام کوئری‌های SQL نهایی حاصل از کوئری‌های LINQ نوشته شده توسط آن، پارامتری هستند. به عبارت دیگر این سیستم به صورت پیش فرض برای افرادی که دارای حداقل اطلاعات امنیتی هستند به شدت امنیت بالایی را به همراه خواهد آورد.

برای مثال کوئری LINQ زیر را در نظر بگیرید:

```
var products = from p in db.products
      where p.description.StartsWith(_txtSearch.Text)
      select new
      {
          p.description,
          p.price,
          p.stock

      };
```

اکنون فرض کنید کاربر به دنبال کلمه sony باشد، آنچه که بر روی اس کیوال سرور اجرا خواهد شد، دستور زیر است (ترجمه نهایی کوئری فوق به زبان T-SQL) :

```
exec sp_executesql N'SELECT [t0].[description], [t0].[price], [t0].[stock]
FROM [dbo].[products] AS [t0]
WHERE [t0].[description] LIKE @p0',N'@p0 varchar(5)',@p0='sony%'
```

برای لاگ کردن این عبارات SQL یا می‌توان از SQL profiler استفاده نمود و یا خاصیت log زمینه مورد استفاده را باید مقدار دهی کرد:

```
db.Log = Console.Out;
```

و یا می‌توان بر روی کوئری مورد نظر در VS.Net یک break point قرار داد و سپس از debug visualizer مخصوص آن استفاده نمود.

همانطور که ملاحظه می‌کنید، کوئری نهایی تولید شده پارامتری است و در صورت ورود اطلاعات خطرناک در پارامتر p0 ، هیچ اتفاق خاصی نخواهد افتاد و صرفا رکوردی بازگشت داده نمی‌شود.

و یا همان مثال کلاسیک اعتبار سنجی کاربر را در نظر بگیرید:

```
public bool Validate(string loginId, string password)
{
 DataClassesDataContext db = new DataClassesDataContext();

 var validUsers = from user in db.USER_PROFILEs
           where user.LOGIN_ID == loginId
           && user.PASSWORD == password
           select user;

 if (validUsers.Count() > 0) return true;
 else return false;
}
```

کوئری نهایی T-SQL تولید شده توسط این ORM از کوئری LINQ فوق به شکل زیر است:

```
SELECT [t0].[LOGIN_ID], [t0].[PASSWORD]
FROM [dbo].[USER_PROFILE] AS [t0]
WHERE ([t0].[LOGIN_ID] = @p0) AND ([t0].[PASSWORD] = @p1)
```

و این کوئری پارامتری نیز در برابر حملات تزریق اس کیوال امن است.

تذکر مهم هنگام استفاده از سیستم LINQ to SQL :

اگر با استفاده از LINQ to SQL مجددا به روش قدیمی اجرای مستقیم کوئری‌های SQL خود همانند مثال زیر روی بیاورید (این امکان نیز وجود دارد)، نتیجه این نوع کوئری‌های حاصل از جمع زدن رشته‌ها، پارامتری "نبوده" و مستعد به تزریق اس کیوال هستند:

```
string sql = "select * from Trade where DealMember='" + this.txtParams.Text + "'";
var trades = driveHax.ExecuteQuery<Trade>(sql);
```

در اینجا باید در نظر داشت که اگر شخصی مجددا بخواهد از این نوع روش‌های کلاسیک استفاده کند شاید همان ADO.Net کلاسیک برای او کافی باشد و نیازی به تحمیل سربار یک ORM را به سیستم نداشته باشد. در این حالت برنامه از type safety کوئری‌های LINQ نیز محروم شده و یک لایه بررسی مقادیر و پارامترها را توسط کامپایلر نیز از دست خواهد داد.

اما روش صحیحی نیز در مورد بکارگیری متد ExecuteQuery وجود دارد. استفاده از این متد به شکل زیر مشکل را حل خواهد کرد:

```
IEnumerable<Customer> results = db.ExecuteQuery<Customer>(
 "SELECT contactname FROM customers WHERE city = {0}", "Tehran");
```

در این حالت، پارامترهای بکارگرفته شده (همان {0} ذکر شده در کوئری) به صورت خودکار به پارامترهای T-SQL ترجمه خواهند شد و مشکل تزریق اس کیوال برطرف خواهد شد (به عبارت دیگر استفاده از +، علامت مستعد بودن به تزریق اس کیوال است و بر عکس).

عنوان: استفاده از خاصیت Local در Entity Framework
نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۵/۰۱ ۱۸:۳۷
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, DbContext, LINQ to Objects

زمانی که از LINQ To Entity استفاده می‌کنیم، با هر بار اجرای یک کوئری، این کوئری به سمت دیتابیس ارسال شده و اطلاعات مورد نظر را بازیابی می‌کند. حال اگر ما موجودیت جدیدی را به Context جاری اضافه کرده ولی آن را ذخیره نکرده باشیم، به علت عدم وجود موجودیت در دیتابیس (در حافظه وجود دارد) کوئری ارسالی ما این موجودیت جدید را شامل نمی‌شود. البته شایان ذکر است زمانیکه از متد Find استفاده می‌کنیم، به صورت پیش فرض ابتدا داخل حافظه کاوش شده و در صورت عدم وجود اطلاعات، کوئری را در دیتابیس اجرا می‌کند.
نکته قابل توجه این است که بعنوان مثال ما نیاز داریم یک لیست از موجودیت‌ها را به اشکال زیر داشته باشیم.
به صورت لیست
به صورت لیست sort شده براساس Name
فیلتر بر روی لیستی از فیلدهای موجود
این لیست باید شامل تمامی داده‌های موجود (چه در رم و چه در دیتابیس) باشد.

نکته: توسط متد ToList میتوان به لیستی از موجودیت‌های مورد نظر دست یافت ولی امکان استفاده از تمامی داده‌های موجود (چه در رم و چه در دیتابیس) میسر نمی‌باشد.
کلاس DbSet خاصیتی به نام Local دارد که امکان استفاده از تمامی داده‌های موجود را به ما می‌دهد و شامل هر داده‌ای که از دیتابیس Load شده، هر داده‌ای که اضافه شده، هر داده‌ای که پاک شده (Delete Flag) ولی هنوز ذخیره نشده می‌شود. بنابراین در هنگام استفاده باید توجه داشت به علت اینکه هیچ نوع کوئری به دیتابیس ارسال نشده، قطعه کد زیر دارای مقدار Destinations in memory: 0 خواهد بود

```
private static void GetLocalDestinationCount()
    {
        using (var context = new BreakAwayContext())
        {
            var count = context.Destinations.Local.Count;
            Console.WriteLine("Destinations in memory: {0}", count);
        }
    }
```

استفاده از متد Load

برای مثال می‌توانیم از Foreach استفاده کنیم و تمام اطلاعات مورد نظر را بدست آوریم

```
private static void GetLocalDestinationCount()
    {
        using (var context = new BreakAwayContext())
        {
            foreach (var destination in context.Destinations)
            {
                Console.WriteLine(destination.Name);
            }
            var count = context.Destinations.Local.Count;
            Console.WriteLine("Destinations in memory: {0}", count);
        }
    }
```

کد بالا یک حلقه بر روی موجودیت‌های Destinations است که نیازی به توضیح خاصی ندارد.
حال با استفاده از متد Load قادر به جمع آوری اطلاعات دیتابیس به داخل رم نیز خواهیم بود و کد بالا تمیزتر خواهد شد.

```
private static void GetLocalDestinationCountWithLoad()
    {
        using (var context = new BreakAwayContext())
        {
            context.Destinations.Load();
            var count = context.Destinations.Local.Count;
            Console.WriteLine("Destinations in memory: {0}", count);
        }
    }
```

متد Load یک extension method روی IQueryable<T> است که در فضای نام System.Data.Entity موجود است. پس امکان اجرای یک LINQ query و سپس Load کردن آن را در حافظه را خواهیم داشت.
به کد زیر توجه کنید:

```
private static void LoadAustralianDestinations()
    {
        using (var context = new BreakAwayContext())
        {
            var query = from d in context.Destinations
                        where d.Country == "Australia"
                        select d;
            query.Load();
            var count = context.Destinations.Local.Count;
            Console.WriteLine("Aussie destinations in memory: {0}", count);
        }
    }
```

همچنین ما قادر به استفاده از LINQ query روی داده‌های Local که این متد در حافظه جمع آوری کرده، نیز خواهیم بود.

به کد زیر توجه کنید.

```
private static void LocalLinqQueries()
    {
        using (var context = new BreakAwayContext())
        {
            context.Destinations.Load();
            var sortedDestinations = from d in context.Destinations.Local
                                     orderby d.Name
                                     select d;
            Console.WriteLine("All Destinations:");
            foreach (var destination in sortedDestinations)
            {
                Console.WriteLine(destination.Name);

            }
        }
        var aussieDestinations = from d in context.Destinations.Local
                                 where d.Country == "Australia"
                                 select d;
        Console.WriteLine();
        Console.WriteLine("Australian Destinations:");
        foreach (var destination in aussieDestinations)
        {
            Console.WriteLine(destination.Name);
        }
    }
```

ابتدا کلیه داده‌ها Load می‌شود سپس به وسیله Foreach نام‌ها استخراج می‌شوند. سپس ار همان داده‌ها جهت اعمال فیلتر، استفاده مشود.

تفاوت بین Linq providerهای مختلف:

عموماً دیتابیسها حساس به حروف کوچک و بزرگ نیستند؛ به عنوان مثال اگر Great و great در دیتابیس وجود داشته باشند اگر به دیتابیس کوئری اسال شود و درخواست great داشته باشد هر دو را شامل میشود. حال اگر از Local استفاده شود به جهت اینکه در واقع از Linq to Object استفاده می‌کند فقط great را شامل خواهد شد.
تفاوت دیگر این است که LINQ to Object از متد Last پشتیبانی می‌کند ولی LINQ to Entities خیر.
در پست بعدی قصد دارم در مورد ObservableCollection توضیحاتی کلی بدهم.

عنوان: ObservableCollection در Entity Framework
نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۵/۰۱ ۱۸:۵۷
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, DbContext

در مبحث ا ستفاده از خاصیت Local در Entity Framework ملاحظه نمودید که خاصیت Local به راحتی می‌تواند از رفت و آمدهای بی جهت به دیتابیس جلوگیری کند.
حال قصد معرفی یک collection را به نام ObservableCollection دارم.
همانطور که از نامش پیداست برای مشاهده و تحت نظر قرار دادن داده‌های اضافه شده یا پاک شده کاربرد دارد. به کد زیر دقت کنید.

```
private static void ListenToLocalChanges()
    {
        using (var context = new BreakAwayContext())
        {
            context.Destinations.Local.CollectionChanged += (sender, args) =>
            {
                if (args.NewItems != null)
                {
                    foreach (Destination item in args.NewItems)
                    {
                        Console.WriteLine("Added: " + item.Name);
                    }
                }
                if (args.OldItems != null)
                {
                    foreach (Destination item in args.OldItems)
                    {
                        Console.WriteLine("Removed: " + item.Name);
                    }
                }
            };
            context.Destinations.Load();
        }
    }
```

در بالا به وسیله یک event handler جدید به collection محلی ما (Local) نظر می‌اندازد و در صورت اضافه شدن یا حذف موجودیتی، آن را به ما نشان می‌دهد. فقط توجه کنید که اگر نیاز دارید در صفحه‌ای این تغییرات را مشاهده کنید باید عمل Refresh کردن صفحه را چه به صورت دستی یا با نوشتن کد خودتان مدیریت کنید. البته با استفاده از WPF میتوان (استفاده از کنترل‌های مانند ListBox) این کار را به صورت خودکار انجام داد.

نظرات خوانندگان

نویسنده: محمد زارع
تاریخ: ۱۳۹۱/۰۵/۰۲ ۰:۱۴

سلام آقای جعفری. ممنون از مطلب مفیدتون. شاید سوالم خیلی ربطی به مطلب شما نداشته باشه ولی میخواستم ازتون بپرسم که ما وقتی توی پروژه WPF از Entity Framework Code First استفاده میکنیم باید اینترفیس INotifyPropertyChanged رو برای هر Entity پیاده سازی کنیم یا نه؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۲ ۸:۱۵

- بله. باید اینکار را انجام دهید.
- استفاده مستقیم از مدل‌های EF در WPF یا MVC یا هر جای دیگری کار توصیه شده‌ای نیست.
View شما باید Model مخصوص به خود را داشته باشد و این مدل الزاما با موجودیت‌های بانک اطلاعاتی شما یکی نیست.
برای نگاشت اطلاعات مدل یک View به مدل داده‌ای می‌شود از کتابخانه‌هایی مانند Auto-Mapper استفاده کرد.

نویسنده: ایلیا
تاریخ: ۱۳۹۱/۰۶/۲۷ ۱۴:۵۹

با سلام.
در پروژه WPF در لایه سرویس یکبار Local رابر میگردانم مانند زیر :

```
public override IList<City> GetAll()
        {
            var query = from item in _tEntities
                        select item;
            query.Load();
            return _tEntities.Local;
        }
```

همه چیز درست است ولی وقتی برای جستجو متد زیر را اجرا می‌کنم باز Local شامل همان داده‌های قبلی است:

```
public override IList<City> GetAll(Func<City, bool> predicate)
        {
            var query = from item in _tEntities
                        select item;
            query.Where<City>(predicate);
            query.Load();
            return _tEntities.Local;
        }
```

لطفاً راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۷ ۱۶:۱۴

در مثال رسمی EF Code first و WPF یک سری Refresh هم هست که باید وقت بگذارید و مطالعه کنید.

نویسنده: kianush
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۳:۲۹

من در winform یک BindingSource دارم و این رو به گرید می‌دم. گرید به یک BindingSource بایند شده، اگه مقادیری رو تغییر بدم یا اضافه کنم در گرید (در واقع به BindingSource پشت صحنه) ، ChangeTracker تشخیص می‌ده و می‌تونم کار رو هندل کنم

اما اگه سطری رو از BindingSource (یا همون گرید) حذف کنم ChangeTracker متوجه نمی‌شه و وضعیتشون Unchanged می‌مونه! (Local هم فایده نداره) و نمی‌تونم کل تغییرات رو ذخیره کنم با صدا زدن SaveChanges
در ChangeTracker ایتم هایی با Stateهای Add, Modified رو می‌تونم ببینم ولی اگه ایتمی رو Delete کنم نمی‌تونم ببینمش و کاری انجام بدم! مشکل از کجاست؟
اینا مربوط به Context هست برای بررسی وضعیت تغییرات:

```
public bool HasChanges()
        {
            return this.ChangeTracker.Entries().Any(e => e.State != EntityState.Unchanged);
        }

        public void RejectChanges()
        {
            foreach (var entry in this.ChangeTracker.Entries())
            {
                switch (entry.State)
                {
                    case EntityState.Modified:
                        entry.State = EntityState.Unchanged;
                        break;

                    case EntityState.Added:
                        entry.State = EntityState.Detached;
                        break;
                }
            }
        }
```

توضیحات بیشتر:
چرا مثل Add , Modify عمل Delete در Entityها و ChangeTracker بصورت خودکار شنیده نمی‌شه؟! من یک کتابخانه‌ی مستقل تهیه کردم که گرید ،خودش یک کنترل BindingSource داره و اصلا نباید به EntityFramework و سرویسهایی که با Entityها کار می‌کنن در ارتباط باشه ، BindingSource باید بتونه CurrentItem اش رو Remove کنه در حافظه مثل کاری که برای Add,Update انجام می‌ده و در انتها من (بعنوان کاربر نهایی) در لایه نمایش تصمیم بگیرم که تمام تغییرات انجام شده در گرید (Add, Modify, Delete) رو ذخیره یا لغو کنم.

******* شاید بشه با کمک رویداد BindingSource.ListChanged به نحوی حل کرد ولی اصلا جالب نمی‌شه چون BindingSource من یک POCO Entity هست و فقط و فقط چنتا property داره و اصلا از وضعیت خودش خبر نداره و قرار هم نیست بیشتر از این باشه و من نمی‌تونم وضعیتش رو تغییر بدم در این رویداد چون همچین چیزی نداره اصلا!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۴:۰

- در مورد Tracking یک مطلب جداگانه در سایت هست. Tracking API همان ObservableCollection نیست. Tracking API در سایر ORMها نامی به شکل سطح اول کش دارد (first level caching).
- با توجه به اینکه برای بررسی کارهای شخصی و کتابخانه‌های مستقل، نیاز به کد کامل هست، بهتر است به مقاله زیر مراجعه کرده و جزئیات کار خودتان را با آن مقایسه کنید:
« Implementing Undo/Redo feature for DbContext of Entity Framework »

نویسنده: kianush
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۴:۳

ممنون.. بررسی می‌کنم ببینم می‌تونم اصولی حل کنم این مسئله رو یا خیر :-؟
ولی یه نکته ، اینکه گفتین "نیاز به کد کامل هست.." اصلا تصور کنین کتابخانه‌ی مستقلی نیست. مثلا یک Form, BindingSource, DataGridView رو داشته باشین و روال بالا که توضیحشو دادم. انگار یک Bug هست! یجورایی که وضعیت "حذف" رو مثل "افزوده شدن" و "تغییر کرده" نمی‌تونه اعلام کنه به دیتاسورس پشت سرش bindingsource

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۴:۲۳

- Tracking API فقط داخل یک Context مفهوم پیدا می‌کند نه مجزای آن.
- همچنین این API دارای متد DetectChanges هم هست که می‌شود به صورت دستی جهت اطمینان بیشتر آن‌را در هر زمانی (مثلا داخل بررسی HasChanges) فراخوانی کرد.

نویسنده: kianush
تاریخ: ۱۳۹۱/۰۸/۰۳ ۱۶:۶

برای این مسئله‌ی من راه حل اصولی ای پیدا نکردم. یه راهی که الان پیاده کردم و جواب گرفتم ولی جالب نیست: در BaseEntity پراپرتی IsDeleted رو کار گذاشتم مثلا یه همچین چیزی فک کنین:

```
public abstract class BaseEntity
    {
        [ColumnInfo("کد",pWidth:70)]
        public int Id { get; set; }

        [ColumnInfo("",pIsVisible:false,pIsEditable:false)]
        [NotMapped]
        public bool IsDeleted { get; set; }
    }
```

و جایی که BindingSource CurrentItem پاک می‌شه ، BindingSource.Current.IsDeleted=true (بصورت dynamic) گذاشتم و در 3 2 ، Context خط دیگه اضافه کردم که این رو هندل کنم برای تمام موجودیت ها.. کار می‌کنه ولی بدیش اینه که یک پراپرتی بی ربط (شاید به نوعی) رو در BaseEntity و در واقع در تمام موجودیتهام تعریف کردم (که البته NotMapped هست) و "رفتار" رو با "خاصیت" قاطی کردم و الان هم عذاب وجدان دارم :دی پ.ن: کماکان دنبال راهی می‌گردم با خوندن مقالات

استفاده از الگوی Repository اضافی در EF Code first؛ آری یا خیر؟!

اگر در ویژوال استودیو، اشاره‌گر ماوس را بر روی تعریف DbContext قرار دهیم، راهنمای زیر ظاهر می‌شود:

```
A DbContext instance represents a combination of the Unit Of Work and Repository patterns such that
it can be used to query from a database and group together changes that will then be written back to
the store as a unit.  DbContext is conceptually similar to ObjectContext.
```

در اینجا تیم EF صراحتا عنوان می‌کند که DbContext در EF Code first همان الگوی Unit Of Work را پیاده سازی کرده و در داخل کلاس مشتق شده از آن، DbSetها همان Repositories هستند (فقط نام‌ها تغییر کرده‌اند؛ اصول یکی است).
به عبارت دیگر با نام بردن صریح از این الگوها، مقصود زیر را دنبال می‌کنند:
لطفا بر روی این لایه Abstraction ایی که ما تهیه دیده‌ایم، یک لایه Abstraction دیگر را ایجاد نکنید!
«لایه Abstraction دیگر» یعنی پیاده سازی الگوهای Unit Of Work و Repository جدید، برفراز الگوهای Unit Of Work و Repository توکار موجود!
کار اضافه‌ای که در بسیاری از سایت‌ها مشاهده می‌شود و ... متاسفانه اکثر آن‌ها هم اشتباه هستند! در ذیل روش‌های تشخیص پیاده سازی‌های نادرست الگوی Repository را بر خواهیم شمرد:

1) قرار دادن متد Save تغییرات نهایی انجام شده، در داخل کلاس Repository

متد Save باید داخل کلاس Unit of work تعریف شود نه داخل کلاس Repository. دقیقا همان کاری که در EF Code first به درستی انجام شده. متد SaveChanges توسط DbContext ارائه می‌شود. علت هم این است که در زمان Save ممکن است با چندین Entity و چندین جدول مشغول به کار باشیم. حاصل یک تراکنش، باید نهایتا ذخیره شود نه اینکه هر کدام از این‌ها، تراکنش خاص خودشان را داشته باشند.

2) نداشتن درکی از الگوی Unit of work

به Unit of work به شکل یک تراکنش نگاه کنید. در داخل آن با انواع و اقسام موجودیت‌ها از کلاس‌ها و جداول مختلف کار شده و حاصل عملیات، به بانک اطلاعاتی اعمال می‌گردد. پیاده سازی‌های اشتباه الگوی Repository، تمام امکانات را در داخل همان کلاس Repository قرار می‌دهند؛ که اشتباه است. این نوع کلاس‌ها فقط برای کار با یک Entity بهینه شده‌اند؛ در حالیکه در دنیای واقعی، اطلاعات ممکن است از دو Entity مختلف دریافت و نتیجه محاسبات مفروضی به Entity سوم اعمال شود. تمام این عملیات یک تراکنش را تشکیل می‌دهد، نه اینکه هر کدام، تراکنش مجزای خود را داشته باشند.

3) وهله سازی از DbContext به صورت مستقیم داخل کلاس Repository

4) Dispose اشیاء DbContext داخل کلاس Repository

هر بار وهله سازی DbContext مساوی است با باز شدن یک اتصال به بانک اطلاعاتی و همچنین از آنجائیکه راهنمای ذکر شده فوق را در مورد DbContext مطالعه نکرده‌اند، زمانیکه در یک متد با سه وهله از سه Repository موجودیت‌های مختلف کار می‌کنید، سه تراکنش و سه اتصال مختلف به بانک اطلاعاتی گشوده شده است. این مورد ذاتا اشتباه است و سربار بالایی را نیز به همراه دارد.
ضمن اینکه بستن DbContext در یک Repository، امکان اعمال کوئری‌های بعدی LINQ را غیرممکن می‌کند. به ظاهر یک شیء IQueryable در اختیار داریم که می‌توان بر روی آن انواع و اقسام کوئری‌های LINQ را تعریف کرد اما ... در اینجا با LINQ to Objects که بر روی اطلاعات موجود در حافظه کار می‌کند سر و کار نداریم. اتصال به بانک اطلاعاتی با بستن DbContext قطع شده، بنابراین کوئری LINQ بعدی شما کار نخواهد کرد.
همچنین در EF نمی‌توان یک Entity را از یک Context به Context دیگری ارسال کرد. در پیاده سازی صحیح الگوی Repository (دقیقا همان چیزی که در EF Code first به صورت توکار وجود دارد)، Context باید بین Repositories که در اینجا فقط نامش

DbSet تعریف شده، به اشتراک گذاشته شود. علت هم این است که EF از Context برای ردیابی تغییرات انجام شده بر روی موجودیت‌ها استفاده می‌کند (همان سطح اول کش که در قسمت‌های قبل به آن اشاره شد). اگر به ازای هر Repository یکبار وهله سازی DbContext انجام شود، هر کدام کش جداگانه خاص خود را خواهند داشت.
5) عدم امکان استفاده از تنها یک DbConetext به ازای یک Http Request
هنگامیکه وهله سازی DbContext به داخل یک Repository منتقل می‌شود و الگوی واحد کار رعایت نمی‌گردد، امکان به اشتراک گذاری آن بین Repositoryهای تعریف شده وجود نخواهد داشت. این مساله در برنامه‌های وب سبب کاهش کارآیی می‌گردد (باز و بسته شدن بیش از حد اتصال به بانک اطلاعاتی در حالیکه می‌شد تمام این عملیات را با یک DbContext انجام داد).

نمونه‌ای از این پیاده سازی اشتباه را در اینجا می‌توانید پیدا کنید. متاسفانه شبیه به همین پیاده سازی، در پروژه MVC Scaffolding نیز بکارگرفته شده است.

چرا تعریف لایه دیگری بر روی لایه Abstraction موجود در EF Code first اشتباه است؟

یکی از دلایلی که حین تعریف الگوی Repository دوم بر روی لایه موجود عنوان می‌شود، این است:
« **به این ترتیب به سادگی می‌توان ORM مورد استفاده را تغییر داد** » چون پیاده سازی استفاده از ORM، در پشت این لایه مخفی شده و ما هر زمان که بخواهیم به ORM دیگری کوچ کنیم، فقط کافی است این لایه را تغییر دهیم و نه کل برنامه را.
ولی سؤال این است که هرچند این مساله از هزار فرسنگ بالاتر درست است، اما واقعا تابحال دیده‌اید که پروژه‌ای را با یک ORM شروع کنند و بعد سوئیچ کنند به ORM دیگری؟!
ضمنا برای اینکه واقعا لایه اضافی پیاده سازی شده انتقال پذیر باشد، شما باید کاملا دست و پای ORM موجود را بریده و توانایی‌های در دسترس آن را به سطح نازلی کاهش دهید تا پیاده سازی شما قابل انتقال باشد. برای مثال یک سری از قابلیت‌های پیشرفته و بسیار جالب در NH هست که در EF نیست و برعکس. آیا واقعا می‌توان به همین سادگی ORM مورد استفاده را تغییر داد؟ فقط در یک حالت این امر میسر است: از قابلیت‌های پیشرفته ابزار موجود استفاده نکنیم و از آن در سطحی بسیار ساده و ابتدایی کمک بگیریم تا از قابلیت‌های مشترک بین ORMهای موجود استفاده شود.
ضمن اینکه مباحث نگاشت کلاس‌ها به جداول را چکار خواهید کرد؟ EF راه و روش خاص خودش را دارد، NH چندین و چند روش خاص خودش را دارد! این‌ها به این سادگی قابل انتقال نیستند که شخصی عنوان کند: «هر زمان که علاقمند بودیم، ORM مورد استفاده را می‌شود عوض کرد!»

دلیل دومی که برای تهیه لایه اضافه‌تری بر روی DbContext عنوان می‌کنند این است:
« **با استفاده از الگوی Repository نوشتن آزمون‌های واحد ساده‌تر می‌شود** ». زمانیکه برنامه بر اساس Interfaceها کار می‌کند می‌توان آن‌ها را بجای اشاره به بانک اطلاعاتی، به نمونه‌ای موجود در حافظه، در زمان آزمون تغییر داد.
این مورد در حالت کلی درست است اما .... نه در مورد بانک‌های اطلاعاتی!
زمانیکه در یک آزمون واحد، پیاده سازی جدیدی از الگوی Interface مخزن ما تهیه می‌شود و اینبار بجای بانک اطلاعاتی با یک سری شیء قرارگرفته در حافظه سروکار داریم، آیا موارد زیر را هم می‌توان به سادگی آزمایش کرد؟
ارتباطات بین جداول‌را، cascade delete، فیلدهای identity، فیلدهای unique، کلیدهای ترکیبی، نوع‌های خاص تعریف شده در بانک اطلاعاتی و مسایلی از این دست.
پاسخ: خیر! تغییر انجام شده، سبب کار برنامه با اطلاعات موجود در حافظه خواهد شد، یعنی LINQ to Objects.
شما در حالت استفاده از LINQ to Objects آزادی عمل فوق العاده‌ای دارید. می‌توانید از انواع و اقسام متدها حین تهیه کوئری‌های LINQ استفاده کنید که هیچکدام معادلی در بانک اطلاعاتی نداشته و ... به ظاهر آزمون واحد شما پاس می‌شود؛ اما در عمل بر روی یک بانک اطلاعاتی واقعی کار نخواهد کرد.
البته شاید شخصی عنوان که بله می‌شود تمام این‌ها نیازمندی‌ها را در حالت کار با اشیاء درون حافظه هم پیاده سازی کرد ولی ... در نهایت پیاده سازی آن بسیار پیچیده و در حد پیاده سازی یک بانک اطلاعاتی واقعی خواهد شد که واقعا ضرورتی ندارد.

و پاسخ صحیح در اینجا و این مساله خاص این است:
لطفا در حین کار با بانک‌های اطلاعاتی مباحث mocking را فراموش کنید. بجای SQL Server، رشته اتصالی و تنظیمات برنامه را به SQL Server CE تغییر داده و آزمایشات خود را انجام دهید. پس از پایان کار هم بانک اطلاعاتی را delete کنید. به این نوع آزمون‌ها اصطلاحا integration tests گفته می‌شود. لازم است برنامه با یک بانک اطلاعاتی واقعی تست شود و نه یک سری شیء ساده

قرار گرفته در حافظه که هیچ قیدی همانند شرایط کار با یک بانک اطلاعاتی واقعی، بر روی آن‌ها اعمال نمی‌شود.
ضمنا باید درنظر داشت بانک‌های اطلاعاتی که تنها در حافظه کار کنند نیز وجود دارند. برای مثال SQLite حالت کار کردن صرفا در حافظه را پشتیبانی می‌کند. زمانیکه آزمون واحد شروع می‌شود، یک بانک اطلاعاتی واقعی را در حافظه تشکیل داده و پس از پایان کار هم ... اثری از این بانک اطلاعاتی باقی نخواهد ماند و برای این نوع کارها بسیار سریع است.

نتیجه گیری:

حین استفاده از EF code first، الگوی واحد کار، همان DbContext است و الگوی مخزن، همان DbSetها. ضرورتی به ایجاد یک لایه محافظ اضافی بر روی این‌ها وجود ندارد.
در اینجا بهتر است یک لایه اضافی را به نام مثلا Service ایجاد کرد و تمام اعمال کار با EF را به آن منتقل نمود. سپس در قسمت‌های مختلف برنامه می‌توان از متدهای این لایه استفاده کرد. به عبارتی در فایل‌های Code behind برنامه شما نباید کدهای EF مشاهده شوند. یا در کنترلرهای MVC نیز به همین ترتیب. این‌ها مصرف کننده نهایی لایه سرویس ایجاد شده خواهند بود.
همچنین بجای نوشتن آزمون‌های واحد، به Integration tests سوئیچ کنید تا بتوان برنامه را در شرایط کار با یک بانک اطلاعاتی واقعی تست کرد.

برای مطالعه بیشتر:

Abstracting your ORM is a futile exercise
Repository is the new Singleton
Night of the living Repositories
Architecting in the pit of doom: The evils of the repository abstraction layer
?Is Repository old skool
?EF Code First: Where’s My Repository
Generic Repository With EF 4.1 what is the point

نظرات خوانندگان

نویسنده: NTC
تاریخ: ۲۳:۱۹:۵۵ ۱۳۹۱/۰۲/۲۴

سلام و ممنون
میشود در مورد خط زیر بیشتر توضیح دهید؟
"در اینجا بهتر است یک لایه اضافی را به نام مثلا Service ایجاد کرد و تمام اعمال کار با EF را به آن منتقل نمود. سپس در قسمت‌های مختلف برنامه می‌توان از متدهای این لایه استفاده کرد. "
یعنی چی تمام اعمال کار با EF را به آن منتقل کنیم؟
چطور؟

نویسنده: ناشناس
تاریخ: ۲۳:۵۱:۵۶ ۱۳۹۱/۰۲/۲۴

قبلا که این روش رو توصیه می کردید http://www.dotnettips.info/2009/10/nhibernate_17.html "کلا استفاده‌ی از هر کدام از ORMs موجود بدون پیاده سازی الگوی Repository اشتباه است. به چند دلیل:" و همین دلایل بالا رو نوشته بودید؟

نویسنده: Javad Darvish Amiry
تاریخ: ۰۰:۰۰:۳۹ ۱۳۹۱/۰۲/۲۵

توی این سری پست ها، از این پست واقعا واقعا واقعا لذت بردم. دست گلتون درد نکنه. خدا خیرتون بده. زنده باشین.

نویسنده: Javad Darvish Amiry
تاریخ: ۰۰:۰۲:۲۶ ۱۳۹۱/۰۲/۲۵

جانا سخن از زبان ما می گویی...
توی این سری پست ها، از این پست واقعا واقعا واقعا لذت بردم. دست گلتون درد نکنه. خدا خیرتون بده. زنده باشین.

نویسنده: mohammad
تاریخ: ۰۰:۲۸:۰۳ ۱۳۹۱/۰۲/۲۵

سلام آقای نصیری. آیا این روش که در خود سایت asp.net انجام شده هم اشتباه هستش؟

http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

نویسنده: شاهین کیاست
تاریخ: ۰۱:۴۵:۴۵ ۱۳۹۱/۰۲/۲۵

سلام ،

ذکر کردید "بهتر است یک لایه ی اضافی به نام سرویس ایجاد شود و اعمال مربوط به EF را به آن منتقل نمود" یعنی به ازای هر موجودیت یک سرویس هم داشته باشیم ؟ این (http://pastebin.com/iUCn1303) نمونه کدی که اینجا قرار دادم صحیح است ؟ - Service همان Business logic ما خواهد بود ؟ یعنی علاوه بر اعمال CRUD ، بررسی منطق تجاری ، Validate کردن Entity و یا اعمال شرط های مختلف در Query ها در همین لایه ی سرویس انجام می شود ؟
-اگر بخواهیم اطلاعات User را با اطلاعات پروفایل در قالب یک لیست برای لایه ی پایین تر بفرستیم ، نوع خروجی چه باید باشد ؟ مثلا UserProfileDataTransferObject ؟

در سری هشتم Refactoring در مورد God Classes توضیح دادید. برای منطق های پیچیده روی یک Enity مثل User مثلا چندین Query با حالت های مختلف کلاس UserService به God Class تبدیل می شود. راه حل چیست ؟ اگر ممکن است پروژه ی کد باز مورد تایید خودتون برای مرور کد ها معرفی کنید.
خیلی ممنون

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۲۵ ۰۸:۴۳:۰۷

سلام. با توجه به مواردی که تا این شماره در مورد EF Code First فرمودید، بنده یک معماری مناسب برنامه را با استفاده از EF اینگونه درک کردم:

1- Domain Layer یا Model Layer: با استفاده از Code First فقط Entityها تعریف می شود (بدون Mapping و Data Annotation)

2- DAL: با استفاده از DbContext و Fluent API موارد مرتبط با Mapping و Data Annotation و ساخت DbSet ها تعریف می شود.

3- Facade DAL: با استفاده از WCF یا هر تکنیک سرویس گرایی دیگر یک لایه سطح بالا بر روی DAL کشیده می شود تا در BLL تنها از آن استفاده شود و عملا لایه های BLL و UI از EF هیچ اطلاعی نخواهند داشت.

4- BLL: کلاس های مربوط به برنامه، Rule ها و Infrastructureها
5- UI: شامل پروژه WPF، ASP.NET MVC و یا برنامه های موبایلی

آیا این معماری درک درستی از مطالب ارائه شده توسط شما است؟
و سوال دوم اینکه در مورد برنامه هایی با منطق MVC و یا MVVM لایه مدل تقریبا با این معماری چیزی نخواهد بود و تنها ویو مدل ها به چشم خواهند خورد. آیا این مورد هم درست است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۰۹:۲۴:۲۶

اون پیاده سازی هم اشتباه است! چون متد Save داخل خود Repository است.
ضمنا اون مطالب رو بر اساس اطلاعات آن روز نوشتم. الان هم پیاده سازی UOW و Repository کاملی رو از NH دارم ولی ... هیچ وقت NH رو در پروژه‌ای که از آن استفاده کردم، تغییر ندادم یا از آن repository برای Unit testing استفاده نکردم.
وابستگی به ORM در عمل زیاد خواهد بود. بنابراین تعویض آن غیرممکن می‌شود.
استفاده از بانک اطلاعاتی واقعی بهتر است و به همین جهت دو کاربردی که برای آن در اکثر سایت‌ها ذکر شده، در عمل استفاده نمی‌شود.
بنابراین استفاده از Repository صرفا پیچیدگی بی‌موردی را به سیستم تحمیل می‌کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۰۹:۳۳:۴۶

اولین StudentRepository را که نوشته، بله. در اینجا Unit of work را نقض کرده.
ولی درادامه ... خیر (زمانیکه UnitOfWork را ارائه داده). ولی کار اضافی انجام داده.
بینید در کلاس UnitOfWork کار وهله سازی Context انجام شده. بنابراین درست است. در همین کلاس هم Save قرار دارد بنابراین درست عمل شده و می‌شود با این سیستم یک تراکنش را انجام داد. ضمنا در این کلاس Uow کار معرفی Repository ها رو انجام داده.
ولی ... خود Context اصلی هم این موارد را دارد (خودش Uow است. کلاس مشتق شده از آن دارای DbSet است). شما به چه نتیجه‌ای از این Abstraction بی‌مورد خواهید رسید؟ احتمالا دو مورد عنوان شده در بالا ... که توضیح دادم در عمل هیچ وقت رخ نخواهد داد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۰۹:۴۴:۴۶

1- بله.
2- بله.
3- این هم خوبه ولی اگر بانک اطلاعاتی و برنامه وب شما مثلا در یک سرور قرار دارند ضرورتی به استفاده از WCF نیست و به کارآیی بیشتری حین استفاده مستقیم از بانک اطلاعاتی خواهید رسید. WCF برای معماری چند tier توصیه می‌شود (هر tier رو یک سرور در اینجا فرض کنید. یک سرور جدای وب، یک سرور جدای اس کیوال و الی آخر ....)
4- BLL همان لایه سرویسی‌است که عنوان کردم. جایی که از EF استفاده می‌شود.
5- بله.

این M در MVC مرتبط با ASP.NET MVC جای بحث زیاد دارد. بیشتر ViewModel است تا Model به معنای Domain Classes .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۰۹:۵۳:۱۳

- بله. چیزی شبیه به همین، البته وجود Interface هم در اینجا غیرضروری است. چون در مورد mocking صحبت کردم که در بانک‌های اطلاعاتی روش مناسبی نیست و بیشتر «توهم» صحیح کارکردن سیستم را به همراه خواهد داشت. توهم پاس شدن آزمون‌های واحدی که در عمل ممکن است تعداد زیادی از آن‌ها روی بانک اطلاعاتی واقعی اجرا نشوند. فرق است بین کار با اشیاء درون حافظه و بانک اطلاعاتی واقعی که بسیاری از قیود را اعمال می‌کند.
- بله. اسامی مهم نیست. یکی عنوان می‌کند BLL یکی Infrastructure یکی Service یکی...
- خروجی لایه سرویس فقط باید از نوع اشیاء معمولی، لیست یا IEnumerable باشد. اگر از IQueryable به عنوان خروجی متد استفاده کردید، به یک Abstraction دارای «نشتی» خواهید رسید. چون به سادگی خارج از منطقی که مدنظر بوده کاملا قابل تغییر است.
- God Class کلاسی است که اطلاعات زیادی را در اختیار سایرین قرار می‌دهد، نه کلاسی که جهت برآورده سازی منطق تجاری برنامه، از اطلاعات زیادی استفاده می‌کند. کلاسی که 1000 تا متد را در اختیار سایر کلاس‌ها قرار می‌دهد God class نام دارد. نه متدی که برای عملکرد صحیح خود نیاز به اطلاعات زیادی است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۰۹:۵۷:۲۶

یک مثال ساده: بجای اینکه در Code behind برنامه شروع کنید به وهله سازی از DbContext و بعد کوئری بنویسید و حاصل را مثلا در اختیار یک Grid قرار دهید، یک کلاسی ... جایی در یک پروژه Class library مجزا درست کنید که حاوی متد مشخصی است که فقط یک IList را برمی‌گرداند. این متد، محل کار با EF است نه Code behind یک فرم. در آنجا فقط باید از لیست نهایی تهیه شده استفاده شود.

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۰:۰۶:۵۷

1- پس به نظر شما نیازی به ایجاد یک لایه facade که بین DAL و BLL قرار گیرد و توابع EF و LINQ را برای استفاده در لایه بیزینس Wrap کند نیست و می توان از EF و LINQ مستقیما در BLL استفاده کرد و نهایتا به یک معماری چهار لایه رسید؟

2- سوال دیگر اینکه جدا سازی Domain ها از لایه DAL چه مزایایی دارد؟ آیا در مهاجرت به یک ORM دیگر مفید است یا ملاحضات دیگری در میان است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۰:۱۴:۴۳

- به همین دلیل در مورد عدم استفاده از Repository توضیح دادم. کار Repository همین warping عملکرد یک ORM است که نه

تنها ضرورتی ندارد بلکه در یک پروژه واقعی به شدت جلوی آزادی عمل شما را خواهد گرفت و همچنین پیاده سازی شما هم قابل انتقال نخواهد بود. استفاده از ORMها وابستگی‌های زیادی را به همراه دارند که شاید تنها قسمتی از آن‌ها را بتوانید مخفی کنید. همچنین برای اینکار باید از قابلیت‌های پیشرفته آن‌ها که ممکن است در سایر ORMs موجود نباشد، صرف نظر کرد. آزمون‌های واحد مرتبط با بانک‌های اطلاعاتی نیاز به بانک اطلاعاتی واقعی دارند تا بتواند قیود را اعمال کند. کار با اشیاء درون حافظه در اینجا اصلا توصیه نمی‌شود.
- بهتره. ضرورتی نداره. در حد یک مدیریت پروژه بهتر است که با یک نگاه بتوان تشخیص داد ... حداقل یک پوشه Models در برنامه هست. تا این حد کفایت می‌کند.

نویسنده: A. Karimi
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۰:۲۱:۵۴

یک دلیل مهم برای ایجاد یک Abstraction بر روی EF CodeFirst وجود دارد و آن هم استقلال از EF و IQueryable است. مثلاً ممکن است شما بخواهید بعداً به جای EF از NHibernate استفاده کنید. و یا ممکن است (حتماً) تکنولوژی جدیدتری بعد از EF وارد کار شود. از همه مهمتر ممکن است بخواهید از الگوی Adapter یا Decorator استفاده کنید. مثلاً یک Repository که اطلاعات یک جدول را به صورت Encrypt شده در بانک اطلاعات ذخیره می‌کند. و مثلاً به صفحه بندی سمت سرور هم نیاز دارید و حتی امکان استفاده از IQueryable هم وجود ندارند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۰:۳۲:۰۵

این همان مورد «به این ترتیب به سادگی می‌توان ORM مورد استفاده را تغییر داد» است که در بالا توضیح دادم. نمی‌تونید به این سادگی وابستگی‌های ذاتی به ORM مورد استفاده را حذف کنید. تمام کار با ORM به Update و Delete ختم نمی‌شود. برای نوشتن یک لایه قابل انتقال نیاز خواهید داشت دست و پای ORM مورد استفاده را قطع کنید که مثال زدید ... استقلال از IQueryable . و سؤال اینجا است که تمام لطف EF همین IQueryable است؛ چرا باید از این مستقل شد. من از کار کردن مستقیم با آن لذت می‌برم! به همین دلیل EF را انتخاب کرده‌ام. فقط لایه سرویس مورد استفاده نباید IQueryable را بازگرداند.

نویسنده: A. Karimi
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۱:۱۵:۴۸

ما با ایجاد لایه های متفاوت از Abstraction این قضیه را حل می‌کنیم. مثلاً در چهارچوبی که تهیه کرده‌ام، یک اینتفریس IRepository دارم که از 7 دولت آزاد است. و یک IQueryableRepository دارم که مخصوص ORMهای با پشتیبانی از IQueryable است. شاید شما بفرمایید وقتی از IQueryableRepository در برنامه استفاده کنی به IQueryable وابسته می‌شوی. بله، درست است، اما مثلا برای عملیات CRUD کلاسهایی برای WPF یا ASP.NET MVC در چهارچوب دارم که کاملاً از ORM مستقل هستند. به این ترتیب وقتی بخواهیم از یک ORM به دیگری Switch کنیم حداقل نیازی نیست کلاس‌های Base (همان کلاسهایی که در چهارچوب و طی زمان بیشتری تهیه شده) کوچکترین تغییری بکنند. و با این مکانیزم در صورتی که پروژه بلند مدت باشد باز هم می‌توان این استقلال را در لایه های[1] دیگر حفظ کرد به این ترتیب:

1. لایه Data Access دارای یک Infrastructure است که فقط اینترفیس‌های UnitOfWork و Repository در آن تعریف شده و هیچ خبری از IQueryable نیست مثلا برای دریافت داده صفحه بندی شده چیزی شبیه startRowIndex و maximumRows و حتی sortExpression پاس داده می‌شود.2. برای هر ORM یک پروژه جدا که به Data.Infrastructre رفرنس دارد ایجاد می‌شود. حالا در این پروژه UnitOfWork و Repository پیاده‌سازی می‌شود که از نظر داخلی به IQueryable وابسته است و مثلاً در پیاده‌سازی همان متد صفحه بندی شده به راحتی از LINQ استفاده می‌کند.3. هیچ کس حق ندارد در لایه های دیگر از IQueryable استفاه کند و باید ابتدا متد مورد نظرش را به Data.Infrastructure اضافه و بعد در لایه مرحله دو پیاده‌سازی کند.4. با استفاده از DI مسئله جایگذین کردن لایه مرحله 2 به راحتی اتفاق می‌افتد و هیچ لایه‌ای مستقیم به پیاده‌سازی سمت Data وابسته نیست بلکه به Data.Infrastructure وابسته است.

من قبول دارم که این کار زمان بیشتری را می‌گیرد اما شما هم خوب می‌دانید که اگر می‌خواهیم وقت صرف کنیم و یک چهارچوب ماندگار ایجاد کنیم باید طراحی مستقل از تکنولوژی داشته باشیم (برای مثال حتی WPF و MVC که عرض کردم هم در چهارچوب جز

Concrete class به حساب می‌آیند؛ هر چند از دید پروژه نهایی و مصرف کننده چهارچوب اینطور نیست).

مثال کاربردی این قضیه این است که در یک پروژه مجبور بودیم اطلاعات را رمزنگاری شده در DB ذخیره و بازیابی کنیم و البته فرم مورد نظر از هما CRUD های داخل چهارچوب بود و اگر این طراحی استفاه نمی‌شد مجبور بودیم برای آن یک فرم از یک ابتدا پیاده‌سازی کنیم و چون CRUD موجود در چهارچوب دارای امکانات قابل توجهی بود در زمان و هزینه تاثیر زیادی داشت.

[1] منظور از لایه، لایه‌های فیزیکی و سنتی نیست.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۱:۵۳:۱۱

- ORMهای با پشتیبانی از IQueryable ، پشتیبانی یکسانی از متدهای الحاقی تعریف شده ندارد. کد شما قابل انتقال نیست. برای مثال NH یک سری متد الحاقی خاص خودش را دارد. بنابراین اگر تصور می‌کنید که می‌توان این پیاده سازی را به ORM دیگری تغییر داد، در عمل ممکن نیست؛ مگر اینکه از توانایی‌های محدود و مشترکی استفاده کنید.
- یکی دیگر از اشتباهات طراحی الگوی Repository متدهای عمومی هستند که دارای خروجی IQueryable است. به این نوع طراحی، طراحی نشتی دار گفته می‌شود چون ابتدای کار مشخص است اما انتهای کار بازگذاشته شده است: (^)
- یک مثال دیگر: NH دارای متدی است به نام tofeautre که توانایی اجرای چندین و چند مثلا sum را بر روی بانک اطلاعاتی در یک رفت و برگشت دارد. لایه Repository شما نمی‌تواند این را مخفی کند و در صورت استفاده از آن قابل انتقال نخواهد بود. استفاده از آن هم ایرادی ندارد چون دلیل استفاده از یک ORM، استفاده از توانایی‌های پیشرفته آن‌ها است. از این نمونه مثال زیاد است.
حالا اینجا اگر شخصی از الگوی Repository استفاده کند، هم بی‌جهت خودش را محدود کرده و هم نهایتا به یک leaky abstraction رسیده.
- «چهارچوبی دارم که کاملاً از ORM مستقل هستند» این همان لایه سرویس است که از آن صحبت شد. نباید کدهای یک ORM (هر ORM ایی) داخل Code behind یا کنترلرهای MVC مشاهده شوند. این روش توصیه شده است.
- رمزنگاری را می‌شود با یک سری متدالحاقی در یک پروژه دیگری به نام Common پیاده سازی کرد. در لایه Service از آن استفاده کرد. بحث ما در اینجا در مورد Repository و عدم ضرورت استفاده از آن است. یا مباحث صفحه بندی هم به همین ترتیب. این‌ها یک سری متدالحاقی عمومی هستند؛ خارج از تعریف الگوی Repository قرار می‌گیرند.

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۲:۰۱:۳۴

و فکر می کنم یک دلیل مهم برای ایجاد انتزاعی (Abstraction) به نام Service پیش بینی همچین مهاجرتی هست. من Service رو لایه BLL نمیدونم (شخصا و طبق تحلیل و تجربه خودم؛ درست و غلطش رو نمیدونم) بلکه یه لایه واسط بین BLL و DAL میدونم. به این ترتیب همیشه یه DAL کاملا مستقل و منتزع از بقیه برنامه دارم. هر وقت خواستم از EF به NH برم -یا تجربه واقعی تر این که اگه خواستم به یه تکنولوژی جدیدتر کوچ کنم- کافیه لایه DAL جدیدم رو بنویسم و فقط نحوه دسترسی در سرویس تغییر میکنه. پس باز هم انتزاع مورد نیاز رو دارم.
اما در مورد UoW خوب باز هم طبق تجربه حتی با وجود DbContext (و Session در NH) فکر میکنم وجودش خیلی مفیده. مثلا تو برنامه های وب UoWم رو برپایه HttpModule قرار میدم و موقع Dispose شدن ماژول، تغییرات رو ذخیره میکنم.
در مورد M در MVC کاملا با آقای نصیری موافقم. این بحثو مدتی پیش با یه دوستی هم داشتیم که زیر بار نرفت. مثالی که اونجا زدم اینجا میزارم، فکر کنم مفیده.
فکر کنید شیئی به نام Member دارید با مثلا 20 پراپرتی. و این 20 خاصیت، در 5 ویوی مختلف نمایش داده میشن. یعنی یه ویو 6 تا، یه ویو 3 تا و همینطور تا آخر. خوب عاقلانه نیست که وقتی که به هر 20 خاصیت نیاز نداریم، همشو واکشی کنیم. پس دو تا راه داریم. یا باید اون 3 4 تا خاصیتی که نیاز داریم رو جداگانه (مثلا تو یه Tuple یا Anon یا حتی متغیرهای منحصر) واکشی کنیم و در اختیار ویو بذاریم؛ یا بیایم و یه ساختمون (class یا struct) تعریف کنیم مخصوص اون 3 4 تا پراپرتی. خوب من روش دوم رو ترجیح میدم که همون view-model های منو میسازه. یا نمایی رو در نظر بگیرید که لازمه از ترکیبی از دو یا چند مدل داده استفاده کنه. تو همون مثال Member، فرض کنید نمایی هست که نام کاربری و نام و نام خانوادگی رو از Member و تعداد ارسال ها رو از Comments و آخرین فعالیت رو از MemberActivitis و آدرس ایمیل عمومی رو از AuthTokens میخونه! چه میکنید؟

ضمنا به فلسفه وجودی مثلا AutoMapper هم فکر کنیم.

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۲:۰۸:۵۲

{ ORMهای با پشتیبانی از IQueryable ، پشتیبانی یکسانی از متدهای الحاقی تعریف شده ندارد.}
مثال واقعی و زنده ای که همین الان میشه زد تفاوت پیاده سازی L2E در NH و EF هست. عملا مهاجرت غیر ممکنه مگه از روشی که آقای نصیری گفتن. (دیدم که میگم ;))

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۲:۱۲:۲۱

همه این‌ها نیکو! ولی زمانیکه از یک ORM استفاده می‌کنید، DAL همین ORM است. تعویض آن هم به سادگی میسر نیست. من با شناختی که مثلا از NH دارم عرض می‌کنم که کسی که از NH استفاده می‌کنه نمی‌تونه توانایی‌های توکار آن‌را با هیچ ORM دیگری عوض کنه. سطح دوم کش، غیرفعال کردن سطح اول کش برای مباحث گزارشگیری. متدهای ویژه‌ای که در QueryOver آن پیش‌بینی شده. استفاده از HQL آن برای اعمالی که نه با LINQ میسر است و نه با QueryOver.
خلاصه از سطحی خیلی بالا اینطور به نظر میرسه که می‌شود یک لایه انتزاعی بر روی ORM درست کرد ولی در عمل اینطور نیست. این لایه قابل انتقال نیست مثلا به EF یا نمونه‌های مشابه.

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۲:۲۳:۵۶

خوب منم همینو عرض میکنم. میگم انتقال ممکن نیست مگر به روش شما (محدود کردن ORM و صرفنظر از کلی از قابلیت هاشون). اگه سرویس رو بین DAL و بقیه اجزا برنامه داشته باشیم چی؟ من از EF استفاده میکنم. سرویس من نه انتیتی ها بلکه viewmodel های مشخصی که هر بخش برنامه نیاز داره میگیره یا بر میگردونه. مثلا متدی که تو همون بخش MVC مثالشو گفتم در نظر بگیرید. اسمشو میذاریم GetMemberInfo. سرویس موظفه viewmodel مرتبط رو پر کنه و برگردونه. امروز از EF استفاده میکنه؛ پس عاقلانه ترین راه استفاده از پروژکشن هست. فردا روزی به هر دلیلی مجبورم به NH کوچ کنم. خوب باقی اجزا برنامه از این تغییر بی خبر میمونن. حالا NH هم پروژکشن داره هم tofeautre که بعنوان یه نمونه، فکر میکنم استفاده از امکان tofeautre اینجا بهتره. خوب کی از این تغییر باخبره؟ فقط سرویس. سرویسه که داره پیاده سازی های مختلفی میشه. بقیه اجزا کاملا از این تغییر مصون میمونن.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۲:۳۰:۰۹

بله. این روش طراحی خیلی خوبه؛ برنامه استفاده کننده از نحوه پیاده سازی GetMemberInfo بی‌خبر است. حالا می‌خواهد NH باشد یا EF یا هر مورد دیگری.
ضمن اینکه این لایه میانی رو که عنوان کردید هم در برنامه‌های وب می‌تونه استفاده شود و هم ویندوزی.

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۲:۳۴:۱۴

اگه باز بخوام توضیح بدم زیاده گویی میشه. ولی راه حل خودم اینه:
برای هر بخش منطقی برنامه که نیاز به داده پایگاه داره یه سرویس در نظر میگیرم. یه interface برای اون سرویس میگیرم. مثلا IMemberService. حالا لایه دسترسی به داده تو یه پروژه جدا قرار میگیره. اگه قراره از EF استفاده کنم، EfMemberService رو میسازم. به همه قابلیت های EF هم دسترسی دارم. متود GetMemberInfo هم به هر روشی که خودش میدونه موظفه اطلاعات مورد نیاز رو واکشی و برگردونه. چون روشش در اختیار خودشه پس میتونه از همه قابلیت های EF استفاده کنه. حالا مثلا اگه بیام و StructureMap استفاده کنم (که میکنم) میتونم فایل رجیستری برای IMemberService رو هم تو همون پروژه بذارم. با استارت برنامه، DependencyResolver میفهمه هر جا به IMemberService نیاز داشت، باید از EfMemberService کنه.
فردا یه تکنولوژی جدیدتر میاد و EF رو به نابودی (یا حداقل حاشیه) میره. مثلا اسم اون ORM رو میذاریم NH. یه پروژه جدید تعریف میکنم برای NH. سرویس ها رو توش پیاده سازی میکنم. مثلا حالا NhMemberService دارم. فایل رجیستری تو همون پروژه قرار

داره. DLL نهایی رو با DLL قبلی عوض میکنم و برنامه رو دوباره استارت میکنم. حالا هر جا به IMemberService نیاز داشتم، NhMemberService استفاده میشه.
یعنی دقیقا همونکاری که شما فرمودید. EfMemberService و NhMemberService کاملا مستقل هستن و هر کدوم میتونن از تمام قابلیت های ORM مورد استفاده شون استفاده کنن.
کل منظورم همین بود.

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۳:۰۰:۳۵

{هم در برنامه‌های وب می‌تونه استفاده شود و هم ویندوزی}
خوب این هم باز یه نکته است. اون UoW که گفتم بالا، دقیقا مربوط میشه به این بخش. مثال:
تو کامنت قبلی از استقلال لایه سرویس عرض کردم. حالا اضافه میکنم، سرویس من داره از EF استفاده میکنه. اما مستقیما نمیاد DbContext رو صدا بزنه. بلکه اون رو از یه UoW میگیره. خاصیتش اینه که UoW من از طریق یه HttpModule وهله سازی و نابود میشه. تو نابود شدنش dirty بودنش رو بررسی و در صورت نیاز commit میشه.
حالا چطور برم رو ویندوز؟ HttpContext اگه null باشه پس رو وب نیستم! راهکارم شبیه سازی HttpContext مثلا با یه کلاس به اسم HatContext هست. حالا میتونم لایه سرویس رو هم در وب و هم در ویندوز استفاده کنم.
UoWم هم سه مرحله توسعه داره. بالاترین سطح، فقط IsDirty رو در اختیار میذاره. سطح دوم فقط Commit و Rollback و سطح سوم TContext. سطح دوم و سوم فقط در اختیار سرویس قرار داره (internal). سطح اول در اختیار کل برنامه است (public).
پس IsDirty، مثلا تو WPF خیلی میتونه مفید واقع بشه. حالا Forward کردن انواع (مثلا با StructureMap) کمکم میکنه که دسترسی به هر کدوم از interface های سه گانه بالا، فقط یه نمونه رو برگردونه.

{این یه مثال انتزاعی نیست؛ دقیقا یه پروژه ی واقعی بود که درگیرش بودم.}

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۳:۰۸:۰۵

بله. این مورد به الگوی «Session Per Request» معروف است که کار آن به اشتراک گذاری یک DbContext در طول مدت عمر یک Http Request است. در این مورد یک مطلب خواهم نوشت.

نویسنده: A. Karimi
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۳:۲۲:۲۰

- همانطور که عرض کردم ما به هیچ عنوان وابستگی Contract ی به IQueryable نداریم. دقت بفرمایید که برای چیزی مثل NH ما اصلاً از IQueryable استفاده نخواهیم کرد. صحبت بنده Abstract تر بود. "لایه Data Access دارای یک Infrastructure است که فقط اینترفیس‌های UnitOfWork و Repository در آن تعریف شده و هیچ خبری از IQueryable نیست".

- طراحی Repository اصلی خروجی و ورودی IQueryable ندارد با این صحبت کاملاً موافق هستم و غیر از این هم نیست.

- همانطور که عرض کردم این موضوع کاملاً قابل اجراست. شما به ازاء هر ORM یک DataAccess خواهید داشت و امکانات قوی هر ORM را در داخل DataAccess خودش به راحتی استفاده می‌کنید به لایه بندی و ارجاعاتی که عرض کردم دقت بفرمایید. مصرف کننده Data به جای استفاده از DataAccess های Concrete از سطوح بالاتری استفاده می‌کنند و DI این موضوع را حل کرده است. شما می‌توانید از هر امکانی که دوست دارید و مخصوص ORM خودتان است در آن استفاده کنید.

- لایه سرویس با لایه Data متفاوت است. با چیزی که شما فرمودید اگر ORM تغییر کند باید کدهای لایه سرویس را دستی تغییر دهید و با Breaking Change روبرو می‌شویم. اما همانطور که قبلاً هم گفتم، با وجود این Abstraction که گفتم، فقط با یک Config ساده در DI این موضوع به طور کامل حل می‌شود. کلاً فلسفه وجودی Repository همین ورود و خروج داده (ذخیره و بازیابی) و Abstract بودن آنهاست. کدهای ORM نه تنها نباید در Code Behinde ها دیده شود بلکه نباید در Service یا Business سیستم هم باشند. به همان دلیلی که عرض کردم. و البته مثال رمزنگاری مثال خوبی است (Service چرا باید درگیر رمزنگاری شود؟ این

وظیفه لایه ذخیره و بازیابی است که همان Repository خواهد بود).

- Repository http://martinfowler.com/eaaCatalog/repository.html همانطور که از نامش پیداست یک مخزن است که ما نمی‌دانید اطلاعات را در کجا نگهداری خواهد کرد (حافظه، بانک و ...) حال در یک پروژه این محل نگهداری باید امن باشد همین! به جای حافظه و بانک و ... تصور کنید یک جای امن را. آیا برای این نمی‌توان (نباید) یک Repository ساخت؟ مورد بعدی اینکه رمزنگاری به لایه دیگری منتقل شده و Repository که گفتم از همان استفاده می‌کند. نکته بعدی اینکه با چیزی که شما فرمودید، باید مراقبت از اینکه همه چیز رمزنگاری شود را به برنامه نویس محول کرد و حتماً حواسش باشد که فراموش نکند قبل از ذخیره یا بازیابی یک Entity متدهای لازم برای رمزنگاری را فراخوانی کند. در صورتی که لایه سرویس باید متمرکز بر روی Business کار باشد نه فراخوانی متدهای رمزنگاری برای ذخیره سازی و ...

نویسنده: A. Karimi
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۳:۳۰:۳۵

"لایه Data Access دارای یک Infrastructure است که فقط اینترفیس‌های UnitOfWork و Repository در آن تعریف شده و هیچ خبری از IQueryable نیست"

صحبت بنده این بوده؛ ما به این فکر می‌کنیم که از IQueryable هم Abstract باشیم. چرا فکر می‌کنید قرار است ORM ها پشتیبانی یکسانی از متدهای الحاقی داشته باشند؟ مشخص است که یکسان نیست! در صورتی که بخواهیم از NH استفاده کنیم انتخاب اول QueryOver است نه LINQ to NH. غیر از مورد نشتی، یکی دیگر از دلایل اینکه گفته می‌شود از IQueryable در Contract ها استفاده نشود همین عدم سازگاری و پشتیبانی است.

به طوری کلی، به عبارت ساده لایه DataAccess ما پیمانه‌ای تر است (دارای Modularity بالاتری است) و بحث اصلا بر سر IQueryable نیست شما باید بتوانید یک Repository بسازید که حتی به عنوان محل ذخیره سازی از یک فایل Text استفاده کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۳:۳۳:۲۵

چندتا بحث هست. مدیریت پروژه. استفاده از Repository.
اینکه پوشه درست کنید، class library درست کنید. اسم‌های متفاوت استفاده کنید. همه این‌ها خوب است.
باز هم در طراحی خودتون اومدید ORM رو مخفی کردید. کل بحث جاری این است که اینکار اتلاف وقت است. «فقط با یک Config ساده در DI این موضوع به طور کامل حل می‌شود» : .... نمی‌شود. خیر! قصد ندارم مواردی رو که عنوان کردم تکرار کنم. مدتی با یک ORM با قابلیت‌های بالا کار کنید، این مطلب رو دیگر عنوان نخواهید کرد.
به نظر در مورد اسامی کمی تداخل اینجا هست. مفهومی رو که از Repository دنبال می‌کنید، همان چیزی است که در لایه سرویس من به آن اشاره کردم.
اگر علاقمند بودید، به پیاده سازی generic repository که لینک دادم مراجعه کنید.
واژه‌های مورد استفاده رو اگر یکسان کنیم شاید خیلی از سؤ تفاهم‌ها برطرف شود.

نویسنده: A. Karimi
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۴:۰۴:۰۰

موافقم، بحث مدیریت پروژه و دیگر مسائل خیلی مطرح و تاثیر گذارند.

در خصوص Config ساده در DI و حلی شدن موضوع، با نظر شما موافق نیستم و این کار انجام شدنیست (بحث مخفی کردن یا پیمانه‌ای کردن). این بحث‌ها بسیار جالب و جذاب است و ای کاش در محیط مناسب تر و راحتتری به بحث می‌پرداختیم.

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۴:۰۹:۰۶

{شما باید بتوانید یک Repository بسازید که حتی به عنوان محل ذخیره سازی از یک فایل Text استفاده کند} کاملا موافقم. و این همون چیزیه که ازش بعنوان لایه سرویس یاد کردم. توضیحات کامل رو تو کامنت های پایین تر دادم.
فکر کنم اختلاف رو نامگذاری ها و برداشت متفاوتی که از تعاریف داریم، باشه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۴:۲۵:۳۵

من فکر می‌کنم طرحی که از Repository در ذهن شما است، تعریف یک اینترفیس که دارای متدهای مثلا Add و Get و امثال آن است. سپس پیاده سازی کامل آن با EF. چون مبتنی بر Interface است می‌شود یک پیاده سازی مبتنی با NH را هم برای آن تدارک دید. بعد این‌ها رو میشه با DI مدیریت کرد. بله. این شدنی است. فقط واژه‌های استفاده شده در اینجا بین من و شما یکی نیست. من Repository رو به عنوان یک لایه سبک محصور کننده خود EF مد نظر دارم. یعنی پیاده سازی که در هزاران سایت اینترنتی داره تبلیغ میشه و به نظر من لزومی ندارد. انتقال پذیر نیست و لذت استفاده از یک ORM واقعی رو از بین می‌بره.
در کل من از واژه سرویس استفاده کردم شما از واژه مخزن. ولی به نظر برداشت هر دو یکی است.

نویسنده: مجید شمخانی
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۷:۵۳:۰۸

به نظر من در این بحث «تعویض ORM» اصل YAGNI را نباید فراموش کرد، ولی با این حال آیا شما استفاده از الگوی Repository و UOW را برای NH پیشنهاد می‌کنید یا خیر؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۱۹:۳۰:۱۱

شما فقط به uow برای پیاده سازی session per request نیاز خواهید داشت (به اشتراک گذاری فقط یک سشن در طول عمر یک http request در لایه‌های مختلف برنامه).
الگوی Repository خصوصا در مورد NH قابل انتقال نیست. چون تا حد بسیار زیادی به جزئیات LINQ، QueryOver و حتی HQL وابسته می‌شود که در سایر ORMهای دیگر معادل ندارد. به همین جهت تحمیل این لایه به سیستم غیرضروری است.
یعنی در ابتدا مقالات را که مطالعه کنید، همه عنوان می‌کنند که با استفاده از Repository به سیستمی خواهید رسید که می‌توانید ORM آن‌را به سادگی تعویض کنید. اما در مورد NH ابدا اینطور نیست. هنوز حتی خیلی از حالات mapping رو که این سیستم پشتیبانی می‌کنه در سایر ORMها نمی‌تونید پیدا کنید. بنابراین این Repository قابل تعویض نیست. بهتره بگم این ORM اصلا قابل تعویض نیست؛ چون اصطلاحا feature rich است. مگر اینکه از توانایی‌های پیشرفته آن استفاده نشود که آن وقت ضرورت استفاده از آن‌را زیر سؤال می‌برد.

نویسنده: مجید شمخانی
تاریخ: ۱۳۹۱/۰۲/۲۵ ۲۲:۲۶:۴۰

اگر امکان دارد نمونه‌ای از این نوع پیاده سازی را برای NH معرفی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۵ ۲۲:۳۸:۳۹

این مورد برای مثال: (^)

نویسنده: Mohsen Esmailpour
تاریخ: ۱۳۹۱/۰۲/۲۶ ۰۱:۵۳:۰۶

از وقتی ک شروع به یاد گرفتن ASP.NET MVC کردم از کتاب Pro ASP.NET MVC 3 Framework گرفت تا آموزشهای خود سایت asp.net و در بسیاری از وبلاگها همه استفاد از repository رو به عنوان best pratice توصیه کردن. استفاده از Interface + EF + DI و mock در unit test به نظر خوب میاد. حالا این سئوال برام پیش اومده آیا از اشکال از تیم EF هست که در پیاده سازی DbContext از الگوی Unit of Work استفاده کرده و یک لایه Abstraction روی اون کشیده و یا ایراد از بی اطلاعی کسانی هست

الگوی Repository و EF Code First رو با هم به کار میبرن. این موضوع برای من مثل این میمونه ک یک اینترفیس رو با پیادسازی اجزاش به آدم بدن و بعد خودت دوباره بیای اجزاش رو پیاده سازی کنی و خبر نداشته که قبلا پیاده سازی شدن. من که گیج شدم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۶ ۰۸:۲۹:۰۶

- استفاده از uow صحیح است. نیاز است یک uow بین لایه‌های مختلف به اشتراک گذاشته شود.
- استفاده از Repository که به صورت یک لایه سبک روی EF عمل کند اتلاف وقت است. به دلایلی که عرض کردم.
- استفاده از لایه سرویس توصیه شده است.

این موارد رو به صورت یک مثال عملی در قسمت بعدی توضیح خواهم داد.

نویسنده: بازرگان
تاریخ: ۱۳۹۱/۱۱/۲۰ ۲۰:۳۳

با سلام؛

خواهشمند یه پروژه نمونه برای asp.net mvc4 که در آن از 5 entity framework code first و unit of work استفاده شده و بحث فوق در آن رعایت شده باشد معرفی نمایید.

آیا پروژه efmvc آنچه شما در اینجا گفته اید رعایت کرده و اگر نه موارد را بیان نمایید.

با سپاس فراوان

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۲۱ ۱۹:۵۲

پروژه قسمت 12 رو که من دیدم با ef5 و mvc4 سازگار است و اصولش فرقی نکرده.

نویسنده: ناظم
تاریخ: ۱۳۹۲/۱۰/۱۷ ۲۲:۲۵

با سلام
جناب نصیری من پس از خواندن مطالب واقعا مفید شما و چند تا مطلب دیگه تو سایتهای CodeProject , Stackoverflow ,Asp.NET , ... تقریبا گیج شدم بنابر این چند تا سوال دارم
1- وقتی از یک orm مثلا EF استفاده میکنم داشتن یک class library به نام DAL و انتقال edmx یا کلاس‌های code first به اون اشکالی که نداره؟
2- لایه سرویس همان BLL هست؟ میتوان اونجا مستقیم به DbContext و توابع اون مثل Delete ، add و غیره دسترسی داشت؟
3- یه جا مثل اینکه فرموده بودید UoW خوبه استفاده کنیم و مخزن نه درسته ؟ اگه آره چرا و چطور ؟
4- من یه Classlibrary تشکیا میدم به اسم Entities و POCOهای EF رو منتقل میکنم اونجا در آینده مشکل ساز که نیست؟
5- جای صحیح استفاده از الگوهای مخزن و UoW کجاست؟
6- میشه یه مثال که همه اینهارو که فرمودید رو رعایت کرده و مورد تایید شماست رو معرفی بفرمایید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۷ ۲۲:۳۴

لطفا قسمت بعدی یعنی 12 را به همراه تمام پرسش و پاسخ‌های آن، مطالعه کنید. پیاده سازی UOW، مثال لایه بندی، تزریق وابستگی‌ها، بحث و غیره، تمامی در آن موجود است. در پرسش و پاسخ‌های آن، دوره‌های پیشنیاز مرتبط و پروژه‌های تکمیلی مرتبط هم معرفی شده‌اند.

عنوان: **Repository ها روی UnitOfWork ایده خوبی نیستند**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۳/۰۲/۰۸ ۱۹:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Entity framework, ORM

در دنیای دات نت گرایشی برای تجزیه (abstract) کردن EF پشت الگوی Repository وجود دارد. این تمایل اساسا بد است و در ادامه سعی می‌کنم چرای آن را توضیح دهم.

پایه و اساس

عموما این باور وجود دارد که با استفاده از الگوی Repository می‌توانید (در مجموع) دسترسی به داده‌ها را از لایه دامنه (Domain) تفکیک کنید و "داده‌ها را بصورت سازگار و استوار عرضه کنید".

اگر به هر کدام از پیاده سازی‌های الگوی Repository در کنار UnitOfWork (EF) دقت کنید خواهید دید که تفکیک (decoupling) قابل ملاحظه ای وجود ندارد.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public class StudentRepository : IStudentRepository, IDisposable
    {
        private SchoolContext context;

        public StudentRepository(SchoolContext context)
        {
            this.context = context;
        }

        public IEnumerable<Student> GetStudents()
        {
            return context.Students.ToList();
        }

        public Student GetStudentByID(int id)
        {
            return context.Students.Find(id);
        }

        //<snip>
        public void Save()
        {
            context.SaveChanges();
        }
    }
}
```

این کلاس بدون SchoolContext نمی‌تواند وجود داشته باشد، پس دقیقا چه چیزی را در اینجا decouple کردیم؟ **هیچ چیز را!**

در این قطعه کد - از MSDN - چیزی که داریم یک پیاده سازی مجدد از LINQ است که مشکل کلاسیک Repository APIهای بی انتها را بدست می‌دهد. منظور از Repository APIهای بی انتها، متدهای جالبی مانند GetStudentById, GetStudentByBirthday, GetStudentByOrderNumber و غیره است.

اما این مشکل اساسی نیست. مشکل اصلی روتین Save() است. این متد یک دانش آموز (Student) را ذخیره می‌کند .. اینطور بنظر می‌رسد. دیگر چه چیزی را ذخیره می‌کند؟ آیا می‌توانید حدس بزنید؟ من که نمی‌توانم .. بیشتر در ادامه.

UnitOfWork تراکنشی است یک UnitOfWork همانطور که از نامش بر می‌آید برای **انجام کاری** وجود دارد. این کار می‌تواند به

سادگی واکشی اطلاعات و نمایش آنها، و یا به پیچیدگی پردازش یک سفارش جدید باشد. هنگامی که شما از EntityFramework استفاده می‌کنید و یک DbContext را وهله سازی می‌کنید، در واقع یک UnitOfWork می‌سازید.

در EF می‌توانید با فراخوانی ()SubmitChanges تمام تغییرات را فلاش کرده و بازنشانی کنید (flush and reset). این کار بیت‌های مقایسه change tracker را تغییر می‌دهد. افزودن رکوردهای جدید، بروز رسانی و حذف آنها. هر چیزی که تعیین کرده باشید. و تمام این دستورات در یک تراکنش یا Transaction انجام می‌شوند.

یک Repository مطلقا یک UnitOfWork نیست

هر متد در یک Repository قرار است فرمانی اتمی (Atomic) باشد - چه واکشی اطلاعات و چه ذخیره آنها. مثلا می‌توانید یک Repository داشته باشید با نام SalesRepository که اطلاعات کاتالوگ شما را واکشی می‌کند، و یا یک سفارش جدید را ثبت می‌کند. منظور از فرمان‌های اتمیک این است، که هر متد تنها یک دستور را باید اجرا کند. تراکنشی وجود ندارد و امکاناتی مانند ردیابی تغییرات و غیره هم جایی ندارند.

یکی دیگر از مشکلات استفاده از Repositoryها این است که بزودی و به آسانی از کنترل خارج می‌شوند و نیاز به ارجاع دیگر مخازن پیدا می‌کنند. به دلیل اینکه مثلا نمی‌دانستید که SalesRepository نیاز به ارجاع ReportRepository داشته است (یا چیزی مانند این).

این مشکل به سرعت مشکل ساز می‌شود، و نیز به همین دلیل است که به UnitOfWork تمایل پیدا می‌کنیم.

بدترین کاری که می‌توانید انجام دهید: Repository<T> این الگو دیوانه وار است. این کار عملا انتزاعی از یک انتزاع دیگر است (abstraction of an abstraction). به قطعه کد زیر دقت کنید، که به دلیلی نامشخص بسیار هم محبوب است.

```
public class CustomerRepository : Repository < Customer > {
  public CustomerRepository(DbContext context){
    //a property on the base class
    this.DB = context;
  }

  //base class has Add/Save/Remove/Get/Fetch
}
```

در نگاه اول شاید بگویید مشکل این کلاس چیست؟ همه چیز را کپسوله می‌کند و کلاس پایه Repository هم به کانتکست دسترسی دارد. پس مشکل کجاست؟

مشکلات عدیده اند .. بگذارید نگاهی بیاندازیم.

آیا می‌دانید این DbContext از کجا آمده است؟

خیر، نمی‌دانید. این آبجکت به کلاس تزریق (Inject) می‌شود، و نمی‌دانید که چه متدی آن را باز کرده و به چه دلیلی. ایده اصلی پشت الگوی Repository استفاده مجدد از کد است. بدین منظور که مثلا برای عملیات CRUD از کلاسی پایه استفاده کنید تا برای هر موجودیت و فرمی نیاز به کدنویسی مجدد نباشد. برگ برنده این الگو نیز دقیقا همین است. مثلا اگر بخواهید از کدی در چند فرم مختلف استفاده کنید از این الگو استفاده میشد.

الگوی UnitOfWork همه چیز در نامش مشخص است. اگر قرار باشد آنرا بدین شکل تزریق کنید، نمی‌توانید بدانید که از کجا آمده است.

شناسه مشتری جدید را نیاز داشتم

کد بالا در CustomerRepository را در نظر بگیرید - که یک مشتری جدید را به دیتابیس اضافه می‌کند. اما CustomerID جدید چه می‌شود؟ مثلا به این شناسه نیاز دارید تا یک log بسازید. چه می‌کنید؟ گزینه‌های شما اینها هستند:

متد ()SubmitChanges را صدا بزنید تا تغییرات ثبت شوند و بتوانید به CustomerID جدید دسترسی پیدا کنید
CustomerRepository خود را باز کنید و متد پایه Add را بازنویسی (override) کنید. بدین منظور که پیش از بازگشت دادن، متد ()SubmitChanges را فراخوانی کند. این راه حلی است که MSDN به آن تشویق می‌کند، و بمبی ساعتی است که در انتظار انفجار است
تصمیم بگیرید که تمام متدهای Add/Remove/Save در مخازن شما باید ()SubmitChanges را فراخوانی کنند

مشکل را می‌بینید؟ مشکل در خود پیاده سازی است. در نظر بگیرید که چرا New Customer ID را نیاز دارید؟ احتمالا برای استفاده از آن در ثبت یک سفارش جدید، و یا ثبت یک ActivityLog.

اگر بخواهیم از StudentRepository بالا برای ایجاد دانش آموزان جدید پس از خرید آنها از فروشگاه کتاب مان استفاده کنیم چه؟ اگر DbContext خود را به مخزن تزریق کنید و دانش آموز جدید را ذخیره کنید .. اوه .. تمام تراکنش شما فلاش شده و از بین رفته!

حالا گزینه‌های شما اینها هستند: 1) از StudentRepository استفاده نکنید (از OrderRepository یا چیز دیگری استفاده کنید). و یا 2) فراخوانی ()SubmitChanges را حذف کنید و به باگ‌های متعددی اجازه ورود به کد تان را بدهید.

اگر تصمیم بگیرید که از StudentRepository استفاده نکنید، حالا کدهای تکراری (duplicate) خواهید داشت.

شاید بگویید که برای دستیابی به شناسه رکورد جدید نیازی به ()SubmitChanges نیست، چرا که خود EF این عملیات را در قالب یک تراکنش انجام می‌دهد!

دقیقا درست است، و نکته من نیز همین است. در ادامه به این قسمت باز خواهیم گشت.

متدهای Repositories قرار است اتمیک باشند

به هر حال تئوری اش که چنین است. چیزی که در Repositoryها داریم حتی اصلا Repository هم نیست. بلکه یک abstraction برای عملیات CRUD است که هیچ کاری مربوط به منطق تجاری اپلیکیشن را هم انجام نمی‌دهد. مخازن قرار است روی دستورات مشخصی تمرکز کنند (مثلا ثبت یک رکورد یا واکشی لیستی از اطلاعات)، اما این مثال‌ها چنین نیستند.

همانطور که گفته شده استفاده از چنین رویکردهایی به سرعت مشکل ساز می‌شوند و با رشد اپلیکیشن شما نیز مشکلات عدیده ای برایتان بوجود می‌آرند.

خوب، راه حل چیست؟

برای جلوگیری از این abstractionهای غیر منطقی دو راه وجود دارد. اولین راه استفاده از Command/Query Separation است که ممکن است در ابتدا کمی عجیب و بنظر برسند اما لازم نیست کاملا CQRS را دنبال کنید. تنها از سادگی انجام کاری که مورد نیاز است لذت ببرید، و نه بیشتر.

آبجکت‌های Command/Query

Jimmy Bogard مطلب خوبی در اینباره نوشته است و با تغییراتی جزئی برای بکارگیری Properties کدی مانند لیست زیر خواهیم داشت. مثلا برای مطالعه بیشتر درباره آبجکت‌های Command/Query به این لینک سری بزنید.

```
public class TransactOrderCommand {
  public Customer NewCustomer {get;set;}
  public Customer ExistingCustomer {get;set;}
  public List<Product> Cart {get;set;}
  //all the parameters we need, as properties...
  //...

  //our UnitOfWork
  StoreContext _context;
  public TransactOrderCommand(StoreContext context){
    //allow it to be injected - though that's only for testing
```

```
    _context = context;
  }

  public Order Execute(){
    //allow for mocking and passing in... otherwise new it up
    _context = _context ?? new StoreContext();

    //add products to a new order, assign the customer, etc
    //then...
    _context.SubmitChanges();

    return newOrder;
  }
}
```

همین کار را با یک آبجکت Query نیز می‌توانید انجام دهید. می‌توانید پست Jimmy را بیشتر مطالعه کنید، اما ایده اصلی این است که آبجکت‌های Query و Command برای دلیل مشخصی وجود دارند. می‌توانید آبجکت‌ها را در صورت نیاز تغییر دهید و یا mock کنید.

DataContext خود را در آغوش بگیرید ایده ای که در ادامه خواهید دید را شخصا بسیار می‌پسندم (که توسط Ayende معرفی شد). چیزهایی که به آنها نیاز دارید را در قالب یک فیلتر wrap کنید و یا از یک کلاس کنترلر پایه استفاده کنید (با این فرض که از اپلیکیشن‌های وب استفاده می‌کنید).

```
using System;
using System.Web.Mvc;

namespace Web.Controllers
{
  public class DataController : Controller
  {
    protected StoreContext _context;

    protected override void OnActionExecuting(ActionExecutingContext filterContext)
    {
      //make sure your DB context is globally accessible
      MyApp.StoreDB = new StoreDB();
    }

    protected override void OnActionExecuted(ActionExecutedContext filterContext)
    {
      MyApp.StoreDB.SubmitChanges();
    }
  }
}
```

این کار به شما اجازه می‌دهد که از DataContext خود در خلال یک درخواست واحد (request) استفاده کنید. تنها کاری که باید بکنید این است که از این کلاس پایه ارث بری کنید. این بدین معنا است که هر درخواست به اپلیکیشن شما یک UnitOfWork خواهد بود. که بسیار هم منطقی و قابل قبول است. در برخی موارد هم شاید این فرض درست یا کارآمد نباشد، که در این هنگام می‌توانید از آبجکت‌های Command/Query استفاده کنید.

ایده‌های بعدی: چه چیزی بدست آوردیم؟ چیزهای متعددی بدست آوردیم.

تراکنش‌های روشن و صریح : دقیقا می‌دانیم که DbContext ما از کجا آمده و در هر مرحله روی چه UnitOfWork ای کار می‌کنیم. این امر هم الان، و هم در آینده بسیار مفید خواهد بود

انتزاع کمتر == شفافیت بیشتر : ما Repositoryها را از دست دادیم، که دلیلی برای وجود داشتن نداشتند. به جز اینکه یک abstraction از abstraction دیگر باشند. رویکرد آبجکت‌های Command/Query تمیزتر است و دلیل وجود هرکدام و مسئولیت آنها نیز روشن‌تر است

شانس کمتر برای باگ ها : رویکردهای مبتنی بر Repository باعث می‌شوند که با تراکنش‌های ناموفق یا پاره ای (partially-executed) مواجه شویم که نهایتا به یکپارچگی و صحت داده‌ها صدمه می‌زند. لازم به ذکر نیست که خطایابی و رفع چنین مشکلاتی شدیدا زمان بر و دردسر ساز است

برای مطالعه بیشتر

ایجاد Repositories بر روی UnitOfWork
به الگوی Repository در لایه DAL خود نه بگویید!
پیاده سازی generic repository یک ضد الگو است
نگاهی به generic repositories
بدون معکوس سازی وابستگی‌ها، طراحی چند لایه شما ایراد دارد

نظرات خوانندگان

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۳/۰۲/۰۸ ۱۹:۵۴

سلام آرمین جان ممنون از مطلبت
به نظرم جای یک بحثی خالی اونم تست پذیری کد.

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۲/۰۸ ۲۱:۱۳

ممنون.
با بیشتر مطالب شما موافقم ولی Repositoryها نیز دلیلی برای وجود دارند.
»الگوی Repository بسیار پروژه را تست پذیر می‌کند. به راحتی با استفاده از کتابخانه‌های Mock می‌توان بخش دسترسی به داده را تست کرد.
»اگر منظور شما از StoreContext ، کلاسی است که مستقیم از DBContext ارث برده است، در نتیجه امکان استفاده از دستوراتی نظیر Entry of T و Set of T یا مواردی مربوط به Change Tracking نیز به صورت مستقیم حتی در الگوی CQRS نیز وجود دارد.
چگونه می‌توانید دستوراتی این چنینی را Mock کنید؟ استفاده از کتابخانه‌های Mock نظیر Moq برای تست دستوراتی نظیر Entry Of T و SetCurrentValues و GetCurrentValues کمکی به شما نمی‌کند. (برای DbSet کتابخانه ای نظیر FakeDbSet وجود دارد ولی برای سایر دستورات خیر...)
»اگر از روش توصیه شده در این جا استفاده کنید باز برای Mock آبجکت IUnitOfWork به مشکل بر خواهید خورد. در این حالت برای تست لایه‌های دسترسی بهتر است از کتابخانه هایی نظیر Effort استفاده نمایید.

» در بخش **شناسه مشتری جدید را نیاز داشتم** یک راه حل را فراموش کردید و آن استفاده از GUID برای تعریف Id هر entity است در نتیجه دیگر نیازی به واکشی مجدد رکورد نخواهید داشت.

»بهتر است متد Save را نیز در Repository قرار ندهید. متد Save باید توسط UnitOfWork به اشتراک گذاشته شده فراخوانی شود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۰۸ ۲۱:۴۰

- How EF6 Enables Mocking DbSets more easily
- Testing with a mocking framework - EF6 onwards

+ شخصا اعتقادی به Unit tests درون حافظه‌ای، در مورد لایه دسترسی به داده‌ها ندارم . به قسمت « Limitations of EF in-memory test doubles » مراجعه کنید؛ توضیحات خوبی را ارائه داده‌است.
تست درون حافظه‌ی LINQ to Objects با تست واقعی LINQ to Entities که روی یک بانک اطلاعاتی واقعی اجرا می‌شود، الزاما نتایج یکسانی نخواهد داشت (به دلیل انواع قیود بانک اطلاعاتی، پشتیبانی از SQL خاص تولید شده تا بارگذاری اشیاء مرتبط و غیره) و نتایج مثبت آن به معنای درست کار کردن برنامه در دنیای واقعی نخواهد بود. در اینجا Integration tests بهتر جواب می‌دهند و نه Unit tests.

نویسنده: جلال
تاریخ: ۱۳۹۳/۰۲/۰۸ ۲۱:۵۷

خدا از دهنت بشنوفه. مدت هاست منم به همین نتیجه رسیدم تازه وقتی فهمیدم بدون اون بازم میشه قابلیت تست پذیری رو داشت. کافیه یه واسط از خود DbContext برنامه بسازی .
ولی الگوی Repository توی استفاده از کلاس‌های پایه ADO.NET مثل DbCommand و DbConnection کارایی خوبی داره.

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۲/۰۸ ۲۲:۲

ممنونم جناب نصیری. دلیل اشاره من به عدم تست پذیری قابل قبول در حالت استفاده مستقیم از Context به خاطر وجود دستوراتی نظیر Entry of T یا موارد مربوط به ChangeTracking است که با تست درون حافظه ای نتیجه مطلوب حاصل نمی‌شود، در نتیجه بهتر است از Effort برای تست لایه دسترسی استفاده شود که عملیات را در قالب یک دیتابیس SqlCE تست می‌کند و نسخه Effort.Ef6 آن نیز از Entity Framework 6 به خوبی پشتیبانی می‌کند.

نویسنده: Ara
تاریخ: ۱۳۹۳/۰۲/۰۸ ۲۳:۱۳

با توجه به متن قضاوتتون عجولانه است !

تو پروژه‌های Huge که توصیه خود مایکروسافت استفاده از Domain Driven و CQRS می‌باشد ، Repository یکی از اصول Domain Driven و Enterprise Application Pattern می‌باشد !

نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۳/۰۲/۰۸ ۲۳:۵۹

با تشکر از همگی دوستان

شخصا نظرم به نظر جناب نصیری نزدیک‌تر است. جناب پاکدل هم به نکات خوبی اشاره فرمودند. اما صرفا توصیه‌های مایکروسافت و دیگران دال بر درستی یا کارآمدی یک رویکرد نمی‌تواند باشد. مطلب پست شده مبتنی بر چندین پست از توسعه دهندگان مطرح دنیای دات نت ترجمه و تالیف شده. مسلما هیچ راه حل نهایی (silver-bullet) ای وجود ندارد و توسعه ساختار پروژه بر اساس نیازها و تعاریف اپلیکیشن‌ها به پیش می‌رود. اما در کل می‌توان اینگونه نتیجه گیری کرد که استفاده از الگوی Repository در کنار فریم ورک‌های ORM مانند EF که مبتنی بر UnitOfWork کار می‌کنند ایده خوبی نیست. برای مطالعات بیشتر به چند لینک نمونه زیر مراجعه شود.

^ ، ^ ، ^ ، ^ ، ^ ، ^

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۲/۰۹ ۱:۲

در پروژه http://nopcommerce.codeplex.com استفاده از Repository جهت اجبار به رویکرد Command/Query بوده است.(البته اینطور برداشت میشود) جهت مطالعه
http://nopcommerce.codeplex.com/SourceControl/latest#src/Libraries/Nop.Data/EfRepository.cs
و همینطور جهت تست پذیری پروژه، راه حل‌های اشاره شده را پیاده سازی کرده.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۲/۰۹ ۱:۱۱

آقای پاکدل لطفا راجع به این جمله بیشتر توضیح بدید:

»بهتر است متد Save را نیز در Repository قرار ندهید. متد Save باید توسط UnitOfWork به اشتراک گذاشته شده فراخوانی شود. در پروژه http://nopcommerce.codeplex.com/SourceControl/latest اینگونه عمل شده
http://nopcommerce.codeplex.com/SourceControl/latest#src/Libraries/Nop.Data/EfRepository.cs
در نهایت این لایه سرویس است که باید اطمینان از انجام عملیات درخواستی و یا عدم انجام آنرا بدهد و برای عملیات‌های

پیچیده‌تر نیز بایستی سیاست خودرا بسط دهد.منظور انجام عملیات Save و ادامه عملیات میباشد.
لایه UI وظیفه فراهم آوری اطلاعات را دارد و مابقی مسائل در لایه سرویس پوشش داده میشوند.
الزام این کار هم به وظیفه این لایه برمیگردد که یا این کار را میتوانم انجام دهم و یا خیر.
پیاده سازی ارائه شده نقضی بر جمله‌ی نقل قول شده میباشد؟

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۲/۰۹ ۱۰:۱

به طور کلی هدف اصلی از الگوی واحد کار یا UnitOfWork به اشتراگ گذاشتن یک Context بین همه نمونه‌های ساخته شده از Repository یا سرویس‌های برنامه است. فرض کنید در یک کنترلر شما از دو یا سه نمونه از سرویس‌ها یا Repositoryها وهله سازی کرده اید. اگر قرار باشد برای اعمال تغییرات، مجبور به فراخوانی متد Save هر Repository باشیم چرا اصلا الگوی واحد کار را به کار بردیم؟ فراخوانی SaveChanges الگوی واحد کار معادل است با فراخوانی متدهای Save تمام Repositoryهای وهله سازی شده در طی یک درخواست.

نویسنده: آرایه
تاریخ: ۱۳۹۳/۰۲/۰۹ ۱۰:۲۰

دلایل منطقی هستند و کد ارائه شده در مثال‌ها واقعاً مشکل دارد. بعضی آثار را شاید بتوان کاهش داد. مثلاً برای رفع Repository APIهای بی‌انتها شاید استفاده از متدی که IQueryable برگرداند و بعد ادامه دادن کوئری در خروجی آن متد کمک کند.
یک پروژه برای پیاده سازی Generic از Repository و Unit Of Work اینجا هست که مشکلات کمتری دارد.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۲/۰۹ ۱۰:۲۸

صد در صد درست.ولی فکر میکنم این مسئله باید در لایه سرویس حل بشه. در یک Application انتظار چندین و چند عملیات در طی یک Request میره. برای نمونه میگم:
- در یک کنترلر قراره یک مشتری تعریف بشه.از طرفی هم لاگ گیری‌های عمومی سیستم نیز باید انجام باشه که اصولا در بعضی از عملیات‌ها مستقل از همدیگه باید باشند.پس باید چند بار SaveChanges فراخوانی بشه.
عملیات لاگ گیری سیستم حتما باید انجام بشه ولی عملیات تعریف یک مشتری میتونه دارای خطایی باشه.(استقلال بعضی از عملیات‌های سیستم در UOW)
-عملیات هایی که در طی یک Action در کنترلر انجام میشه:بایستی تمام اینها به لایه‌ی سرویس منتقل بشه و اونجا در طی یک SaveChange عملیات مورد نظر نتیجه بده.(رویکرد Command/Query)
- الگوی واحد کار هدف‌های بیشتری داره.
-مسئولیت هر متد در لایه سرویس مشخصه و نتیجه بازگشتی از لایه سرویس عملا بایستی دلالت بر نتیجه‌ی عملیات رو داشته باشه.نه اینکه در یک متد در لایه سرویس عملیات درج رو انجام بده و بعد در UI عملیات خطا بده.
-جدا سازی منطق لایه‌ها در این کار مشخص نیست.(تا حدی)
*مدیریت پیچیده وظایف در لایه سرویس به درستی انجام بشه SaveChangesها با کمترین سربار و بهترین کارایی انجام میشه.
البته فکر میکنم در پروژه اشاره شده نیز به همین مسئله دلالت داره.
http://nopcommerce.codeplex.com/SourceControl/latest#src/Libraries/Nop.Data/EfRepository.cs
و اینکه در بعضی از مسائل نیز باید تغییراتی صورت بگیره .مانند عملیات‌های گروهی.
و در نهایت صحبت آقای ضیا دلالت بر تفکرات متفاوت درستره.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۲/۰۹ ۱۱:۳

This is a leaky abstraction

نویسنده: Ara
تاریخ: ۱۳۹۳/۰۲/۱۳ ۰:۵۳

تو مبحث DDD دلیل اصلی که Repository وارد داستان شده Persistence Ignorance می‌باشد ،
همونطور که میدونید ، این قضیه می‌گه که شما تو Domain نبایید بگید EF این طوری Select می‌زنه Nhibernate یک نوع دیگر ، NoSql یک نحو دیگر (NoSql ها هم بخاطر اینکه میتونند براحتی یک Aggregate رو ذخیره کنند میتونند ابزار خوبی برای DDD باشند!)
چون Domain نباید به تکنولوژی وابسته باشد ! نباید رفرنسی به دیتا اکسس یا EF و یا ... داشته باشد فقط یک سری Interface تعریف می‌کند ، که یکی که بعدا به نام لایه دیتا اکسس می‌باشد باید این اینترفیس رو Implement کند !
در مورد CQRS هم چون معمولا Application Layer بر روی Rest هاست می‌شوند پس هر Request فقط شامل یک Command می‌باشد که Unit Of work رو هم فقط روی همان Command ایجاد می‌کنند
جالبه براتون بگم که در Domain Driven Design اصل بر این هست که شما در هر ترانزاکشن فقط یک Aggregate رو باید ذخیره کنید و تغییر در Aggregate های دیگه بوسیله Event Source ها Publish می‌شه

و از توصیه‌های اولیه DDD اینه که برای پروژه‌های Complex و Huge استفاده بشه ،پس قطعا برای یک پروژه که از این متد استفاده نمی‌شه و یا در ابعاد کوچکتر می‌باشد کاملا حرف شما درست باشد و از پیچیده شدن برنامه جلوگیری می‌کند

نویسنده: Ara
تاریخ: ۱۳۹۳/۰۲/۱۳ ۱:۳

پیاده سازی خوبیه
البته زمانی که از DDD استفاده می‌شه استفاده از IQueryable در IRepository به عنوان نشت اطلاعات خوانده می‌شود و تاکیدا نباید استفاده شود !

این Repository های Generic میتوانند داخل یک کلاس که IRepository را Implement کرده استفاده شوند و یا به عنوان کلاس Base ان باشند

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۲/۱۳ ۱:۱۸

این generic repository الان از امکانات async در EF 6 داره استفاده می‌کنه. برای مثال NH چنین توانمندی async ایی رو در حال حاضر نداره. آیا در این حالت Persistence Ignorance تامین شده؟ یعنی راحت میشه زیر ساخت این مخزن رو عوض کرد و سوئیچ کرد به یک ORM دیگه؟ و اگر نخواهیم از async استفاده کنیم، خوب یک ORM داریم که توانمندی‌های جدیدش رو باید ازش صرفنظر کرد. خروجی IQueryable آن که جای خودش. ORMهای مختلف متدهای الحاقی خاص خودشون رو دارند و پیاده سازی یکسانی از LINQ رو ندارند. یعنی اگر با EF کار کردید و متد Include آن توسط این generic repository بخاطر خروجی IQueryable در دسترس بود، معادلی در سایر ORMها نداره (متدهای الحاقی اون‌ها فرق می‌کنه). یا مثلا NH سطح دوم کش رو با متد الحاقی Cacheable پیاده سازی کرده. فرض کنید این رو در generic repository قرار دادیم (یک روکش روی این متد تا به ظاهر مستقیما در دسترس نباشه). خوب، الان فلان ORM دیگه که متد Cacheable رو نداره چکار باید باهاش کرد؟ این برنامه و سیستم به این سادگی‌ها قابل تبدیل به یک ORM دیگه نیست. رسیدن به Persistence Ignorance در دنیای واقعی کار ساده‌ای نیست مگر اینکه از توانمندی‌های خوب ORM انتخاب شده صرفنظر کنیم و به قولی دست و پاشو ببریم تا قد بقیه بشه.
گذشته از این‌ها بحث مدل سازی هم هست. نگاشت‌های کلاس‌ها و خواص اون‌ها به جداول بانک اطلاعاتی در ORMهای مختلف 100 درصد با هم متفاوت هست. حداقل EF و NH روش‌های خاص خودشون رو دارند که انطباقی با هم ندارن. یعنی این Persistence Ignorance محدود نیست به روکش کشیدن روی insert/update/delete. اینجا صحبت از یک سیستم هست که اجزای هماهنگ زیادی داره که باید درنظر گرفته بشه؛ از نگاشت‌ها تا اعتبارسنجی‌های خاص تا قابلیت‌های ویژه و صددرصد اختصاصی. به این می‌گن تا خرخره فرو رفتن!

نویسنده: Ara
تاریخ: ۱۳۹۳/۰۲/۱۳ ۷:۴۱

در مورد async راست می‌گید ! باید ببینم راهی داره یا نه ، در ضمن در لایه دیتا اکسس هر جور که می‌خواهید می‌تونید include و غیره بزنید مشکلی وجود نداره ، چون رو اینتفریس از شما می‌خواهند که یک entity چه چیزهایی همراهش باشه یا نباشه
خوب اگه به cqrs نگاه کنید در سمت Command شما قسمت اصلی و insert , Update , delete و Get رو دارید و برخی مواقع getall که خیلی کمه ولی سمت query کاملا دستتون بازه هر جور که کار کنید کلا پشت query سرویس هر جور که راحتی با هر چی که راحتی کار کن !

نویسنده: Ara
تاریخ: ۱۳۹۳/۰۲/۱۳ ۱۷:۱۷

مثل اینکه async کردن متدهای Repository زیاد پیچیده نمی‌باشد !

پس می‌شه query های NH رو هم Async کرد ، پس روی IRepository می‌تونیم هم متد Async هم متد Sync رو با هم داشته باشیم

نویسنده: علیرضا
تاریخ: ۱۳۹۳/۰۲/۲۶ ۱۲:۵۷

1- من دقیقا متوجه نشدم منظور شما از decoupling اول مقاله چیه؟ منظورتون تفکیک Domain از DAL هست؟ اگر اینطوره چه ربطی به UoW و انواع پیاده سازی اون داره؟
اگر منظورشما انفکاک بین EFContext و Repository هست، توجه شما رو به این نکته جلب میکنم که StudentRepository که در اول مقاله آورده شده در حقیقت یک پیاده سازی برپایه EF هست به عبارتی EFStudentRepository اسم مناسب‌تری میتونه باشه. بنابراین تزریق Context با هیچ اصلی مغایر نیست. چرا که این Repository یک پیاده سازی خاص از IStudentRepository است.

2- وجود متد Save در Repository؟ نه تنها قابل قبول نیست که اصلا اگر قرار باشه هر Repository مستقلا Save رو صدا بزنه که مفهوم Transaction از بین میره یا حداقل سخت میشه بهش رسید.

3- با شما موافقم که Generic Repository ایده خوبی نیست. البته فقط تا اینجا موافقم که این الگو برای Expose کردن Interface یک Repository مناسب نیست. چه بسا Repository هایی که فقط SELECT میکنند. ولی اگر پیاده سازی خاصی از یک Repository مد نظر دارید (مثلا پیاده سازی برپایه EF یا NHibernate) اونوقت دقیقا چیزی که به کمک شما میاد همین Generic Repository برای جلوگیری از کدهای تکراریه.

4- اصولا Repository برای اینکه منطق برنامه (یا به قول شما منطق تجاری) رو پیاده سازی کنه نیست. در حقیقت لایه ای که استفاده کننده مستقیم از Repository است میداند که چه موقع به چه Repository فرمانهای CRUD بده تا منطق برنامه پیاده سازی بشه.

5- در واقع استفاده از امکانات هر ORM تا حد بینهایتی امکان پذیره به شرطی که ORM و توانمندیهاشو در همون لایه DAL محصور کنید مثلا IQueriable و Cachable و گرنه Leaky Abstraction به طور خزنده و ساکتی کل برنامه رو مثل سرطان در خودش میکشه.

نهایتا اینکه نمیشه یک پیاده سازی مشکل دار از مفهوم Repository + UoW رو بدون درنظر گرفتن مفاهیم مهمی مثل Service Layer و Domain Model نقد کرد و بعدا نتیجه گرفت که این الگوها صحیح نیستند. ضمن اینکه این موضوع بسته به تجربه و نظر هر برنامه نویس و معماری میتونه پیاده سازی خاص خودشو داشته باشه که من شخصا هنوز موارد جالب و جدیدی که یک برنامه نویس باهوش برداشت کرده رو میبینم و نتیجه میگیرم که مفهوم Repository + UoW در بین ماها هنوز به یک تعریف جهانشمول نرسیده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۶ ۱۳:۱۸

- مباحث الگوی مخزن، در حالت کلی درست هستند؛ یک بحث انتزاعی، بدون درنظر گرفتن فناوری پیاده سازی کننده‌ی آن.
- در مورد EF به خصوص (در این مطلب)، DbSet و DbContext آن پیاده سازی کننده‌ی الگوهای Repository و Uow هستند (و منکر *آن نیستند*). به همین جهت عنوان می‌کنند که روی Repository آن، دوباره یک Repository درست نکنید. در بحث هم اشاره به «یک abstraction از abstraction دیگر» همین مطلب است.

```
public class MyContext : DbContext
class System.Data.Entity.DbContext
A DbContext instance represents a combination of the Unit Of Work and Repository patterns
```

تصویری است از قرار دادن کرسر ماوس بر روی DbContext در VS.NET که به *صراحت* در آن از پیاده سازی الگوی مخزن یاد شده

اینترفیس IDbSet معروف در EF دقیقا یک abstraction است و بیانگر ساختار الگوی مخزن. کاملا هم قابلیت mocking دارد؛ از نگارش 6 به بعد EF البته (^ و ^ و ^).

- راه حل‌های ارائه شده به دلیل اینکه Uow را تزریق نمی‌کنند مشکل دارند. اساسا هرگونه لایه بندی بدون تزریق وابستگی‌ها مشکل دارد؛ نمی‌شود یک وهله از یک شیء را بین چندین کلاس درگیر به اشتراک گذاشت (مباحث مدیریت طول عمر در IoC Containerها). مثلا در راه حل آخر ارائه شده فقط آغاز و پایان اجرای یک متد از یک کنترلر مشخص تحت نظر هستند. واقعیت این است که تا اجرای یک اکشن متد به پایان برسد، در طول یک درخواست، پردازش referrer رسیده هم در کلاسی دیگر به موازت آن باید انجام شود (در یک HTTP Module مجزا) و امثال آن. در این حالت چون یک وهله از Uow به اشتراک گذاشته نشده، مدام باید وهله سازی شود؛ بجای اینکه از آن تا پایان درخواست، استفاده‌ی مجدد شود. برای حل آن، در متن ذکر شده مطمئن شوید که «globally accessible» است. این مورد و راه حل‌های استاتیک (مانند نحوه‌ی فراخوانی MyApp آن) و singleton در برنامه‌های وب تا حد ممکن باید پرهیز شود. چون به معنای به اشتراک گذاری آن در بین تمام کاربران سایت. این مورد تخریب اطلاعات را به همراه خواهد داشت. چون DbContext جاری در حال استفاده توسط کاربر الف است و در همان زمان کاربر ب هم چون دسترسی عمومی به آن تعریف شده، مشغول به استفاده از آن خواهد شد. در این بین عملا تراکنش تعریف شده بی‌معنا است چون اطلاعات آن خارج از حدود متدهای مدنظر توسط سایر کاربران تغییر کرده‌اند.
همچنین به دلیل عدم تزریق وابستگی‌ها، پیاده سازی‌های آن تعویض پذیر نیستند و قابلیت آزمایش واحد پایینی خواهند داشت. برای مثال در بحث mocking که مطرح شد، می‌توانید بگویید بجای این متد خاص از کلاس اصلی، نمونه‌ی آزمایشی من را استفاده کن.

عنوان: EF Code First #12
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۶ ۱۰:۵۸:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

پیاده سازی الگوی Context Per Request در برنامه‌های مبتنی بر EF Code first

در طراحی برنامه‌های چند لایه مبتنی بر EF مرسوم نیست که در هر کلاس و متدی که قرار است از امکانات آن استفاده کند، یکبار DbContext و کلاس مشتق شده از آن وهله سازی شوند؛ به این ترتیب امکان انجام امور مختلف در طی یک تراکنش از بین می‌رود. برای حل این مشکل الگویی مطرح شده است به نام Session/Context Per Request و یا به اشتراک گذاری یک Unit of work در لایه‌های مختلف برنامه در طی یک درخواست، که در ادامه یک پیاده سازی آن‌را با هم مرور خواهیم کرد.
البته این سشن با سشن ASP.NET یکی نیست. در NHibernate معادل DbContextایی که در اینجا ملاحظه می‌کنید، Session نام دارد.

اهمیت بکارگیری الگوی Unit of work و به اشتراک گذاری آن در طی یک درخواست

در الگوی واحد کار یا همان DbContext در اینجا، تمام درخواست‌های رسیده به آن، در صف قرار گرفته و تمام آن‌ها در پایان کار، به بانک اطلاعاتی اعمال می‌شوند. برای مثال زمانیکه شیءایی را به یک وهله از DbContext اضافه/حذف می‌کنیم، یا در ادامه مقدار خاصیتی را تغییر می‌دهیم، هیچکدام از این تغییرات تا زمانیکه متد SaveChanges فراخوانی نشود، به بانک اطلاعاتی اعمال نخواهند شد. این مساله مزایای زیر را به همراه خواهد داشت:

الف) کارآیی بهتر
در اینجا از یک کانکشن باز شده، حداکثر استفاده صورت می‌گیرد. چندین و چند عملیات در طی یک batch به بانک اطلاعاتی اعمال می‌گردند؛ بجای اینکه برای اعمال هرکدام، یکبار اتصال جداگانه‌ای به بانک اطلاعاتی باز شود.

ب) بررسی مسایل همزمانی
استفاده از یک الگوی واحد کار، امکان بررسی خودکار تمام تغییرات انجام شده بر روی یک موجودیت را در متدها و لایه‌های مختلف میسر کرده و به این ترتیب مسایل مرتبط با ConcurrencyMode عنوان شده در قسمت‌های قبل به نحو بهتری قابل مدیریت خواهند بود.

ج) استفاده صحیح از تراکنش‌ها
الگوی واحد کار به صورت خودکار از تراکنش‌ها استفاده می‌کند. اگر در حین فراخوانی متد SaveChanges مشکلی رخ دهد، کل عملیات Rollback خواهد شد و تغییری در بانک اطلاعاتی رخ نخواهد داد. بنابراین استفاده از یک تراکنش در حین چند عملیات ناشی از لایه‌های مختلف برنامه، منطقی‌تر است تا اینکه هر کدام، در تراکنشی جدا مشغول به کار باشند.

کلاس‌های مدل مثال جاری

در مثالی که در این قسمت بررسی خواهیم کرد، از کلاس‌های مدل گروه محصولات کمک گرفته شده است:

```
using System.Collections.Generic;

namespace EF_Sample07.DomainClasses
{
    public class Category
    {
        public int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual string Title { get; set; }
```

```
        public virtual ICollection<Product> Products { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;


namespace EF_Sample07.DomainClasses
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }


        [ForeignKey("CategoryId")]
        public virtual Category Category { get; set; }
        public int CategoryId { get; set; }
    }
}
```

در کلاس Product، یک خاصیت اضافی به نام CategoryId اضافه شده است که توسط ویژگی ForeignKey، به عنوان کلید خارجی جدول معرفی خواهد شد. از این خاصیت در برنامه‌های ASP.NET برای مقدار دهی یک کلید خارجی توسط یک DropDownList پر شده با لیست گروه‌ها، استفاده خواهیم کرد.

پیاده سازی الگوی واحد کار

همانطور که در قسمت قبل نیز ذکر شد، DbContext در EF Code first بر اساس الگوی واحد کار تهیه شده است، اما برای به اشتراک گذاشتن آن بین لایه‌های مختلف برنامه نیاز است یک لایه انتزاعی را برای آن تهیه کنیم، تا بتوان آن‌را به صورت خودکار توسط کتابخانه‌های Dependency Injection یا به اختصار DI در زمان نیاز به استفاده از آن، به کلاس‌های استفاده کننده تزریق کنیم. کتابخانه‌ی DI ایی که در این قسمت مورد استفاده قرار می‌گیرد، کتابخانه معروف StructureMap است. برای دریافت آن می‌توانید از Nuget استفاده کنید؛ یا از صفحه اصلی آن در Github : (^).
اینترفیس پایه الگوی واحد کار ما به شرح زیر است:

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


namespace EF_Sample07.DataLayer.Context
{
    public interface IUnitOfWork
    {
        IDbSet<TEntity> Set<TEntity>() where TEntity : class;
        int SaveChanges();
    }
}
```

برای استفاده اولیه آن، تنها تغییری که در برنامه حاصل می‌شود به نحو زیر است:

```
using System.Data.Entity;
using EF_Sample07.DomainClasses;


namespace EF_Sample07.DataLayer.Context
{
    public class Sample07Context : DbContext, IUnitOfWork
```

```
    {
        public DbSet<Category> Categories { set; get; }
        public DbSet<Product> Products { set; get; }


        #region IUnitOfWork Members
        public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
        {
            return base.Set<TEntity>();
        }
        #endregion
    }
}
```

توضیحات:

با کلاس Context در قسمت‌های قبل آشنا شده‌ایم. در اینجا به معرفی کلاس‌هایی خواهیم پرداخت که در معرض دید EF Code first قرار خواهند گرفت.

DbSetها هم معرف الگوی Repository هستند. کلاس Sample07Context، معرفی الگوی واحد کار یا Unit of work برنامه است.

برای اینکه بتوانیم تعاریف کلاس‌های سرویس برنامه را مستقل از تعریف کلاس Sample07Context کنیم، یک اینترفیس جدید را به نام IUnitOfWork به برنامه اضافه کرده‌ایم.

در اینجا کلاس Sample07Context پیاده سازی کننده اینترفیس IUnitOfWork خواهد بود (اولین تغییر).

دومین تغییر هم استفاده از متد base.Set می‌باشد. به این ترتیب به سادگی می‌توان به DbSetهای مختلف در حین کار با IUnitOfWork دسترسی پیدا کرد. به عبارتی ضرورتی ندارد به ازای تک تک DbSetها یکبار خاصیت جدیدی را به اینترفیس IUnitOfWork اضافه کرد. به کمک استفاده از امکانات Generics مهیا، اینبار

```
uow.Set<Product>
```

معادل همان db.Products سابق است؛ در حالتیکه از Sample07Context به صورت مستقیم استفاده شود.

همچنین نیازی به پیاده سازی متد SaveChanges نیست؛ زیرا پیاده سازی آن در کلاس DbContext قرار دارد.

استفاده از الگوی واحد کار در کلاس‌های لایه سرویس برنامه

```
using EF_Sample07.DomainClasses;
using System.Collections.Generic;


namespace EF_Sample07.ServiceLayer
{
    public interface ICategoryService
    {
        void AddNewCategory(Category category);
        IList<Category> GetAllCategories();
    }
}
```

```
using EF_Sample07.DomainClasses;
using System.Collections.Generic;


namespace EF_Sample07.ServiceLayer
{
    public interface IProductService
    {
        void AddNewProduct(Product product);
        IList<Product> GetAllProducts();
    }
}
```

لایه سرویس برنامه را با دو اینترفیس جدید شروع می‌کنیم. هدف از این اینترفیس‌ها، ارائه پیاده سازی‌های متفاوت، به ازای

ORMهای مختلف است. برای مثال در کلاس‌های زیر که نام آن‌ها با Ef شروع شده است، پیاده سازی خاص Ef Code first را تدارک خواهیم دید. این پیاده سازی، قابل انتقال به سایر ORMها نیست چون نه پیاده سازی یکسانی را از مباحث LINQ ارائه می‌دهند و نه متدهای الحاقی همانندی را به همراه دارند و نه اینکه مباحث نگاشت کلاس‌های آن‌ها به جداول مختلف یکی است:

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.DomainClasses;


namespace EF_Sample07.ServiceLayer
{
    public class EfCategoryService : ICategoryService
    {
        IUnitOfWork _uow;
        IDbSet<Category> _categories;
        public EfCategoryService(IUnitOfWork uow)
        {
            _uow = uow;
            _categories = _uow.Set<Category>();
        }


        public void AddNewCategory(Category category)
        {
            _categories.Add(category);
        }


        public IList<Category> GetAllCategories()
        {
            return _categories.ToList();
        }
    }
}
```

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.DomainClasses;


namespace EF_Sample07.ServiceLayer
{
    public class EfProductService : IProductService
    {
        IUnitOfWork _uow;
        IDbSet<Product> _products;
        public EfProductService(IUnitOfWork uow)
        {
            _uow = uow;
            _products = _uow.Set<Product>();
        }


        public void AddNewProduct(Product product)
        {
            _products.Add(product);
        }


        public IList<Product> GetAllProducts()
        {
            return _products.Include(x => x.Category).ToList();
        }
    }
}
```

توضیحات:

همانطور که ملاحظه می‌کنید در هیچکدام از کلاس‌های سرویس برنامه، وهله سازی مستقیمی از الگوی واحد کار وجود ندارد. این لایه از برنامه اصلا نمی‌داند که کلاسی به نام Sample07Context وجود خارجی دارد یا خیر.
همچنین لایه اضافی دیگری را به نام Repository جهت مخفی سازی سازوکار EF به برنامه اضافه نکرده‌ایم. این لایه شاید در نگاه اول برنامه را مستقل از ORM جلوه دهد اما در عمل قابل انتقال نیست و سبب تحمیل سربار اضافی بی موردی به برنامه می‌شود؛ ORMها ویژگی‌های یکسانی را ارائه نمی‌دهند. حتی در حالت استفاده از LINQ، پیاده سازی‌های یکسانی را به همراه ندارند.
بنابراین اگر قرار است برنامه مستقل از ORM کار کند، نیاز است لایه استفاده کننده از سرویس برنامه، با دو اینترفیس IProductService و ICategoryService کار کند و نه به صورت مستقیم با پیاده سازی آن‌ها. به این ترتیب هر زمان که لازم شد، فقط باید پیاده سازی‌های کلاس‌های سرویس را تغییر داد؛ باز هم برنامه نهایی بدون نیاز به تغییری کار خواهد کرد.

تا اینجا به معماری پیچیده‌ای نرسیده‌ایم و اصطلاحا over-engineering صورت نگرفته است. یک اینترفیس بسیار ساده IUnitOfWork به برنامه اضافه شده؛ در ادامه این اینترفیس به کلاس‌های سرویس برنامه تزریق شده است (تزریق وابستگی در سازنده کلاس). کلاس‌های سرویس ما «می‌دانند» که EF وجود خارجی دارد و سعی نکرده‌ایم توسط لایه اضافی دیگری آن‌را مخفی کنیم. شیوه کار با IDbSet تعریف شده دقیقا همانند روال متداولی است که با EF Code first کار می‌شود و بسیار طبیعی جلوه می‌کند.

استفاده از الگوی واحد کار و کلاس‌های سرویس تهیه شده در یک برنامه کنسول ویندوزی

در ادامه برای وهله سازی اینترفیس‌های سرویس و واحد کار برنامه، از کتابخانه StructureMap که یاد شد، استفاده خواهیم کرد. بنابراین، تمام برنامه‌های نهایی ارائه شده در این قسمت، ارجاعی را به اسمبلی StructureMap.dll نیاز خواهند داشت.
کدهای برنامه کنسول مثال جاری را در ادامه ملاحظه خواهید کرد:

```
using System.Collections.Generic;
using System.Data.Entity;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.DomainClasses;
using EF_Sample07.ServiceLayer;
using StructureMap;


namespace EF_Sample07
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample07Context,
Configuration>());

            HibernatingRhinos.Profiler.Appender.EntityFramework.EntityFrameworkProfiler.Initialize();
            ObjectFactory.Initialize(x =>
            {
                x.For<IUnitOfWork>().CacheBy(InstanceScope.Hybrid).Use<Sample07Context>();
                x.For<ICategoryService>().Use<EfCategoryService>();
            });


            var uow = ObjectFactory.GetInstance<IUnitOfWork>();
            var categoryService = ObjectFactory.GetInstance<ICategoryService>();


            var product1 = new Product { Name = "P100", Price = 100 };
            var product2 = new Product { Name = "P200", Price = 200 };
            var category1 = new Category
            {
                Name = "Cat100",
                Title = "Title100",
                Products = new List<Product> { product1, product2 }
            };
            categoryService.AddNewCategory(category1);
            uow.SaveChanges();
        }
    }
```

```
}
```

در اینجا بیشتر هدف، معرفی نحوه استفاده از StructureMap است.
ابتدا توسط متد ObjectFactory.Initialize مشخص می‌کنیم که اگر برنامه نیاز به اینترفیس IUnitOfWork داشت، لطفا کلاس Sample07Context را وهله سازی کرده و مورد استفاده قرار بده. اگر ICategoryService مورد استفاده قرار گرفت، وهله مورد نظر باید از کلاس EfCategoryService تامین شود.
توسط ObjectFactory.GetInstance نیز می‌توان به وهله‌ای از این کلاس‌ها دست یافت و نهایتا با فراخوانی uow.SaveChanges می‌توان اطلاعات را ذخیره کرد.

چند نکته:
- به کمک کتابخانه StructureMap، تزریق IUnitOfWork به سازنده کلاس EfCategoryService به صورت خودکار انجام می‌شود. اگر به کدهای فوق دقت کنید ما فقط با اینترفیس‌ها مشغول به کار هستیم، اما وهله‌سازی‌ها در پشت صحنه انجام می‌شود.
- حین معرفی IUnitOfWork از متد CacheBy با پارامتر InstanceScope.Hybrid استفاده شده است. این enum مقادیر زیر را می‌تواند بپذیرد:

```
public enum InstanceScope
{
        PerRequest = 0,
        Singleton = 1,
        ThreadLocal = 2,
        HttpContext = 3,
        Hybrid = 4,
        HttpSession = 5,
        HybridHttpSession = 6,
        Unique = 7,
        Transient = 8,
}
```

برای مثال اگر در برنامه‌ای نیاز داشتید یک کلاس به صورت Singleton عمل کند، فقط کافی است نحوه کش شدن آن‌را تغییر دهید.
حالت PerRequest در برنامه‌های وب کاربرد دارد (و حالت پیش فرض است). با انتخاب آن وهله سازی کلاس مورد نظر به ازای هر درخواست رسیده انجام خواهد شد.
در حالت ThreadLocal، به ازای هر Thread، وهله‌ای متفاوت در اختیار مصرف کننده قرار می‌گیرد.
با انتخاب حالت HttpContext، به ازای هر HttpContext ایجاد شده، کلاس معرفی شده یکبار وهله سازی می‌گردد.
حالت Hybrid ترکیبی است از حالت‌های HttpContext و ThreadLocal. اگر برنامه وب بود، از HttpContext استفاده خواهد کرد در غیراینصورت به ThreadLocal سوئیچ می‌کند.

استفاده از الگوی واحد کار و کلاس‌های سرویس تهیه شده در یک برنامه ASP.NET MVC

یک برنامه خالی ASP.NET MVC را آغاز کنید. سپس یک HomeController جدید را نیز به آن اضافه نمائید و کدهای آن‌را مطابق اطلاعات زیر تغییر دهید:

```
using System.Web.Mvc;
using EF_Sample07.DomainClasses;
using EF_Sample07.ServiceLayer;
using EF_Sample07.DataLayer.Context;
using System.Collections.Generic;


namespace EF_Sample07.MvcAppSample.Controllers
{
    public class HomeController : Controller
    {
        IProductService _productService;
        ICategoryService _categoryService;
        IUnitOfWork _uow;
```

```
        public HomeController(IUnitOfWork uow, IProductService productService, ICategoryService
categoryService)
        {
            _productService = productService;
            _categoryService = categoryService;
            _uow = uow;
        }


        [HttpGet]
        public ActionResult Index()
        {
            var list = _productService.GetAllProducts();
            return View(list);
        }



        [HttpGet]
        public ActionResult Create()
        {
            ViewBag.CategoriesList = new SelectList(_categoryService.GetAllCategories(), "Id", "Name");
            return View();
        }


        [HttpPost]
        public ActionResult Create(Product product)
        {
            if (this.ModelState.IsValid)
            {
                _productService.AddNewProduct(product);
                _uow.SaveChanges();
            }


            return RedirectToAction("Index");
        }


        [HttpGet]
        public ActionResult CreateCategory()
        {
            return View();
        }


        [HttpPost]
        public ActionResult CreateCategory(Category category)
        {
            if (this.ModelState.IsValid)
            {
                _categoryService.AddNewCategory(category);
                _uow.SaveChanges();
            }


            return RedirectToAction("Index");
        }
    }
}
```

نکته مهم این کنترلر، تزریق وابستگی‌ها در سازنده کلاس کنترلر است؛ به این ترتیب کنترلر جاری نمی‌داند که با کدام پیاده سازی خاصی از این اینترفیس‌ها قرار است کار کند.

اگر برنامه را به همین نحو اجرا کنیم، موتور ASP.NET MVC ایراد خواهد گرفت که یک کنترلر باید دارای سازنده‌ای بدون پارامتر باشد تا من بتوانم به صورت خودکار وهله‌ای از آن‌را ایجاد کنم. برای رفع این مشکل از کتابخانه StructureMap برای تزریق خودکار وابستگی‌ها کمک خواهیم گرفت:

```
using System;
using System.Data.Entity;
using System.Web.Mvc;
using System.Web.Routing;
using EF_Sample07.DataLayer.Context;
```

```
using EF_Sample07.ServiceLayer;
using StructureMap;


namespace EF_Sample07.MvcAppSample

{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801


    public class MvcApplication : System.Web.HttpApplication
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new HandleErrorAttribute());
        }


        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");


            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter
defaults
            );
        }


        protected void Application_Start()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample07Context,
Configuration>());
            HibernatingRhinos.Profiler.Appender.EntityFramework.EntityFrameworkProfiler.Initialize();
            AreaRegistration.RegisterAllAreas();
            RegisterGlobalFilters(GlobalFilters.Filters);
            RegisterRoutes(RouteTable.Routes);
            initStructureMap();
        }


        private static void initStructureMap()
        {
            ObjectFactory.Initialize(x =>
            {
                x.For<IUnitOfWork>().HttpContextScoped().Use(() => new Sample07Context());
                x.ForRequestedType<ICategoryService>().TheDefaultIsConcreteType<EfCategoryService>();
                x.ForRequestedType<IProductService>().TheDefaultIsConcreteType<EfProductService>();
            });


            //Set current Controller factory as StructureMapControllerFactory
            ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
        }


        protected void Application_EndRequest(object sender, EventArgs e)
        {
            ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
        }
    }


    public class StructureMapControllerFactory : DefaultControllerFactory
    {
        protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
        {
            return ObjectFactory.GetInstance(controllerType) as Controller;
        }
    }
}
```

توضیحات:

کدهای فوق متعلق به کلاس Global.asax.cs هستند. در اینجا در متد Application_Start، متد initStructureMap فراخوانی شده است.
با پیاده سازی ObjectFactory.Initialize در کدهای برنامه کنسول معرفی شده آشنا شدیم. اینبار فقط حالت کش شدن کلاس Context برنامه را HttpContextScoped قرار داده‌ایم تا به ازای هر درخواست رسیده یک بار الگوی واحد کار وهله سازی شود.
نکته مهمی که در اینجا اضافه شده‌است، استفاده از متد ControllerBuilder.Current.SetControllerFactory می‌باشد. این متد نیاز به وهله‌ای از نوع DefaultControllerFactory دارد که نمونه‌ای از آن‌را در کلاس StructureMapControllerFactory مشاهده می‌کنید. به این ترتیب در زمان وهله سازی خودکار یک کنترلر، اینبار StructureMap وارد عمل شده و وابستگی‌های برنامه را مطابق تعاریف ObjectFactory.Initialize ذکر شده، به سازنده کلاس کنترلر تزریق می‌کند.
همچنین در متد Application_EndRequest با فراخوانی ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects از نشتی اتصالات به بانک اطلاعاتی جلوگیری خواهیم کرد. چون وهله الگوی کار برنامه HttpScoped تعریف شده، در پایان یک درخواست به صورت خودکار توسط StructureMap پاکسازی می‌شود و به نشتی منابع نخواهیم رسید.

استفاده از الگوی واحد کار و کلاس‌های سرویس تهیه شده در یک برنامه ASP.NET Web forms

در یک برنامه ASP.NET Web forms نیز می‌توان این مباحث را پیاده سازی کرد:

```
using System;
using System.Data.Entity;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.ServiceLayer;
using StructureMap;


namespace EF_Sample07.WebFormsAppSample
{
    public class Global : System.Web.HttpApplication
    {
        private static void initStructureMap()
        {
            ObjectFactory.Initialize(x =>
            {
                x.For<IUnitOfWork>().HttpContextScoped().Use(() => new Sample07Context());
                x.ForRequestedType<ICategoryService>().TheDefaultIsConcreteType<EfCategoryService>();
                x.ForRequestedType<IProductService>().TheDefaultIsConcreteType<EfProductService>();


                x.SetAllProperties(y=>
                {
                    y.OfType<IUnitOfWork>();
                    y.OfType<ICategoryService>();
                    y.OfType<IProductService>();
                });
            });
        }


        void Application_Start(object sender, EventArgs e)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample07Context,
Configuration>());
            HibernatingRhinos.Profiler.Appender.EntityFramework.EntityFrameworkProfiler.Initialize();
            initStructureMap();
        }


        void Application_EndRequest(object sender, EventArgs e)
        {
            ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
        }
```

در اینجا کدهای کلاس Global.asax.cs را ملاحظه می‌کنید. توضیحات آن با قسمت ASP.NET MVC آنچنان تفاوتی ندارد و یکی است. البته منهای تعاریف SetAllProperties که جدید است و در ادامه به علت اضافه کردن آن‌ها خواهیم رسید.
در ASP.NET Web forms برخلاف ASP.NET MVC نیاز است کار وهله سازی اینترفیس‌ها را به صورت دستی انجام دهیم. برای این

منظور و کاهش کدهای تکراری برنامه می‌توان یک کلاس پایه را به نحو زیر تعریف کرد:

```
using System.Web.UI;
using StructureMap;


namespace EF_Sample07.WebFormsAppSample
{
    public class BasePage : Page
    {
        public BasePage()
        {
            ObjectFactory.BuildUp(this);
        }
    }
}
```

سپس برای استفاده از آن خواهیم داشت:

```
using System;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.DomainClasses;
using EF_Sample07.ServiceLayer;


namespace EF_Sample07.WebFormsAppSample
{
    public partial class AddProduct : BasePage
    {
        public IUnitOfWork UoW { set; get; }
        public IProductService ProductService { set; get; }
        public ICategoryService CategoryService { set; get; }


        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                bindToCategories();
            }
        }


        private void bindToCategories()
        {
            ddlCategories.DataTextField = "Name";
            ddlCategories.DataValueField = "Id";
            ddlCategories.DataSource = CategoryService.GetAllCategories();
            ddlCategories.DataBind();
        }


        protected void btnAdd_Click(object sender, EventArgs e)
        {
            var product = new Product
            {
                Name = txtName.Text,
                Price = int.Parse(txtPrice.Text),
                CategoryId = int.Parse(ddlCategories.SelectedItem.Value)
            };
            ProductService.AddNewProduct(product);
            UoW.SaveChanges();
            Response.Redirect("~/Default.aspx");
        }
    }
}
```

اینبار وابستگی‌های کلاس افزودن محصولات، به صورت خواصی عمومی تعریف شده‌اند. این خواص عمومی توسط متد SetAllProperties که در فایل global.asax.cs معرفی شدند، باید یکبار تعریف شوند (مهم!).

سپس اگر دقت کرده باشید، اینبار کلاس AddProduct از BasePage ما ارث بری کرده است. در سازند کلاس BasePage، با فراخوانی متد ObjectFactory.BuildUp، تزریق وابستگی‌ها به خواص عمومی کلاس جاری صورت می‌گیرد.
در ادامه نحوه استفاده از این اینترفیس‌ها را جهت مقدار دهی یک DropDownList یا ذخیره سازی اطلاعات یک محصول مشاهده می‌کنید. در اینجا نیز کار با اینترفیس‌ها انجام شده و کلاس جاری دقیقا نمی‌داند که با چه وهله‌ای مشغول به کار است. تنها در زمان اجرا است که توسط StructureMap ، به ازای هر اینترفیس معرفی شده، وهله‌ای مناسب بر اساس تعاریف فایل Global.asax.cs در اختیار برنامه قرار می‌گیرد.

کدهای کامل مثال‌های این سری را از آدرس زیر هم می‌توانید دریافت کنید: (^)

به روز رسانی
کدهای قسمت جاری را به روز شده جهت استفاده از EF 6 و StructureMap 3 در VS 2013، از اینجا می‌توانید دریافت کنید:
EF_Sample07.zip

نظرات خوانندگان

نویسنده: Sh Mjsoft
تاریخ: ۱۳۹۱/۰۲/۲۶ ۱۸:۵۳:۵۱

واقعا ممنون از مطالبت

نویسنده: NTC
تاریخ: ۱۳۹۱/۰۲/۲۶ ۲۲:۳۷:۰۴

این قسمت کمی مشکل بود.کمی گیج شدم
تصویر زیر را ببینید.
در یک Win Application
جای تعاریف درست است؟؟
چرا عبارت "HibernatingRhinos" شناخته نمیشود؟

نویسنده: Sirwan Afifi
تاریخ: ۱۳۹۱/۰۲/۲۶ ۲۲:۴۸:۱۳

سلام استاد خیلی ممنون
یه سوال :

این متد SaveChanges خودش به صورت توکار کار مدیریت کانکشن (Open و Close کردن کانکشن) رو انجام میده مثل متد Fill کلاس SqlDataAdapter در ADO.NET یا روال کار در اینجا به صورت دیگری است؟ در کل منظورم همون بحث Connection oPooling

نویسنده: Sirwan Afifi
تاریخ: ۱۳۹۱/۰۲/۲۶ ۲۲:۵۲:۲۳

استاد راستی برای یادگیری و تسلط به این الگوها شما کتابی خاصی رو میتونید بهم معرفی کنید؟
چون اطلاعاتم در رابطه با این الگو ها کم بود نتونستم از کل مطالبتون استفاده کنم.
در هر حال ممنون از اینکه دانش خودتون رو به اشتراک میذارید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۷ ۰۰:۱۴:۰۲

قبلا در این مورد مطلب نوشتم: (^) | (^)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۷ ۰۰:۱۵:۳۳

مدیریت اتصالات توسط DbContext مدیریت می‌شود؛ با وهله سازی و سپس dispose آن.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۷ ۰۰:۱۶:۵۰

HibernatingRhinos مربوط است به برنامه EF Prodiler که در قسمت 10 توضیح دادم.

نویسنده: amir hosein jelodari
تاریخ: ۱۳۹۱/۰۲/۲۷ ۱۱:۰۶:۱۰

فقط میتونم بگم که دمتون گرم ... یعنی ایول :دی

نویسنده: Hossein Raziee
تاریخ: ۱۳۹۱/۰۲/۲۷ ۱۱:۱۱:۲۴

با درود
ابتدا سپاس به خاطر مطالب بسیار مفیدی که مینویسید.
در یک پروژه وب که به صورت ماژولار تعریف شده باشه و در ابتدا مشخص نباشه که چه امکاناتی داره و قرار باشه در آینده به پروژه اضافه بشه، نمیتونیم Model های مختلف رو در ابتدا در DbContext تعریف کرد.
بنابر این باید برای هر ماژول dll ای تولید کرد که حاوی DomainClass ها , ServiceLayer ها ، Controller ها و DbContext مربوط به اون ماژول باشه.
به نظر شما برای تعریف این قسمت ها در Application_Start باید چه کار کید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۷ ۱۱:۲۹:۴۱

StructureMap امکان اسکن اسمبلی‌های اضافه شده به سیستم را دارد. باید وقت بگذارید مستندات آن‌را مطالعه کنید: (^)

نویسنده: NTC
تاریخ: ۱۳۹۱/۰۲/۲۷ ۱۴:۲۹:۳۰

شرمنده
کتابخانه ی structuremap را هم از NuGet اضافه کردم و هم از Github.
ولی هر دور در زمان اجرا اخطار زیر را میدهند :
"The type or namespace name 'StructureMap' could not be found (are you missing a using directive or an assembly reference?)"

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۲۷ ۱۶:۰۷:۴۷

StructureMap در حالت استفاده از Client profile کار نمی‌کند و load نمی‌شود. علت هم این است که ارجاعی را به اسمبلی System.Web دارد. به همین جهت به خواص پروژه مراجعه کرده و Client profile را به حالت Full تغییر دهید، مشکل حل می‌شود.

نویسنده: Mona
تاریخ: ۱۳۹۱/۰۲/۳۰ ۰۹:۱۱:۵۸

با سلام خدمت آقای نصیری
با تشکر فراوان از مطالب بسیار حوب و سطح بالای شما.
متاسفانه این مطلب کمی برای من سنگین است.
سوال: اگر به هیچ عنوان قصد تغییر ORM را در برنامه نداشته باشیم و فقط بخواهیم از قابلیت های Session/Context Per Request استفاده کنیم و کلاس واسط Service را پیاده سازی کنیم (فرضا در ASP.NET MVC)، امکان دارد راهنمایی بفرمایید یا نمونه ای را معرفی کنید.
با تشکر
(کمی هنگ کردم)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۳۰ ۱۰:۱۱:۳۴

اصل قضیه در اینجا مدیریت Context در طی یک درخواست Http است که به خوبی توسط StructureMap مدیریت می‌شود؛ فقط

با چند سطر کد نویسی (قسمت HttpContextScoped و بعد هم ReleaseAndDisposeAllHttpScopedObjects). اگر بخواهید اینترفیس‌ها را حذف کنید و از StructureMap استفاده نکنید به چند صد سطر کد برای جایگزینی آن خواهید رسید که ضرورتی ندارد.

نویسنده: Mona
تاریخ: ۱۳۹۱/۰۲/۳۰ ۱۰:۵۰:۰۹

بسیار متشکرم از پاسخ کامل و سریع شما.
البته مشکل اصلی من با تعدد کلاس ها و زمانبر بودن پیاده سازی آنها است.
مثلاً برای کلاس Product یک بار باید خود آن را ساخت، یک بار EfProductService و IProductService
آیا راه ساده تری وجود دارد؟ یا من کلاً قضیه را اشتباه متوجه شدم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۳۰ ۱۱:۰۰:۲۰

ساخت اینترفیس از روی کلاس در ویژوال استودیو ساده است. روی نام کلاس کلیک راست کنید. بعد گزینه Refactor و سپس گزینه Extract Interface را انتخاب کنید. با چند کلیک، یک اینترفیس کامل برای شما تولید خواهد شد.

نویسنده: iman
تاریخ: ۱۳۹۱/۰۳/۲۸ ۱۲:۲۶

با سلام خدمت استاد عزیز
اول از همه بی نهایت از مطالب خوبتون تشکر میکنم.مقالات شما در پیشرفت من فوق العاده تاثیر داشت.برای تشکر فقط میتونم بگم واقعاً خسته نباشی و امیدوارم همیشه سلامت باشید.
من این pattern رو در پروژه خودم استفاده کردم و واقعاً سودمند بود.من برای validation سمت سرور از data annotation استفاده کردم و هنگام save changes عمل validation فراخوانی و اجرا میشود.اما چون میخواستم code behind رو کاملاً خلوت کنم ، exception و مقادیر خروجی validation error رو در ایمپلیمنت اینترفیس بررسی کردم و برای اینکار

```
_uow.SaveChanges();
```

رو بجای code behind در ایمپلیمنت اینترفیس نوشتم.این روش درست است؟
آیا شما برای چک کردن validation با code first روش دیگری را پیشنهاد میکنید؟البته با توجه به pattern فوق.
در ضمن پروژه بنده asp.net web form می باشد.
با تشکر از استاد و معذرت بخاطر طولانی شدن سوال

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۲۸ ۱۲:۳۲

سلام؛ من هم تقریبا همین کار رو انجام می‌دم.

```
public class MyDbContextBase : DbContext, IUnitOfWork
```

یک کلاس پایه MyDbContextBase دارم که پیاده سازی IUnitOfWork و DbContext اصلی خود EF رو داره. به این ترتیب حجم کدهای تکراری من کم میشه و از این کلاس پایه استفاده می‌کنم. داخلش در زمان Save تغییرات می‌شود DbEntityValidationException را هم بررسی کرد و مواردی از این دست.
البته اگر از MVC استفاده کنید این data annotation در سمت کلاینت هم به صورت خودکار اعمال می‌شود.

نویسنده: iman
تاریخ: ۱۳۹۱/۰۳/۲۸ ۱۲:۳۹

ممنون استاد.یه سوال دیگه که شاید مربوط به این بخش نباشد.
asp.net web form بر خلاف mvc
attribute برای compare validation ندارد.این مشکل رو چگونه حل کنم؟
حقیقتش خودم خواستم با ایمپلیمنت کلاس validation Attribute اینکار رو انجام بدم ولی نشد.
ممنون میشم کمکم کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۲۸ ۱۳:۴۳

دستی باید این مساله رو مدیریت کنید. سمت سرور آن: دو فیلد دارید که باید مقایسه شوند. سمت کلاینت آن هم در اینجا بحث شده.

نویسنده: iman
تاریخ: ۱۳۹۱/۰۳/۲۸ ۱۴:۷

متشکرم استاد
سمت کلاینت رو با Jquery Validation انجام دادم.مشکلم سمت سرور بود که میخواستم مثل بقیه attribute ها ، واسه این مورد هم از attribue استفاده کنم که انگار راهی نیست و باید دستی انجام داد.
ممنون از وقتی که گذاشتید.هر روز منتظر مطالب پر بارتون هستم

نویسنده: Mona
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۳:۴۶

با سلام خدمت جناب نصیری عزیز
بسیار متشکرم از مطالب شما
آیا امکان دارد روش خودتان را که می‌فرمایید به صورت یک پروژه Open Source ارائه کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۳:۵۶

ترکیبی است از همین مثال سورس باز فوق و قسمت tracking که در مورد یکی کردن ی و ک ارسالی پیشتر بحث شد. مورد بیشتری ندارد. فقط یک try/catch اضافه شده در زمان save نهایی که DbEntityValidationException را بررسی می‌کند.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۷:۳۲

بار دیگر سلام ... سوالی که دارم اینه که آیا استفاده از context (در اینجا sample07context) در لایه‌ی سرویس کار درستی است؟! ... چون بعضی از مواقع به ویژگی‌های کلاس DbContext نیاز میشود مثه Entry !

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۸:۲۰

در طراحی فوق در تمام مواقع در لایه سرویس از IUnitOfWork شرح داده شده استفاده می‌شود؛ که توسط IDbSet های متناظر با اشیاء در معرض دید EF ، با EF ارتباط برقرار می‌کند.

نویسنده: امیرحسین جلوداری

تاریخ: ۱۳۹۱/۰۳/۲۹ ۲۳:۵۰

بنده همین محدودیت رو عرض کردم! ... اصلن چه دلیلی داره که لایه‌ی سرویس رو به استفاده از IunitOfWork محدود کنیم؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۲۹ ۲۳:۵۳

این محدودیت نیست. IUnitOfWork دسترسی کاملی رو به Context جاری به شما می‌ده. مطلب رو یکبار لطفا از ابتدا مطالعه کنید. اگر از IUnitOfWork استفاده نشه باید به صورت مستقیم Context رو وهله سازی کنید. یعنی هر کلاسی یکبار تراکنش خاص خودش را باید باز کند، یکبار کانکشن خاص خودش را. با استفاده از IUnitOfWork اگر متد جاری شما از 10 کلاس هم استفاده کند، تماما با یک وهله از Context کار می‌کنند (یعنی یک کانکشن و یک تراکنش که نحوه صحیح کار به این صورت است).

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۰۳/۳۰ ۰:۱۷

یعنی اگه بخوام فرضا از متد Entry تو DbContext استفاده کنم باید اونو تو اینترفیس IUnitOfWork قرار بدم؟

```
public interface IUnitOfWork
    {
        IDbSet<TEntity> Set<TEntity>() where TEntity : class;
        int SaveChanges();
        DbEntityEntry<TEntity> Entry<TEntity>(TEntity entity) where TEntity : class;
    }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۳۰ ۰:۴۷

بله. اینکار رو میشه انجام داد. در زمان استفاده، نیازی هم به پیاده سازی نداره چون در DbContext تعریف شده و از همان استفاده می‌شود.
مزیت استفاده از اینترفیس در اینجا این است که کتابخانه DI مورد استفاده کار تزریق تنها یک وهله از Context رو به n کلاسی که هم اکنون مثلا در روال جاری کلیک برنامه درگیر هستند و تنها IUnitOfWork رو می‌شناسند به صورت خودکار انجام می‌ده. یک کانکشن؛ یک تراکنش؛ سربار کم و سرعت بالای کار.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۰۳/۳۰ ۱:۱۹

خیلی ممنون ... شیرفهم شدم :دی

نویسنده: iman
تاریخ: ۱۳۹۱/۰۳/۳۰ ۱۲:۲۰

با سلام و تشکر از زحمات استاد نصیری
من چند روز پیش مطلبی رو راجع به ساخت attribute برای compare validation در روش code first عنوان کردم(البته در asp.net web form).که در واقع اصلی‌ترین کاربردش همون کاربرد معروف مقایسه رمز عبور و تکرار رمز عبور هست. یه سری کارایی در این زمینه انجام دادم و خواستم در مورد مشکل مربوطه و در کل ، صحیح یا غلط بودن این روش کمکم کنید.

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Property | AttributeTargets.Field |
AttributeTargets.Parameter, AllowMultiple = true, Inherited = true)]
    public  class CompareAttribute : ValidationAttribute
    {
        public CompareAttribute(string originalProperty, string confirmProperty)
        {
            OriginalProperty = originalProperty;
            ConfirmProperty = confirmProperty;
        }
        public string ConfirmProperty
        {
            get;
```

```
            private set;
        }
        public string OriginalProperty
        {
            get;
            private set;
        }
        public override bool IsValid(object value)
        {
            PropertyDescriptorCollection properties = TypeDescriptor.GetProperties(value);
            return String.Equals(((User)value).Password, ((User)value).RepeatPassword);
        }
    }
```

تا اینجا یه کلاس تعریف شده که خیلی خلاصه Validation Attribute رو پیاده سازی کرده.این هم از کلاس User ، فقط در حد تعریف propertyهای لازم این مثال

```
//[CompareAttribute("Password", "RepeatPassword", ErrorMessage = "Not Compare!")]
    public class User
    {
        public Int64 UserId { get; set; }




        [Required(ErrorMessageResourceName = "Password", ErrorMessageResourceType =
typeof(ErrorMessageResource))]
        public String Password { get; set; }




        [NotMapped]
        [Required(ErrorMessageResourceName = "RepeatPassword", ErrorMessageResourceType =
typeof(ErrorMessageResource))]
       //[CompareAttribute("Password","RepeatPassword" , ErrorMessage = "Not Compare!")]
        public String RepeatPassword { get; set; }
    }
```

مشکل اصلی استفاده از همین compare attribute هست که ساخته شده .
وقتی اونو بالای کلاس User میزارم کار میکنه.ولی خب این یه کار خیلی زشتیه! چون به ازای همه propertyها اجرا میشه.ولی وقتی اونو تو جای اصلیش که همون بالای repeat password هست میزارم کار نمیکنه.
یه جورایی مشکل از اینه که نمیشه انگار مقدار property رو به این سمت پاس داد.ولی در حالتی که بالای کلاس میزارمش چون خود کلاس رو پاس میده ، طبیعتا*
اسم propertyها رو هم پیدا میکنه.
ممنون میشم راهنماییم کنید .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۳۰ ۱۳:۴۵

به این ترتیب باید پیاده سازی بشه. یک حالت عمومی است و به کلاس و شیء خاصی گره نخورده:

```
using System;
using System.ComponentModel.DataAnnotations;
namespace Test
{
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
    public class CompareAttribute : ValidationAttribute
    {
        public CompareAttribute(string originalProperty, string confirmProperty)
        {
            OriginalProperty = originalProperty;
            ConfirmProperty = confirmProperty;
```

```
        }
        public string ConfirmProperty { get; private set; }
        public string OriginalProperty { get; private set; }
        protected override ValidationResult IsValid(object value, ValidationContext ctx)
        {
            if (value == null)
                return new ValidationResult("لطفا فیلدها را تکمیل نمائید");
            var confirmProperty = ctx.ObjectType.GetProperty(ConfirmProperty);
            if (confirmProperty == null)
                throw new InvalidOperationException(string.Format("لطفا فیلد {0} را تعریف نمائید",
ConfirmProperty));
            var confirmValue = confirmProperty.GetValue(ctx.ObjectInstance, null) as string;
            if (string.IsNullOrWhiteSpace(confirmValue))
                return new ValidationResult(string.Format("لطفا فیلد {0} را تکمیل نمائید",
ConfirmProperty));
            var originalProperty = ctx.ObjectType.GetProperty(OriginalProperty);
            if (originalProperty == null)
                throw new InvalidOperationException(string.Format("لطفا فیلد {0} را تعریف نمائید",
OriginalProperty));
            var originalValue = originalProperty.GetValue(ctx.ObjectInstance, null) as string;
            if (string.IsNullOrWhiteSpace(originalValue))
                return new ValidationResult(string.Format("لطفا فیلد {0} را تکمیل نمائید",
OriginalProperty));
            return originalValue == confirmValue ? ValidationResult.Success : new
ValidationResult("مقادیر وارد شده یکسان نیستند");
        }
    }
}
```

نویسنده: iman
تاریخ: ۱۳۹۱/۰۳/۳۰ ۱۴:۲۸

مثل همیشه کامل و بی نقص
عالی بود.ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۳۰ ۱۸:۲

خواهش می‌کنم.
لطفا از این پس مطالبی رو که در قسمت نظرات عنوان می‌کنید متناسب با عنوان موضوع جاری آن (برای مثال context per request در اینجا) باشد. با تشکر.

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۰:۲۷

سلام؛
من از وهله سازی نوع HybridHttpOrThreadLocalScoped در یک برنامه ویندوزی استفاده کردم.اما فکر میکنم چون حالت ThreadLocal رو برای این حالت انتخاب میکنه ، وهله مربوط به اشیاء ساخته شده تا زمان بستن برنامه در حافظه باقی بمونه.از این رو فکر میکنم چون وهله ساخته شده از بین نمیره ، کانکشن به DataBase تا زمان بسته شدن برنامه باز بمونه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۰:۴۸

توضیحات بالا برای برنامه‌های وب بهینه شده. در آنجا در Application_EndRequest به صورت خودکار کانکشن بسته میشه (با کد ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects که نوشته شد).
در برنامه‌های ویندوزی این مدیریت رو باید خودتون دستی انجام بدید و چنین مکانیزمی در آن طراحی نشده.

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۶:۱۶

سلام آقای نصیری . خسته نباشید
همونطور که منو توی یکی از کامنت‌ها به اینجا ارجاع داه بودین، مطلب رو خوندم. ولی متاسفانه کمی گیج شدم. من توی پروژه ام از روش EF Code First استفاده نکردم. روش من اینجوری بود که دیتابیس رو ساختم و بعد از روی اون فایل .edmx رو ساختم و باهاش توی کلاس هام کار کردم. حالا میخواستم بدونم آیا من می‌تونم با توجه به این موضوع از توضیحاتی که در بالا گفتین استفاده کنم یا حتما باید EF Code First باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۴ ۱۶:۳۷

طراحی فوق برای روش database first نیست و بحث اینجا متفاوت است. اینترفیس‌های تعریف شده مطلب فوق و زیرساخت آن متفاوت است با database first.
ضمن اینکه از ef code first برای کار با بانک اطلاعاتی موجود هم می‌شود استفاده کرد. روش و ابزار مهندسی معکوس آن وجود دارد: (^)

نویسنده: میثم
تاریخ: ۱۳۹۱/۰۴/۱۶ ۲۲:۵۷

سلام و خسته نباشید استاد نصیری

سوال بنده اینه که استفاده از لایه سرویس ضرورتی داره ؟ مشکلی پیش میاد اگه این لایه رو به طور کامل خذف کنیم و مستقیم با UnitOfWork کار کنیم؟

چون با استفاده از این لایه به ازای یک عمل مانند ذخیره کردن یک شی در پایگاه داده باید 2 شی (یکی از لایه مدل و دیگری از لایه سرویس) تعریف بشه ضمن آنکه وقتی تعداد کلاس‌ها زیاد بشه متد initStructureMap() پیچیده میشه

تشکر فراوان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۶ ۲۳:۴۰

منهای تمام مباحثی که عنوان شد، یکی دیگر از مزایای استفاده از لایه سرویس، جدا سازی منطقی قسمت‌های مختلف برنامه از هم است. به این ترتیب الان شما دقیقا می‌دونید اعمال کار با یک موجودیت دقیقا در کدام کلاس قرار گرفته و مرتبا در قسمت‌های مختلف برنامه پراکنده و تکرار نشده. اگر مشکلی وجود داشته باشد، در یکجا باید اصلاح شده و اثرش به صورت خودکار به تمام برنامه اعمال می‌شود.

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۴/۲۳ ۱۴:۵۸

سلام مهندس
توی پیاده سازی قسمت MVC شما کد زیر توی فایل global فراخوانی کردین

```
private static void initStructureMap()
        {
            ObjectFactory.Initialize(x =>
            {
```

```
                x.For<IUnitOfWork>().HttpContextScoped().Use(() => new Sample07Context());
                x.ForRequestedType<ICategoryService>().TheDefaultIsConcreteType<EfCategoryService>();
                x.ForRequestedType<IProductService>().TheDefaultIsConcreteType<EfProductService>();
            });
            //Set current Controller factory as StructureMapControllerFactory
            ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
        }
```

اینجا تعداد Entityهای ما از قبل ثابت و مشخصه اگر خواستیم به این لیست Entityهای جدیدی اضافه بشه چکار باید بکنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۳ ۱۶:۳۵

امکان اسکن اسمبلی‌های اضافه شده به سیستم هم وجود دارد: (^)

نویسنده: میثم
تاریخ: ۱۳۹۱/۰۴/۲۴ ۱۳:۱۰

سلام و خسته نباشید استاد
آیا نحوه استفاده که برای asp.net و asp.net MVC معرفی نمودید کامل است ، یعنی با ناتمام ماندن یک درخواست از سمت کلاینت (به هر دلیلی) منابع به طور کامل آزاد میگردند و یا نیاز است تا با استفاده از HttpModule اشیا نابود بشوند.
شبیه چیزی که تو NH انجام میدادیم یا اینجا ذکر شده است
همیشه دعایتان می‌کنم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۴ ۱۳:۳۲

HttpModule همان اعمال قابل تنظیم در global.asax را در اختیار شما قرار می‌دهد. فقط اینبار کدهای شما اندکی نظم بهتری پیدا می‌کنند؛ به علاوه تعدادی روال رخدادگردان بیشتر نیز دراختیار شما خواهند بود.
در global.asax.cs دارید:

```
protected void Application_BeginRequest
```

در یک ماژول خواهید داشت:

```
public void Init(HttpApplication context)
{
    context.BeginRequest += beginRequest;
}
```

و الی آخر.

نویسنده: بابک
تاریخ: ۱۳۹۱/۰۴/۳۱ ۳:۲۶

در این روش شما نحوه ویرایش رکورد را چطور انجام می‌دهید؟ می‌خواهم در متد add در صورتیکه رکورد موجود باشد update شود و اگر نباشد در دیتابیس اینزرت شود یا اینکه 2 متد جدا به اسم add. edit در لایه سرویس داشته باشم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۳۱ ۸:۳۷

این یک سری به هم پیوسته است.
در مورد add or update
در مورد نحوه صحیح به روز رسانی اطلاعات و اشتباهات متداول مرتبط

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۴/۳۱ ۱۴:۱

ممنون از پاسخ شما
اینجا هم نمونه خوبی از Scan گذاشته

نویسنده: بابک
تاریخ: ۱۳۹۱/۰۵/۰۱ ۰:۱۶

من از EF 4.3.1 استفاده می‌کنم ولی متد AddOrUpdate ندارد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۱ ۰:۲۱

در فضای نام System.Data.Entity.Migrations قرار دارد.

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۵/۰۳ ۰:۴۷

سلام
مهندس نصیری این لینک یه نگاه بندازین بد نیست ظاهرا که کامل کار کرده روی موضوع شما

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۵/۰۳ ۰:۵۰

اینجا هم نمونه خوبی از Scan کردن اسمبلی‌ها گذاشته

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۳ ۱:۰

مربوط است به db first و این مشکلات را دارد:
- کلاس واحد کار رو استاتیک تعریف کرده. این مورد در یک برنامه asp.net یعنی به اشتراک گذاری واحد کار جاری با تمام کاربران سایت.
- از StructureMap استفاده کرده اما چون درک درستی از تزریق وابستگی‌ها نداشته از الگوی service locator آن (ObjectFactory.GetInstance) برای وهله سازی استفاده کرده (از این مورد فقط در حالت‌های ناچاری مانند تهیه یک role provider سفارشی که وهله سازی آن در کنترل ما نیست و راسا مدیریت می‌شود باید استفاده کرد)
- از StructureMap استفاده کرده اما نمی‌دونسته که این کتابخانه خودش می‌تونه در پایان درخواست‌های وب اشیاء مورد استفاده رو dispose کنه و کار اضافی انجام داده.
و ....

نویسنده: صابر فتح اللهی
تاریخ: ۱۳۹۱/۰۵/۰۳ ۹:۲۴

ممنونم مهندس

مثل همیشه کامل و بی نقص

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۱:۴۷

من همین پیاده سازی رو انجام دادم و در متد Seed هم دیتای اولیه رو قرار دادم ولی هر دفعه حین اجرای برنامه این متد فراخوانی میشه و دیتای تکراری وارد بانک میکنه، میشه جلوی این کار رو گرفت یا نه، مگر این تنظیم

```
void Application_Start(object sender, EventArgs e)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<Sample07Context,
Configuration>());
```

برای بروز کردن بانک نیست

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۶ ۱۱:۵۲

به این ترتیب طراحی شده. نظر یکی از اعضای تیم EF در این مورد: (^). اگر می‌خواهید رکورد تکراری ثبت نشود از متد AddOrUpdate استفاده کنید.

استفاده‌ای که من از متد seed می‌کنم در عمل، تعریف قیودی مانند unique است با sql نویسی (داخل try/catch البته).

نویسنده: عرفان رضایی
تاریخ: ۱۳۹۱/۰۶/۰۷ ۱۲:۱۲

سلام آقای نصیری،

یه سوال داشتم ازتون:

در عمل ما تو یه برنامه‌ی وب (مثلا mvc) به متدهای زیادی فقط برای سرویس دهی به موضوعات احتیاج داریم،مثلاً

getTopCategories

getLastCategories

(getCategoryByID(int Id

(getCategoriesByDate(Datetime date

()getProductCategories

و ...

حالا سوالم اینه پیاده سازی‌های این متدها باید تو کدوم لایه انجام بشه؟

-1مثلا باید لیست موضوعات،(همین متد GetAllCategories مربوط به سرویس شما)، رو از سرویس برگردوند و داخل controller کویری‌های Linq رو روش اجرا کرد و اطلاعاتی که میخوایم رو ازش بکشیم بیرون و نمایش داد؟

-2یا باید توی اینترفیس ICategoryService **تک تک متد هایی که احتیاج داریم** رو تعریف و بعد توی EFCategoryService تو لایه سرویس اونارو پیاده سازی کنیم و فقط نتیجه رو به controller برگردوند و ازش استفاده کرد(یعنی تو controller فقط پارامترهای مورد نیاز متدهای لایه سرویس رو بهش پاس کنیم)؟

امیدوارم مفهوم رو رسونده باشم.

ممنون به خاطر زحمت هایی که می‌کشید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۰۷ ۱۲:۱۸

حالت دوم صحیح است. تمام پیاده سازی‌ها باید در لایه سرویس باشند. استفاده نهایی در یک کنترلر یا code behind و امثال آن.

نویسنده: عرفان رضایی

تاریخ: ۱۳۹۱/۰۶/۰۷ ۱۲:۲۵

واقعاً ممنونم از سرعتتون

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۰:۱۸

سلام آقای نصیری،
شما تو این مقاله گفتید که:
"همچنین نیازی به پیاده سازی متد SaveChanges نیست؛ زیرا پیاده سازی آن در کلاس DbContext قرار دارد."
ولی این متد رو تو اینترفیس IUnitOfWork ذکر کردید،اینجوری که اگه پیاده سازیه این متد رو انجام ندیم ارور میده!

بالاخره این متد باید توی اینترفیس IUnitOfWork ذکر بشه و توی Context هم پیاده سازی بشه
یا
توی اینترفیس IUnitOfWork ذکر نشه که در اینصورت نیاز باشه توی Context هم پیاده سازی بشه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۰:۳۷

- پیاده سازی متد SaveChanges در کلاس پایه DbContext که توسط تیم EF ارائه شده، انجام شده و وجود دارد.
- ذکر یک متد در اینترفیس (یک قرار داد) به جهت امکان استفاده از آن است. شما نهایتا با متدهای تعریف شده در طراحی IUnitOfWork در لایه سرویس قرار است کار کنید نه مستقیما با کلاس مشتق شده از DbContext.
زمانیکه می‌نویسید:

```
public class MyContext : DbContext, IUnitOfWork
```

چند کار با هم انجام می‌شود:
MyContext به صورت خودکار امکان دسترسی به متد SaveChanges موجود در DbContext را پیدا می‌کند. کتابخانه StructureMap می‌تونه زمانیکه نیازی به یک وهله پیاده ساز IUnitOfWork بود، از MyContext استفاده کنه. همچنین چون الان SaveChanges با امضایی که در اینترفیس IUnitOfWork وجود دارد در کلاس MyContext هم قابل دسترسی است، نیازی به پیاده سازی مجدد آن نیست.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۰:۴۴

آخه برای بنده ارور میده که این متد حتماً باید پیاده سازی بشه!
منظور شما این نیست که باید تو پیاده سازی متد save changes اینترفیس IUnitOfWork فقط کد زیر رو بنویسیم؟

```
Return Base.SaveChanges();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۰:۵۱

کدهای کامل و قابل کامپایل مورد نظر این قسمت از این آدرس قابل دریافت است. می‌تونید وقت بذارید و با کدهای خودتون مقایسه‌اش کنید.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۱:۱

ببخشید آقای نصیری من این کدا رو دیدم،یه سوالم پیش اومد،اینکه همونطور که اینجا عنوان شد ما باید پیاده سازی اینترفیس IUnitOfWork رو تو کلاس MyDbContextBase انجام بدیم،حالا پیاده سازی متد SaveChanges اینترفیس IUnitOfWork تو اونجا باید به صورت زیر باشه؟

```
try
{
applyCorrectYeKe();


auditFields();


//and another methods ...


Return base.SaveChanges();
}
catch (DbEntityValidationException validationException)
{
   //...
}
catch (DbUpdateConcurrencyException concurrencyException)
{
   //...
}
catch (DbUpdateException updateException)
{
   //...
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۱:۱۹

بله. اگر این‌ها رو بخواهید با هم ترکیب کنید همین شکلی است.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۱:۲۵

نمیدونم چطور خوبیتونو جبران کنم ...
فقط میتونم بگم ممنونم ...

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۱:۳۹

بازم ببخشید آقای نصیری،
در اینصورت به پیاده سازی متد SaveChanges اینترفیس IUnitOfWork باید **Overrides** هم اضافه بشه که کامپایلر بدونه متد SaveChanges اینترفیس IUnitOfWork متد SaveChanges کلاس DbContext رو تحریف کرده ،درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۱:۴۳

اگر این متد، return base.SaveChanges را بر می‌گرداند نیازی به ذکر override نیست و می‌تونید برای متد SaveChanges حتی پارامتر هم تعریف کنید (البته اینترفیس هم باید به همین شکل اصلاح شود):

```
public int SaveChanges(string userName, bool updateAuditFields = true)
```

این شکلی است که من خودم ازش استفاده می‌کنم.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۱:۵۳

به خدا گیج شدم،این هشدارها رو برای من میده:

```
Warning   1   function 'SaveChanges' shadows an overridable method in the base class 'DbContext'. To
override the base method, this method must be declared 'Overrides'.
```

در صورتیکه شما گفتید "اگر این متد، return base.SaveChanges را بر می‌گرداند نیازی به ذکر override نیست"!

```
Warning 2   Function 'SaveChanges' doesn't return a value on all code paths. Are you missing a 'Return'
statement?
```

اینم پیاده سازی متد SaveChanges اینترفیس IUnitOfWork من هستش:

```
Public Function SaveChanges() As Integer Implements IUnitOfWork.SaveChanges
            Try
                ApplyCorrectYeKe()
                'auditFields()
                Return MyBase.SaveChanges()
            Catch validationException As DbEntityValidationException
                '...
            Catch concurrencyException As DbUpdateConcurrencyException
                '...
            Catch updateException As DbUpdateException
                '...
            End Try
        End Function
```

به نظر شما مشکل چیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۲:۰

- من تمام مطالبی رو که اینجا عنوان کردم در مورد سی شارپ بود و الان در کارهای خودم دارم ازش استفاده می‌کنم. نمونه قابل کامپایل هم در سایت گذاشتم که لینکش رو دادم.
- این متد SaveChanges آخری با امضای جدید آن، دیگر متد SaveChanges کلاس پایه رو مخفی نمی‌کنه. به همین جهت نیازی به override نداره. بحث من در این مورد بود. نهایتا شما قراره با IUnitOfWork کار کنید. نام این متد رو اصلا تغییر بدید به ApplyChanges بعد هم داخل آن کارهای خودتون رو قرار بدید و دست آخر return base.SaveChanges بازگشت داده شود. ضرورتی ندارد حتما در این اینترفیس از نام SaveChanges استفاده شود. این یک انتخاب بود، بر اساس قسمت 12 جاری که ترکیبی نیست از چند قسمت دیگر. به این صورت می‌شد مبحث رو ساده‌تر و طبیعی‌تر توضیح داد.

نویسنده: عرفان
تاریخ: ۱۳۹۱/۰۶/۱۶ ۲۲:۶

ظاهراً تو VB باید **Overrides** هم اضافه بشه.

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۶/۲۱ ۱۵:۴۵

بنظرتون میشه این قسمتی که مربوط به StructureMap هست رو بیرون از پروژه سرویس نگهداری کرد یا نه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۱ ۱۶:۱۶

در مثال (^) فوق، تنظیمات StructureMap فقط در فایل Global.asax.cs برنامه‌های MVC و WebForms تعریف شدند نه در لایه سرویس.

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۶/۲۱ ۱۶:۴۱

ممنون از جوابتون، تو نگاه به این پروژه بنظرم اومد شاید بشه این تنظیمات رو توی یک پروژه جداگانه نگهداری کرد!

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۶/۲۶ ۱۱:۴۰

سلام؛
کد زیر که درون IUow تعریف شده آیا برای NH هم قابل استفاده است؟ آیا مستقل از Orm است؟

```
IDbSet<TEntity> Set<TEntity>() where TEntity : class;
```

درون Wpf من نیاز به خاصیت Local دارم که ازنوع ObservableCollection است . آیا درست است که از لایه Service این نوع را برگردانم ؟ آیا برای پیاده سازی NH نیز این قابل استفاده است ؟ آخه من درون Vm مجبورم

```
_uow.Set<Person>().Local
```

استفاده کنم.
در صورت امکان راهنمایی کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۶ ۱۲:۳

- IDbSet یک اینترفیس است؛ پیاده سازی نیست. اینترفیسی به همین نام را برای هر ORM دلخواه دیگری طراحی و پیاده سازی کنید. کدهای شما قابل انتقال خواهند بود.
- بله. دسترسی به خاصیت Local داخل لایه سرویس صورت گرفته و کپسوله شده. به همین جهت امضای متدی که ارائه شده به هر ORM دیگری نیز قابل انتقال است. فقط در آنجا یک تبدیل لیست به ObservableCollection را در پیاده سازی داخلی لایه سرویس خود خواهید داشت. اما استفاده کننده نهایی فقط با اینترفیس و قراردادهای تعریف شده در آن هست که کار می‌کند و کاری به جزئیات پیاده سازی لایه سرویس شما ندارد.
به همین جهت است که در اینجا کار با اینترفیس‌ها و قراردادها ترویج شده؛ تا جزئیات پیاده سازی لایه سرویس از دید استفاده کننده مخفی باقی بماند.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۶/۲۶ ۱۴:۵

متشکرم آقای نصیری.
الان حدود 3 ماه میشه هر روز به سایتتون سر میزنم، این مطالب منو بیشتر به برنامه نویسی علاقه مند کرد. مطالب عالی شما .
نمی‌دونم چطوری تشکر کنم از شما، ولی می‌دونم که شناسایی حق در حق شناسیست ، ممنون.

نویسنده: رضا

تاریخ: ۱۳۹۱/۰۷/۰۱ ۱۱:۳۰

آقای نصیری من دوتا سوال داشتم:
اول اینکه معادل دستور زیر، در صورتی که بخواهیم از این روش استفاده کنیم چه چیزی میشود؟

```
db.Entry(post).State = EntityState.Modified;
```

و سوال بعدیم اینکه برای DI چه کتابخوانه ای رو توصیه میکنید؟ همین StructureMap یا Ninject و ...؟
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۱ ۱۲:۱۵

- نیاز خواهید داشت یک چنین تعریفی را اضافه کنید:

```
public interface IUnitOfWork
{
   //...
   DbEntityEntry<TEntity> Entry<TEntity>(TEntity entity) where TEntity : class;
}
```

+ در این حالت این IUnitOfWork آنچنان قابل انتقال و تعویض نخواهد بود چون DbEntityEntry کلاس خاصی است در EF و در سایر ORMها معادلی ندارد. اما اگر فقط با EF کار می‌کنید و قصد تعویض ORM را ندارید، این روش کار می‌کنه و مناسب است.
- از این موارد زیاد است. فرقی هم نمی‌کند (سرعت هم به تنهایی ملاک نیست). با هر کدام که راحت هستید، همان مطلوب است.

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۷/۰۵ ۹:۵۱

با سلام و خسته نباشید
آیا برای مسائلی نظیر transaction commit و transaction rollback در الگوی UOW راهی وجود داره؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۵ ۱۰:۳

طراحی EF Code first مبتنی است بر روان بودن و سادگی؛ هر چند پشت صحنه ساده‌ای ندارد. هر وهله از DbContext به صورت خودکار یک تراکنش را تشکیل می‌دهد و در زمان بسته شدن و dispose، این تراکنش را commit می‌کند (همچنین متد SaveChanges در پشت صحنه از تراکنش‌ها بهره می‌گیرد). هر استثنایی این بین رخ دهد، تراکنش rollback خواهد شد. به همین جهت الگوی واحد کار مطرح شده تا تعداد وهله‌های زیادی از DbContext هربار ایجاد نشوند و کل عملیات در یک DbContext یا یک تراکنش انجام شود.

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۷/۰۵ ۱۰:۵۱

ممنون از پاسخ کاملتون.
فقط برای روشنتر شدن موضوع این سوال را میپرسم.
پس یعنی وقتی که ما در پیاده سازی متد یک Interface ، درج و حذف و بروزرسانی‌های مختلفی در آن متد داریم و در نتیجه چند بار از saveChanges استفاده میکنیم.

در این حالت کلیه این عملیات در قالب یک transaction به حساب می‌آید و در صورت بروز استثنا ، کل عملیات داخل متد rollback خواهد شد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۵ ۱۱:۳۶

خیر. به ازای هر SaveChanges یک تراکنش خاتمه یافته و تراکنش جدیدی آغاز می‌شود (این موارد رو می‌تونید با SQL Server Profiler دقیقا مشاهده کنید).
+ ضرورتی ندارد در یک تراکنش، از چندین و چند SaveChanges استفاده کنید؛ از این جهت که EF از مکانیزم Tracking برخوردار است و می‌تواند با یک SaveChanges ، چندین و چند عملیات insert و update را (بهینه‌ترین حالتی را که محاسبه کرده) با هم در طی یک تراکنش بر اساس مواردی که ردیابی کرده، انجام دهد.

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۷/۰۵ ۱۱:۵۲

ممنون مهندس بابت پاسخ کاملتون
من ذهنم به سمت TransactionScope رفته بود و به Tracking در EF توجه نکرده بودم.
متشکرم.

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۷/۰۹ ۱۲:۵۱

با عرض سلام
چه موقع نیاز است از UnitOfWork در EF استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۹ ۱۳:۵

در تمام برنامه‌های واقعی که حالت مثال نداشته باشند.

نویسنده: kia
تاریخ: ۱۳۹۱/۰۷/۰۹ ۱۷:۲

سلام و ممنون از این سری codeFirst. یک سوال؟

چرا توی لایه **UI** مستقیم به لایه **DL** **دسترسی** پیدا کردین؟ فقط چون مثال هست اینکارو کردین؟
چون چیزی که هست گفته می‌شه اینکارو نکنین.
و در یک پروژه واقعی به چه صورت باید کار کرد؟ چجوری می‌شه UOW (که در DL هست) رو مخفی کرد از دید UI؟ یا اصلا همچین کاری باید کرد؟ (منظور اینکه راه درست استفاده در سمت UI وقتی که قراره با این دید بریم جلو به چه صورت است؟ چون در نهایت باید یک Container ای باشه ک موجودیتها در داخل اون باشن و اون کانتینر بتونه کارایی مثل Transaction رو انجام بده.
من روی WinForm الان می‌خواستم پیاده کنم که با این مسئله به مشکل برخوردم)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۹ ۱۷:۱۶

خیر. زمانیکه از EF استفاده می‌کنید، DAL همان EF است و تنظیمات آن.
پیشنهاد من این است که یکبار پروژه مربوطه رو دریافت کنید و بعد پروژه‌های DataLayer و همچنین ServiceLayer و غیره آن‌را بررسی کنید.
در این مثال‌ها فقط از اینترفیس‌های ServiceLayer (و نه DataLayer مجزای آن) به کمک ترزیق وابستگی‌ها در لایه نمایشی

استفاده شده.

نویسنده: kia
تاریخ: ۱۳۹۱/۰۷/۰۹ ۲۰:۵۵

اینکه DAL همان EF هست درست، من پروژه رو اجرا و بررسی کردم و مشکلی با روالها ندارم الانم مطالبی می‌خونم از این پست و کامنتاش و البته جاهای دیگه راجع به ابهاماتی که برام هست هنوز اما راستش هنوز جواب سوالمو نگرفتم:
شما می‌گین که

در این مثال‌ها فقط از اینترفیس‌های ServiceLayer (و نه DataLayer مجزای آن) به کمک ترزیق وابستگی‌ها در لایه نمایشی استفاده شده.
اما چرا (مثلا) در پروژه Console EF_Sample07 (یا همون لایه نمایشی **UI**) **رفرنسی به DataLayer** زده شده و از UOW که اینترفیسی در لایه DL هست استفاده شده؟ ایا اینکار یکسری قراردادها رو نقض نمی‌کنه؟ +

```
using EF_Sample07.DataLayer.Context;
```

```
_uow.SaveChanges();
```

درسته ک یکسری قرارداد هست این چیزا ولی هرچی خوندم بیشتر در مورد این بود که از سمت UI هیچ دسترسی ای به DL نباید باشه و UI با ServiceLayer یا BL در تعامل باید باشه.
مثلا در برنامه ای که بصورت nTier قراره اجرا بشه اینکار مشکل ساز خواهد بود و شاید اصلا مجوز قرارگیری DAL روی سیستمی که UI هست داده نشه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۹ ۲۱:۱۲

- من در مورد الگوی مخزن در قسمت 11 این سری بحث کردم (کامنت‌های آن‌را هم بخوانید)؛ همچنین این مباحث در مورد EF Code first است و نه db first و نه EF 4. به علاوه این لینک‌هایی که مطرح کردید مثلا نمونه code project، داخل به اصطلاح BLL خودش، پر است از وهله سازی Context و من در این مطلب توضیح دادم که چرا اینکار غلط است و چگونه استفاده از یک تراکنش برای چندین عملیات مرتبط را زیر سؤال می‌برد.
- اون اینترفیس IUnitOfWork مطرح شده در مثال جاری، وابستگی خاصی به DataLayer نداره. می‌تونه در لایه سرویس هم تعریف بشه (منظور این است می‌تونه در یک اسمبلی و پروژه جداگانه هم قرار بگیره و مشکلی نیست). اما DataLayer باید بتونه در حین تزریق وابستگی‌ها وهله‌ای از IUnitOfWork رو فراهم کنه تا به اون معنا ببخشه؛ به همین جهت Context برنامه باید آن‌را پیاده سازی کند تا توسط StructureMap قابل شناسایی و استفاده شود.
اما نهایتا وهله سازی اینترفیس یاد شده توسط DAL صورت می‌گیره. uow به خودی خود موجودیتی نداره. در اینجا مثلا EF هست که به اون معنا می‌بخشه و سبب وهله سازی آن خواهد شد. هرچند به ظاهر برنامه با اینترفیس‌ها کار می‌کند اما تزریق وابستگی‌ها است که به این اینترفیس‌ها موجودیت می‌بخشد و سبب دسترسی به وهله‌ای که قرار داد ارائه شده توسط آن‌ها را پیاده سازی کرده می‌شود.
- در یک سیستم nTier هم مباحث ذکر شده در این قسمت، جاری است. مثلا یک WCF Service قرار گرفته روی یک سرور مجزا هم می‌تونه از DataLayer و ServiceLayer مثال جاری استفاده کند. استفاده کننده نهایی برای نمایش آن در UI با هیچکدام از دو مورد ذکر شده کاری ندارد و فقط با قراردادهای WCF Service کار می‌کنه.

نویسنده: مهمان
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۱:۳۲

با عرض سلام
با اجرای این sample خطای زیر رخ داد. علت خطای زیر چیست؟

CreateDatabase is not supported by the provider. **Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.ProviderIncompatibleException: CreateDatabase is not supported by the provider

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۱:۵۵

سطر «HibernatingRhinos.Profiler» را حذف کنید. اطلاعات بیشتر در اینجا .

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۷/۱۸ ۱۶:۳۱

با تشکر از مطالبتون
در هنگام دیباگ نسخه کنسول این مثال و ردگیری کوئری‌ها در EFProf به نظر میاد که کوئری‌های Insert درست بعد از uow.SaveChanges اجرا نمی‌گردند؛ بلکه این امر در انتهای کار و پس از از بین رفتن ابجکت uow صورت می‌پذیرد. قاعدتا با توجه به توضیحاتتان نباید این اتفاق می‌افتاد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۸ ۲۲:۴۸

علت این است که **یک** SaveChanges در اینجا تعریف شده (ضمنا زمان اجرای این‌ها مهم نیست. بحث ما در مورد تعدد تراکنش‌ها بود). چند سطر زیر را اضافه کنید بعد از سطر آخر و مجددا تست کنید:

```
//...
uow.SaveChanges();
//...
var product3 = new Product { Name = "P300", Price = 300 };
var category2 = new Category
{
                Name = "Cat200",
                Title = "Title200",
                Products = new List<Product> { product3 }
};
categoryService.AddNewCategory(category2);
uow.SaveChanges();
```

من دو تراکنش مجزا رو در برنامه مطمئن و تست شده SQL Server Profiler مشاهده می‌کنم (به ازای هربار فراخوانی SaveChanges).

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۷/۱۹ ۰:۱۲

آقای نصیری این پیاده سازی (Context per request) مناسب برنامه‌های Silverlight هست ؟ یا لزومی دارد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۹ ۰:۱۹

ORMها کلا در سیلورلایت مستقیما قابل استفاده نیستند چون سیلورلایت سمت کاربر اجرا می‌شود و دسترسی کاملی هم به کل دات نت ندارد. سیلورلایت از طریق سرویس‌های WCF می‌تونه با سرور ارتباط برقرار کنه و این مباحث در سرویس‌های WCF هم قابل استفاده است.
البته برای سیلورلایت WCF RIA Services تعریف شده که روش مرجح است و در آن امکان دسترسی به EF Code first وجود دارد .

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۷/۱۹ ۷:۵۴

با سپاس
با SQL Server Profiler تست شد و نتیجه درست بود.
احتمالا مسئله بر می‌گرده به عدم آشنایی من با طرز کار دقیق EFProf.

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۳۹۱/۰۷/۲۶ ۲۱:۳۵

وقت بخیر مهندس نصیری. خسته نباشید.
یک سوال.
در لایه سرویس اگر یک عملیات مشترک باشد (به عنوان مثال درهم سازی (Hash) کلمه عبور کاربر) به نظر شما بهتر است در کجا قرار گیرد.
1) به عنوان مثال اگر در Ef.....Service قرار گیرد خیلی جالب، زیبا و مربوط نیست.
2) میشه در یک بخش دیگر (مثلا مشترک) قرار گیره، که خوب بازم مسئله اینه که این متد همیشه به بخش کاربران سرویس میده و عملا نباید جدا باشه.
3) میشه از یک کلاس میانی انتزاعی استفاده کرد و متدهای مشترک در تمام انواع سرویس (EF، Fake، و یا ....) در دسترس باشه.
ممنون میشم که راهنمایی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۲۶ ۲۲:۳۲

من در تمام پروژه‌هام یک class library به نام MyProject.Common ایجاد می‌کنم برای قرار دادن توابعی که می‌تونه در پروژه‌های دیگر بدون وابستگی خاصی به پروژه جاری مورد استفاده قرار گیرد. دیدم جاهای دیگر اسمش رو گذاشتند Application framework من اسمش رو گذاشتم MyProject.Common. مثلا تابع SHA1 می‌تونه در چندین و چند پروژه بدون وابستگی خاصی استفاده شود و بین این‌ها مشترک است یا مثلا تابع فشرده سازی یک فایل هم به همین صورت و الی آخر.
سرویس‌های برنامه هم می‌تونند از این کتابخانه مشترک استفاده کنند و سرویس دهند.

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۳۹۱/۰۷/۲۷ ۰:۲۴

ممنون.

نویسنده: حسینی
تاریخ: ۱۳۹۱/۰۸/۰۴ ۱۱:۴۱

با سلام
از مطالب مفیدتون بینهایت سپاسگزارم.
من برای فهم بهتر این مطلب به نمونه ویندوزی اون احتیاج دارم.
با سپاس

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۴ ۱۲:۱

نمونه ویندوزی آن برنامه کنسولی است که به همراه کد این قسمت ارائه شده: (^)

نویسنده: حسینی
تاریخ: ۱۳۹۱/۰۸/۰۵ ۱۵:۱۶

با سلام
از پاسختون ممنون
در برنامه ای که یکی از دوستان زحمت کشیده بود؛ علاوه بر اینترفیس هایی که به ازای هر کلاس تعریف کرده بودند؛ یک اینترفیس هم تعریف کرده بودند و تمام متدها رو در اون تعریف کرده بودند. همه اینترفیس‌ها از اون ارث بری کرده بودند و تمام متدها رو به اون منتقل کرده بودند. داخل خودشون هیچ متدی باقی نمونده بود.
-1 این روش مورد تائید شما میباشد؟
-2 تعریف اینترفیسی که در روش بالا عرض کردم به شکل زیر است:
منظور از IDisposable چیست؟
-3 در صورت تائید روش ذکر شده, چه لزومی به تعریف مابقی اینترفیس‌ها است در صورتی که همه EFClassها میتوانند مستقیما از اون اینترفیس ارث بری کنند.

```
public interface IGenericService<T>: IDisposable where
 T : class
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۰۵ ۱۶:۱۳

به صورت خلاصه: شما می‌تونید یک سری متد عمومی رو در یک base class جنریک هم قرار بدید و از اون ارث بری کنید. به این ترتیب حجم کد نویسی کمتری خواهید داشت. اما این چند متد عمومی پاسخگوی نیاز یک برنامه واقعی نیستند. به همین جهت نیاز است مابقی اینترفیس‌ها و کلاس‌ها هم به صورت مجزا تعریف شوند.

نویسنده: فریدون غلامی
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۰:۱۶

سلام؛
من اگر بخوام Structure map را در پروژه Windows Application استفاده بکنم باید چکار کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۰:۲۵

- لطفا نظرات این مطلب را یکبار مطالعه کنید. پیشتر به این سؤال پاسخ داده شده:
خلاصه آن: قسمت «استفاده از الگوی واحد کار و کلاس‌های سرویس تهیه شده در یک برنامه کنسول ویندوزی» عنوان شده در مطلب فوق، یک برنامه ویندوزی است. سورس کامل این سری هم در دسترس است (لینک داده شده در پایان مطلب). شبیه به برنامه‌های وب که یک سری روال مانند شروع و پایان درخواست را دارند، در اینجا شروع یک فرم، پایان یک فرم، شروع و پایان مثلا یک کلیک را دارید.

نویسنده: فریدون غلامی
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۰:۳۱

بله شما درست می‌فرمایید ولی شما یه جا فرمودید که باید دستی کانکشن را از بین برد.
این چه طوری هستش و باید در کدام قسمت انجام داد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۰:۵۷

بحث ASP.NET متفاوت است با Windows Forms یا WPF. در برنامه‌های وب یک زیر ساخت آماده برای آغاز و پایان درخواست‌ها و مدیریت خودکار این مسایل وجود دارد. معادل آن مثلا در یک برنامه مبتنی بر MVVM، مدیریت طول عمر یک context در طول عمر ViewModel برنامه است. علت این مساله هم به stateless بودن برنامه‌های وب و state-full بودن برنامه‌های ویندوزی بر می‌گردد. در یک برنامه وب در پایان درخواست، تمام اشیاء یک فرم در سمت سرور تخریب می‌شوند. اما در یک برنامه ویندوزی تا زمانیکه یک فرم باز است، اشیاء آن تخریب نخواهند شد. بنابراین مدیریت context در برنامه‌های ویندوزی «دستی» است. در زمان شروع فرم context شروع خواهد شد، زمان تخریب/بستن آن، با بستن یا dispose یک context، خودبخود اتصالات هم قطع خواهند شد.

در متد Program.Main هم می‌تونید تنظیمات اولیه ObjectFactory رو قرار بدید (شبیه به Application_Start برنامه‌های وب).

بنابراین در برنامه‌های وب «context/session per http request» داریم؛ در برنامه‌های ویندوزی «context per operation or per form». یعنی می‌تونید بسته به معماری برنامه ویندوزی خود، context را در سطح یک فرم تعریف کنید و مدیریت؛ و یا در سطح یک عملیات کوتاه مانند یک کلیک. تمام مباحث uow.SaveChanges ، ObjectFactory.GetInstance و یا Dispose آن هم دستی است و زیر ساختی برای مدیریت خودکار آن‌ها همانند برنامه‌های مثلا ASP.NET MVC وجود ندارد. حداکثر اینکه یک سری base class را شبیه به مثال Web forms زده شده تهیه کنید، تا میزان کدهای تکراری را کاهش بدید.

نویسنده: فریدون غلامی
تاریخ: ۱۳۹۱/۱۰/۲۶ ۱۱:۲

واقعا ممنون از توضیحات جامعتون

نویسنده: فرهاد یزدان پناه
تاریخ: ۱۳۹۱/۱۰/۲۸ ۲۳:۵۰

وقت بخیر
یک سوال
اگر من به دلایلی لازم باشه از چند DbContext استفاده کنم (فرض کنید یکی برای اطلاعات اصلی و یکی برای فایل‌ها و ...) در این حالت به چه شکلی می‌توان از این الگو استفاده کرد؟
آیا لازم است که چند نوع UOW ایجاد شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۹ ۰:۱۳

بله. به ازای هر DbContext یک سری تنظیمات مجزا نیاز است. ردیابی تغییرات در یک context کار می‌کند. همچنین مباحث migrations هم به ازای یک context عمل خواهند کرد.

نویسنده: حسین
تاریخ: ۱۳۹۱/۱۰/۲۹ ۹:۲۷

سلام. من توی برنامه WPF MVVM ای که نوشتم یه Context توی هر ViewModel ایجاد می‌کنم میکنم و در پایان توی بستن ویومدل Context رو Dispose می‌کنم. قبلاً از using برای مدیریت اتصال به دیتابیس استفاده می‌کردم ولی وقتی از using استفاده می‌کنم دیگه تغییراتی که اعمال می‌کنم (حذف، ویرایش، افزودن) UI متوجه نمیشه.
میخواستم بدونم این مشکل با استفاده از الگوی Context Per Request حل میشه یا نه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۹ ۹:۴۸

خیر. این الگو خارج از توضیحات مطلب فوق در مورد «اهمیت بکارگیری الگوی Unit of work و به اشتراک گذاری آن در طی یک درخواست» کار دیگری را انجام نمی‌دهد.

نویسنده: فریدون غلامی
تاریخ: ۱۳۹۱/۱۱/۰۳ ۹:۹

سلام

structureMap را در پروژه ویندوزی Add کردم اما خطایی زیر را دارد:

Error 36 The type or namespace name 'StructureMap' could not be found

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۳ ۹:۱۹

قبلا بحث شده . لطفا نظرات را یکبار کامل مطالعه کنید.

نویسنده: علی
تاریخ: ۱۳۹۱/۱۱/۰۵ ۱۶:۳۰

سلام، در مثالی که برای MVC ذکر کردید، آیا امکان داره که تزریق وابستگی به این صورت زیر انجام بشه:

```
public static class DataFactory
{
    public static IUnitOfWork UnitOfWork
    {
        get { return ObjectFactory.GetInstance<IUnitOfWork>(); }
    }

    public static ICategoryService CategoryService
    {
        get { return ObjectFactory.GetInstance<IUnitOfWork>(); }
    }

    public static IProductService ProductService
    {
        get { return ObjectFactory.GetInstance<IUnitOfWork>(); }
    }
}

...

 public HomeController()
 {
 _productService = DataFactory.ProductService;
 _categoryService = DataFactory.CategoryService;
 _uow = DataFactory.UnitOfWork;
 }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۵ ۱۷:۲۳

خیر. برنامه‌های وب چند کاربری هستند. ProductService *استاتیک* یعنی به اشتراک گذاری یک وهله بین تمام کاربران سایت .

نویسنده: علی
تاریخ: ۱۳۹۱/۱۱/۰۵ ۲۱:۲۵

ممنون، ولی مگر

```
ObjectFactory.GetInstance<IProductService>()
```

هربار یک وهله از کلاس پیاده سازی شده‌ی آن (ProductService) ارائه نمی‌ده؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۶ ۹:۱۳

بله. مسایل همزمانی رو چطور مدیریت می‌کنید؟ زمانی که یک وهله استاتیک در اختیار برنامه قرار دادید آیا می‌تونید تضمین کنید که از بین مثلا 100 نفری که دارند از سایت استفاده می‌کنند، هیچکدام به صورت اتفاقی در آن واحد به همان وهله استاتیک دریافتی دسترسی پیدا نمی‌کنند؟ این وهله به اشتراک گذاشته شده می‌تونه اطلاعات مدیریتی باشه که نباید در اختیار یک کاربر با سطح دسترسی معمولی قرار بگیره.
ضمن اینکه در EF وهله DbContext به صورت Thread safe طراحی نشده و امکان به اشتراک گذاری آن بین چندین ترد وجود ندارد. به ازای هر ترد باید یک وهله جداگانه از آن تهیه شود تا شاهد تخریب اطلاعات نباشید.

نویسنده: سجاد
تاریخ: ۱۳۹۱/۱۱/۰۶ ۱۳:۲۷

با سلام؛
با این شرایط باید متد های add ,edit,delete ,.... رو در لایه سرویس برای همه‌ی کلاس‌ها بصورت جداگانه تعریف کرد امکانش وجود ندارد که لایه سرویس مون رو به صورت جنریک برای همه کلاس هامون داشته باشیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۶ ۱۳:۴۸

خیر. هستند یک سری الگوی مخزن عمومی به این شکل که در قسمت 11 سری EF نقد شدند و دارای مشکلات زیادی بوده که نیازی به تکرار آن در اینجا نیست. به علاوه دنیای واقعی با چند مورد متد ساده عمومی مدل نمی‌شود. عموما جمع چند عملیات هست که در قالب یک متد مشخص، خروجی یک سرویس را تشکیل می‌دهد. این عملیات هم می‌تواند مرتبط به چندین موجودیت باشد در آن واحد. تمام این موارد باید به صورت بسته بندی شده در قالب یک متد در اختیار لایه‌های دیگر قرار گیرد.

نویسنده: نوید
تاریخ: ۱۳۹۱/۱۲/۱۰ ۲۳:۴۱

با سلام
من در حین اجرای نمونه کدهای این مقاله در بخش MVC به خطای Value Cannot be null در کلاس

```
public class StructureMapControllerFactory : DefaultControllerFactory
    {
        protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
        {
            return ObjectFactory.GetInstance(controllerType) as Controller;
        }
    }
```

مواجه شدم که با اضافه کردن :

```
routes.IgnoreRoute("{*favicon}", new { favicon = @"(.*/)?favicon.ico(/.*)?" });
```

به متد Register Route برطرف شد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۱ ۰:۶

برای استفاده در شرایط واقعی:

```
public class StructureMapControllerFactory : DefaultControllerFactory
    {
        protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
        {
            if (controllerType == null)
                throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
            return ObjectFactory.GetInstance(controllerType) as Controller;
        }
    }
```

نویسنده: Masoud
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۱:۲۲

سلام

با توجه به گفته دوستمون (بنابر این باید برای هر ماژول dll ای تولید کرد که حاوی DomainClassها , ServiceLayerها ، Controllerها و DbContext مربوط به اون ماژول باشه)

اگر از این الگو برای طراحی نرم افزار استفاده شود خوب برای ارتباط بین ماژول‌ها که باید رفرنس همدیگر را در خود درج نمایند و در این صورت با circular dependency روبه رو میشویم

برای جلوگیری ای این خطا چه راه حلی پیشنهاد میدهید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۱:۳۷

- من هر نوع طراحی رو تائید نمی‌کنم. چرا یک برنامه باید چندین DbContext داشته باشد؟ نیازی نداره. چرا باید چندین ماژول کنترلر داشته باشه؟
- سؤال شما خارج از موضوع بحث است (در اینجا بحثی در مورد طراحی «افزونه پذیر» مطرح نشده). برای طراحی افزونه پذیر می‌تونید به مباحث زیر مراجعه کنید:
ابتدا فقط و فقط یک DbContext مرکزی را در کل برنامه تعریف کنید. بعد تنظیمات نگاشت‌ها را به صورت پویا یافته و به آن اضافه کنید. سپس موجودیت‌های مهیا را به صورت پویا یافته و به Context مرکزی اضافه نمائید.
+ در EF نمی‌تونید در عمل چندین DbContext داشته باشید مرتبط با یک دیتابیس. Change tracking در EF بر مبنای یک DbContext کار می‌کند. اگر قرار باشد چندین وهله از DbContextهای مختلف مثلا در طی یک درخواست وجود داشته باشند، یعنی چندین اتصال باز شده به دیتابیس و چندین تراکنش مجزا در حال انجام است (کل بحث جاری از ابتدا). به علاوه قابلیت کار کردن با چندین موجودیت را به صورت همزمان در طی یک تراکنش از دست می‌دهید.
- برای اینکه در حین کار با Structure Map خطای Circular dependency را مشاهده نکنید، نیاز است یک کتابخانه یا حتی یک کلاس واسط طراحی کنید تا مشترکات در آن قرار گیرند.

نویسنده: behrouz
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۸:۳۲

با سلام
به نظر می‌رسد با توجه به معماری که ارائه دادید منطق سیستم (BLL) و کدهای دستکاری داده‌ها یعنی کد هایی که اعمال CRUD را در دیتابیس انجام می‌دهند (DAL) را در یک لایه به نام لایه سرویس قرار دادید
به نظر شما برای خوانایی بیشتر بهتر نیست این دو از یکدیگر جدا شوند؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۸:۴۸

لایه‌ی Data Access در این معماری همان ORM مورد استفاده هست.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۳ ۲:۵۸

نظر شما در مورد این پیاده سازی الگوی کار چیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۳ ۹:۳۸

- ابتدای کار یک روش غلط رو شروع کرده، بعد اواسط کار اون رو نقد کرده و رد.
- چون از لایه سرویس استفاده نکرده کل منطق کار رو برده داخل کنترلرها.
- از تزریق وابستگی‌ها استفاده نکرده، بنابراین برخلاف شکلی که در ابتدای کار گذاشته، کنترلرهاش رو نمی‌تونه مستقل از یک سری کلاس کاملا مشخص، تست کنه.
- الگوی مخزنی که ارائه داده در این مثال ساده کار می‌کنه اما اگر قرار باشه با چند موجودیت کار کرد و نتیجه رو ترکیب، کارآیی خوبی نداره چون خیلی از قابلیت‌های ذاتی EF مثل کوئری‌های به تاخیر افتاده (deferred LINQ queries) در اینجا قابل پیاده سازی نیست. اگر هم بخوان این رو اضافه کنن باید به لایه مخزن خروجی IQueryable اضافه کنن که به یک طراحی نشتی دار خواهند رسید چون انتهای کار با خروجی IQueryable کاملا باز باقی می‌ماند (نمونه‌اش متد Get ایی است که طراحی کرده).
یا یکی دیگر از اهداف ظاهری لایه مخزن، امکان تعویض آن در صورت نیاز است و مثلا کوچ به یک ORM دیگر. دنیای واقعی: include ایی که اینجا تعریف شده فقط در EF وجود خارجی دارد یا یک سری از نکات دیگر بکار گرفته شده در این الگوی مخزن. (در قسمت 11 سری EF سایت بحث شده)
- در مثالی که زده باگ امنیتی وجود دارد. متد Updateاش به دلیل عدم استفاده از ViewModel آسیب پذیر هست. (در این مورد در سری MVC بحث شده)

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۴ ۸:۴۱

سلام
عالی مثل همیشه.
مهندس شما فرمودین:
الگوی مخزنی که ارائه داده در این مثال ساده کار می‌کنه اما اگر قرار باشه با چند موجودیت کار کرد و نتیجه رو ترکیب، کارآیی خوبی نداره چون خیلی از قابلیت‌های ذاتی EF مثل کوئری‌های به تاخیر افتاده (deferred LINQ queries) در اینجا قابل پیاده سازی نیست. اگر هم بخوان این رو اضافه کنن باید به لایه مخزن خروجی IQueryable اضافه کنن که به یک طراحی نشتی دار خواهند رسید چون انتهای کار با خروجی IQueryable کاملا باز باقی می‌ماند (نمونه‌اش متد Get ایی است که طراحی کرده).

البته (البته چندین جای دیگه هم گفتین) در مورد نشتی حافظه، کاربرد IQueryable پس توی کدام لایه از کار ما می‌تونه باشه با توجه به انعطاف پذیری که به کار ما میده؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۰:۵۷

نشتی طراحی مد نظر بوده؛ نه نشتی حافظه. نشتی طراحی به این معنا است که اگر متد شما خروجی IQueryable داشته باشه، در لایه‌های دیگر می‌شود کلا مقصود آن‌را تغییر شکل داد به فرم دیگری که اصلا شاید هدف اولیه این نبوده (چون IQueryable یک عبارت است و نه اجرای یک فرمان). به همین جهت باید در لایه سرویس و بدنه متدهای آن از IQueryable استفاده شود و نهایتا این متدها باید IList یا IEnumerable را بازگشت دهند. به این ترتیب حد و مرز یک لایه مشخص می‌شود.

نویسنده: صابر فتح الهی

تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۱:۸

آخه بعضا دیده شده (مثلا متدی مانند GetAll) کل رکوردهارو به صورت یکجا از بانک واکشی می‌کنه، اما ما می‌خواهیم قسمتی از اونها واکشی بشه مثلا 20 رکورد اول، با این تفاسیر در صورتی که خروجی از نوع IList (یا هر نوعی شبیه این) باشه اون وقت یکبار واکشی میشه کل رکوردها و بعد متد ما روی اون عمل انتخاب انجام میده.

1- ایا این باعث عدم کارایی نمیشه؟

2-خروجی نوع IQueryable کجا به کار ببریم؟

3-در کدام لایه تبدیل IQueryable به IList (یا انواع مشابه) باید انجام بشه.

معذرت دیگه زیادی دارم بحث کش میدم و می‌دونم اینجا جای پرسش و پاسخ نیست، بازم به بزرگواری و تجربتون من را ببخشید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۱:۱۸

لایه سرویس شما می‌تونه متد Paging دار هم داشته باشه. مثلا:

```
public IList<BlogPost> GetLatestBlogPosts(int pageNumber, int recordsPerPage = 4)
        {
            var skipRecords = pageNumber * recordsPerPage;
            return _blogPosts
                        .OrderByDescending(x => x.Id)
                        .Skip(skipRecords)
                        .Take(recordsPerPage)
                        .ToList();
        }
```

در این حالت در بدنه این متد لایه سرویس از IQueryable استفاده شده اما خروجی آن یک لیست مشخص است.

نویسنده: محسن
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۱:۱۹

به این مطلب مراجعه کنید.یه پیاده سازی کوچیکه که کلی از ابهامات را رفع میکنه.

نویسنده: عبدی
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۳:۸

سلام و تبریک سال نو

آقای نصیری نظر شما در مورد انتخاب بین IList یا IEnumerable را برای خروجی لایه سرویس می‌خوام بدونم. معمولا خودتون کدوم را استفاده و توصیه می‌فرمایید. ممنون میشم یه توصیه و توضیحی راجع به این مورد بدید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۳:۲۳

- این مساله در لابلای قسمت‌های مختلف سری EF در سایت بحث شده. قسمت 12 رو قسمت آخر تلقی کنید نه قسمت شروع.

- IList هم دقیقا از IEnumerable مشتق شده و یک سری قابلیت مانند افزودن آیتم به آن و همچنین دسترسی بر اساس ایندکس به آن اضافه شده.

- اگر در حین کار با خروجی لیستی یک متد، فراخوانی‌های بعدی نتیجه آن، فقط یکبار است، از IEnumerable استفاده کنید. اگر بیشتر از یکبار است از IList استفاده کنید. چون در EF هربار مراجعه به نتیجه یک IEnumerable مساوی است با واکشی دوباره اطلاعات از سرور. در حالت استفاده از IList کار یکبار انجام شده و تمام می‌شود.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۸ ۱۱:۴۹

اگر در یک سناریو نیاز باشد رشته‌ی اتصال در زمان اجرا عوض شود (مثلا در یک برنامه‌ی ویندوزی که طراحی به گونه ای هست که به ازای هر یک از یک موجودیت خاص یک دیتابیس ایجاد شود.) ، یک روش پاس دادن آن در زمان وهله سازی DbContext به سازنده هست.
در این روش که DbContext به صورت مستقیم در دسترس نیست ، چه روشی برای انجام این کار پیشنهاد می‌کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۸ ۱۲:۴۳

- DbContext در زمان شروع برنامه در دسترس است. مانند کلاس Sample07Context مثال این قسمت.
- اگر قرار است به ازای هر موجودیت یک دیتابیس داشته باشید یعنی چندین کلاس DbContext مجزا نیاز هست با تعاریف مختلف DbSetها در ابتدای کار. هر کدام را باید جداگانه در شروع برنامه با روش زیر مدیریت کرد. مثلا:

```
public partial class MyCtx1 : DbContext
{
    public MyCtx1(string connectionString) : base(connectionString) { }
}
```

و در این حالت بدیهی است هر کدام در Context مختلفی کار می‌کنند و بحث اعمال الگوی واحد کار در یک Context معنا دارد نه در چندین Context. در EF مباحث Tracking فقط در یک Context کار می‌کنند.
و اگر قرار است تمام موجودیت‌ها در یک کلاس DbContext باشند، خوب ... همگی نهایتا در زمان فعال سازی migrations باید در دیتابیس مشترکی قرار گیرند و گرنه برنامه آغاز نخواهد شد یا اینکه migrations را کلا از کار انداخت.
- به علاوه در همین مثال جاری در زمان تزریق وابستگی‌ها می‌شود نوشت:

```
x.For<IUnitOfWork>().HttpContextScoped().Use(() =>
                {
                    var ctx = new Sample07Context();
                    ctx.Database.Connection.ConnectionString = "...";
                    return ctx;
                });
```

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۸ ۱۴:۳۳

متشکرم.
منظور من از داشتن دیتایس به ازای هر موجودیت بد بیان شد.
یک DbContext داریم حاوی DbSet ها.
در یک برنامه‌ی ویندوزی کاربر می‌تواند دیتابیس‌های مختلف با نام‌های مختلف با یک طراحی داشته باشد.
در واقع به دلایلی مثل محدودیت SQL Server Express در ذخیره سازی (10 گیگ) طراحی به گونه ای انجام شده که برنامه بتواند به ازای هر پروژه یک دیتابیس داشته باشد.
برنامه به یک دیتابیس وصل است و زمانی که کاربر قصد ایجاد یک پروژه‌ی جدید دارد باید یک دیتابیس جدید ایجاد شود و Connection String عوض شود. و از این پس DbContext باید به دیتابیس جدید متصل شود.

```
public static void ChangeDatabase(string name)
{
    var sqlConnectionStringBuilder =
        new SqlConnectionStringBuilder(ConfigHelper.ActiveConnection);
    sqlConnectionStringBuilder["Database"] = name
    ConfigHelper.ActiveConnection = sqlConnectionStringBuilder.ToString();
    Database.DefaultConnectionFactory =
        new
System.Data.Entity.Infrastructure.SqlConnectionFactory(ConfigHelper.ActiveConnectionString());

    Database.SetInitializer(
        new MigrateDatabaseToLatestVersion<TestContext, MigrationConfiguration>());
    using (var context = new TestContext())
```

```
    {
        context.Database.Initialize(true);
    }
}
```

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۱۸ ۱۶:۲۴

با استفاده از این لینک و پاس دادن رشته‌ی اتصال هنگام تزریق وابستگی مشکلم حل شد.

نویسنده: حامد
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۰:۵۱

ممنون آقای نصیری
یه سوال مهم دارم
من اگه بخوام پروژه سیلورلایت انجام بدم با استفاره از معماری MVVM ، ترکیب اون با این مدل 5 لایه به چه شکل خواهد شد؟
میشه یه مثال خوب هم در این زمینه معرفی کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۸

- مباحث سمت سرور آن تفاوتی نمی‌کند؛ از این جهت که سیلورلایت نهایتا با استفاده از سرویس‌های سمت سرور WCF و یا WCF RIA Services قرار است به بانک اطلاعاتی دسترسی پیدا کند و برای نمونه امکان استفاده از EF Code first در WCF RIA Services مدتی هست که فراهم شده .
- ضمن اینکه سیلورلایت آینده مشخصی نداره ؛ بهتره روی ASP.MVC سرمایه گذاری کنید.

نویسنده: سجاد
تاریخ: ۱۳۹۲/۰۲/۰۸ ۲۳:۱۷

با سلام
وقتی از IOC استفاده می‌کنم کد زیر به درستی کار نمیکنه و خطای
No parameterless constructor defined for this object رو میده
با تشکر

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
            SelectMethod="GetAllProducts" TypeName="ServiceLayer.EfProductService">
        </asp:ObjectDataSource>
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۰۸ ۲۳:۲۳

مراجعه کنید به مطلب صفحه بندی اطلاعات در ListView .

نویسنده: debugger
تاریخ: ۱۳۹۲/۰۲/۲۲ ۲۰:۵۱

با سلام

"در پاسخ یکی از سوالهافرمودید با استفاده از IUnitOfWork اگر متد جاری شما از 10 کلاس هم استفاده کند، تماما با یک وهله از Context کار می‌کنند"

موقعی که از context یک وهله می‌گیریم و از Using استفاده می‌کنیم آیا به ازا هر کلاسی که در این scope هست connection جدیدی استفاده میشه ؟

آیا هنگام استفاده از Using موارد مربوط به همزمانی و transaction مدیریت نمیشه ؟

اگر این طور نیست مزیت روش بالا نسبت به استفاده از Using چیست ؟

ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۲۲ ۲۲:۱۰

مزیت این روش، استفاده از یک IoC Container برای مدیریت طول عمر DbContext در طول یک درخواست است.
در برنامه‌های وب، کار صرفا به یک کلیک ساده ختم نمی‌شود که در همان لحظه، یک Context آغاز و پایان یابد. در طی یک درخواست وب، قسمتی از صفحه لیست گروه‌ها، قسمتی دیگر لیست نویسندگان، قسمتی دیگر گزارش درصد استفاده از مرورگرها و قسمتی دیگر لیست آخرین مطالب را نمایش می‌دهد. تمام این‌ها هم در طی یک درخواست رخ می‌دهند و هرکدام، ماژول ماژول طراحی شده‌اند و از هم جدا.
اینجا است که ارزش استفاده از قابلیت‌های مدیریت طول عمر IoC containers برای به اشتراک گذاری یک DbContext در طی یک درخواست بهتر مشخص می‌شود. به این ترتیب می‌شود به سرباری کم و سرعتی بالا دست یافت چون مدام به ازای قسمت‌های مختلف برنامه Context ایجاد و تخریب نمی‌شود.

نویسنده: سجـــــاد ف
تاریخ: ۱۳۹۲/۰۲/۲۸ ۸:۱۴

با سلام
من از یک سری توابع جنریک استفاده میکنم که نیاز به دسترسی DbContext داره آیا تعریف این توابع در IUnitOfWork درسته؟
یک نمونه

```
public interface IUnitOfWork
    {
        IDbSet<TEntity> Set<TEntity>() where TEntity : class;
        int SaveChanges();
        void Update<TEntity>(TEntity entity) where TEntity : class;
    }


 public class MyContext : DbContext, IUnitOfWork
    {
        public void Update<TEntity>(TEntity entity) where TEntity : class
        {
            var fqen = GetEntityName<TEntity>();

            object originalItem;
            EntityKey key = (IObjectContextAdapter)this).ObjectContext.CreateEntityKey
                            (fqen, entity);
            if (((IObjectContextAdapter)this).ObjectContext.TryGetObjectByKey(key, out
                                            originalItem))
            {
               ((IObjectContextAdapter)this).ObjectContext.ApplyCurrentValues
                  (key.EntitySetName, entity);
            }
               ((IObjectContextAdapter)this).ObjectContext.ApplyCurrentValues
                        (key.EntitySetName, entity);
        }
```

```
        private string GetEntityName<TEntity>() where TEntity : class
        {
            return string.Format("{0}.{1}", ((IObjectContextAdapter)this).ObjectContext.
            DefaultContainerName, _pluralizer.Pluralize(typeof(TEntity).Name));
        }

        #region IUnitOfWork Members
        public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
        {
            return base.Set<TEntity>();
        }
        #endregion
    }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۲۸ ۹:۰

نیازی به این پیچ و تاب‌ها در EF Code first نیست. تابع استاندارد Find ایی که در آن اضافه شده همین کار ابتدا مراجعه به کش و بعد مراجعه به دیتابیس رو انجام می‌ده.

نویسنده: داود
تاریخ: ۱۳۹۲/۰۲/۲۸ ۹:۱۳

با سلام
ما چرا در mvc موجودیت هامون رو تو Model تعریف نکردیم و تو Domain Classes تعریف کردیم حالا اگر نیاز به اعتبارسنجی بود آیا باید تو همون DomainClasses اینکار رو با Metadata که فرموده بودید انجام بدیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۲۸ ۹:۱۸

نیاز است با یک سری پیشنیاز مانند ViewModel و همچنین « مقابله با مشکل امنیتی Mass Assignment در حین کار با Model binders » آشنا باشید تا علت جدا سازی این موارد از هم مشخص بشه.

نویسنده: داود
تاریخ: ۱۳۹۲/۰۲/۲۸ ۱۰:۴۲

با سلام و تشکر از پاسختون
من با ViewModel آشنایی دارم و مطالب Mvc شما رو کامل خوندم آیا درست متوجه شدم جهت اعتبار سنجی باید یک کلاس در ViewModel بسازم و اعتبارسنجی رو انجام بدیم و بعد کلاس ViewModel رو تبدیل به کلاس اصلی کرده و برای انجام عملیات مربوطه به Service بفرستیم؟ یا نه باید یک کلاس مشابه کلاس موجود در DomainClasses در Model بسازم و اعتبارسنجی رو اونجا انجام بدم و بعد ارسال به Service
یا باید به صورت زیر توی همون domainClasses انجام بدم

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer
{
    class CustomerMetadata
     {
     }
}
public partial class Customer : IValidatableObject
{
```

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۲/۲۸ ۱۰:۵۹

ViewModel متناظر است با اشیاء یک View که الزاما تطابق یک به یکی با Domain Model و موجودیت‌های بانک اطلاعاتی ندارند. مثلا یک صفحه تعویض پسورد هست و فقط یک فیلد پسورد داره. اینجا در معرض دید قرار دادن کل موجودیت کاربر در یک برنامه وب MVC اشتباه است چون به سادگی مورد حمله واقع خواهد شد. خلاصه هر دو مورد ViewModel و Domain model نیاز به اعتبارسنجی دارند؛ به هر روشی که صلاح می‌دونید.
نهایتا اطلاعات ViewModel در حالت Post، به اطلاعات Model انتساب داده میشه. یا دستی و یا مثلا توسط AutoMapper ؛ در این حالت هم هر طور که راحت هستید عمل کنید. قانون یا روش بهتری برای این نوع انتساب‌ها وجود نداره.

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۳/۰۱ ۱۵:۳۰

سلام
نمونه ای از پیاده سازی روش بالا در WPF با MVVM سراغ دارید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۱ ۱۶:۲۶

پیشنیاز تئوری قسمت 12، دوره‌ای است به نام « بررسی مفاهیم معکوس سازی وابستگی‌ها و ابزارهای مرتبط با آن » در حدود 11 قسمت.
بعد از مطالعه آن، خودتان به سادگی می‌توانید این مباحث را در الگوهای مختلف پیاده سازی کنید.

نویسنده: m.d
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۲:۳۵

با تشکر از معرفی دوره آموزشی شما
متاسفانه هنوز متوجه نشدم چگونه uow را در ViewModelهای برنامه استفاده کنم
ترکیب StructureMap , MVVM Light ، EF Code First چگونه پیاده سازی میشود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۲:۴۲

دوره جدیدی به سایت اضافه شده تحت عنوان « طراحی یک فریم ورک برای کار با WPF و EF Code First توسط الگوی MVVM ».
دسترسی به آن فقط برای نویسندگان سایت با حداقل یک مطلب ارسالی در طی یک ماه قبل است. البته همه‌ی کاربران عضو ، می‌توانند مشارکت کنند و در سایت مطلب ارسال کنند. از این لحاظ محدودیتی وجود ندارد.

نویسنده: m.d
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۴:۲۳

یعنی برای استفاده از این دوره می‌بایست یک مطلب به سایت ارسال کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۴:۲۷

بله. البته پس *از* تائید محتوای آن و انتشار در صفحه اول سایت، بلافاصله دسترسی شما باز خواهد شد.

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۳/۲۹ ۰:۱۵

با سلام، من کد شما را به صورت زیر تغییر دادم
ابتدا یک اینترفیس به صورت زیر ایجاد کردم

```
namespace Service.Interfaces
{
    public interface IGenericService<T>
    {
        void AddOrUpdate(T entity);
        void Delete(T entity);
        T Find(Func<T, bool> predicate);
        T GetLast(Func<T, bool> predicate);
        IList<T> GetAll();
        IList<T> GetAll(Func<T, bool> predicate);
        IList<T> GetAll(Expression<Func<T, object>> orderby);
        IList<T> GetAll(Func<T, bool> predicate, Expression<Func<T, object>> orderby);

        Task<List<T>> GetAllAsync();
        Task<List<T>> GetAllAsync(Func<T, bool> predicate);
        Task<List<T>> GetAllAsync(Expression<Func<T, object>> orderby);
        Task<List<T>> GetAllAsync(Func<T, bool> predicate, Expression<Func<T, object>> orderby);

        int Count();
        int Count(Func<T, bool> predicate);
    }
}
```

و تمام اینترفیس‌های دیگر از این به صورت زیر به ارث برده شده اند

```
public interface IBookGroupService:IGenericService<BookGroup>
    {
    }
```

و در قسمت Servise یک کلاس ایجاد کردم که اینترفیس IGenericService را پیاده سازی می‌کند که کدهای آن به صورت زیر است

```
public class EFGenericService<TEntity> : IGenericService<TEntity>
         where TEntity : class
    {
        protected IUnitOfWork _uow;
        protected IDbSet<TEntity> _tEntities;

        public EFGenericService(IUnitOfWork uow)
        {
            _uow = uow;
            _tEntities = _uow.Set<TEntity>();
        }


        public void AddOrUpdate(TEntity entity)
        {
             _tEntities.AddOrUpdate(entity);
        }

        public virtual void Delete(TEntity entity)
        {
            _tEntities.Remove(entity);
        }

        public virtual TEntity Find(Func<TEntity, bool> predicate)
        {
            return _tEntities.Where(predicate).FirstOrDefault();
        }

        public virtual TEntity GetLast(Func<TEntity, bool> predicate)
        {
            return _tEntities.Where(predicate).Last();
        }


        public virtual IList<TEntity> GetAll()
        {
            return _tEntities.ToList();
```

```
        }

        public virtual IList<TEntity> GetAll(Func<TEntity, bool> predicate)
        {
            return _tEntities.Where(predicate).ToList();
        }

        public virtual IList<TEntity> GetAll(Expression<Func<TEntity, object>> @orderby)
        {
            return _tEntities.OrderBy(@orderby).ToList();
        }

        public virtual IList<TEntity> GetAll(Func<TEntity, bool> predicate, Expression<Func<TEntity,
object>> @orderby)
        {
            return _tEntities.OrderBy(@orderby).Where(predicate).ToList();
        }

        public async Task<List<TEntity>> GetAllAsync()
        {
           return await Task.Run(() => _tEntities.ToList());
        }

        public async Task<List<TEntity>> GetAllAsync(Func<TEntity, bool> predicate)
        {
          return await Task.Run(() => _tEntities.Where(predicate).ToList());
        }

        public async Task<List<TEntity>> GetAllAsync(Expression<Func<TEntity, object>> @orderby)
        {
           return await Task.Run(() => _tEntities.OrderBy(@orderby).ToList());
        }

        public async Task<List<TEntity>> GetAllAsync(Func<TEntity, bool> predicate,
Expression<Func<TEntity, object>> @orderby)
        {
           return await Task.Run(()=> _tEntities.OrderBy(@orderby).Where(predicate).ToList());
        }

        public virtual int Count()
        {
            return _tEntities.Count();
        }

        public virtual int Count(Func<TEntity, bool> predicate)
        {
            return _tEntities.Count(predicate);
        }
    }
```

و بقیه کلاس‌ها از کلاس بالا به ارث می‌برند.

```
public class EFBorrowService:EFGenericService<Borrow>,IBorrowService
    {
        public EFBorrowService(IUnitOfWork uow) : base(uow)
        {
        }
    }
```

حال سوال من اینه که این پیاده سازی از لحاظ پیاده سازی مشکلی ندارد؟ و می‌توانم در پروژه هام از این روش استفاده کنم یا خیر؟

ممنونم

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۲/۰۴/۰۱ ۹:۲۱

با سلام.
من کل پوشه Controllers را درون یک اسمبلی جداگانه قرار دادم. ولی هنگام اجرای برنامه خطای زیر رخ میدهد.

```
public class StructureMapControllerFactory : DefaultControllerFactory
    {
        protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
        {
            if (controllerType == null)
            {
                throw new InvalidOperationException(string.Format("Page not found: {0}",
requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
            }
            return ObjectFactory.GetInstance(controllerType) as Controller;
        }
    }

StructureMap Exception Code:  202
No Default Instance defined for PluginFamily MyProject.Controllers.HomeController,
MyProject.Controllers, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
```

با تشکر.

نویسنده: اکبر بابامرادی
تاریخ: ۱۳۹۲/۰۴/۲۰ ۰:۴

با سلام و خسته نباشید

آقای نصیری با توجه به توضیحات شما اگه نیاز به متدهای (مثلا: where, Skip,take,...) داشته باشیم آیا این متدها رو هم باید پیاده سازی کنیم؟

یا اینکه لیستی از مقادیر موجود رو بخونیم و بعد با استفاده از linq کوری مورد نظر رو استخراج کنیم!؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۰ ۰:۱۳

- منطق تجاری مورد نظر رو باید به صورت یک متد با حد و مرز مشخص، کپسوله و داخل این متد از Skip و Take و سایر امکانات LINQ استفاده کنید.
- ضمنا نباید لیست تهیه کنید و بعد روی آن Take انجام دهید؛ چون کارآیی پایینی داره:
« تفاوت بین IQueryable و IEnumerable در حین کار با ORMs »

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۲/۰۴/۲۷ ۰:۲

با سلام
یکی از دوستان در این قسمت نظری به شرح {در حین اجرای نمونه کدهای این مقاله در بخش MVC به خطای Value Cannot be null مواجه شدم } بیان کردند و راه حل نیز ارائه کردند و شما نیز پاسخی برای نظر ایشان ذکر کردید میشه در مورد وقوع این خطا بیشتر توضیح بدید ؟

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۷ ۰:۳۸

علت وقوع خطا، در استثنای صادره ذکر شده: Page not found . یعنی کاربر یا حتی مرورگر درخواست آدرسی رو کرده که در سایت شما موجود نیست (مثلا کروم اولین کاری که انجام می‌ده، وجود فایل استاندارد آیکن سایت رو به صورت خودکار بررسی می‌کنه) . بنابراین امکان وهله سازی کنترلر معادل آن صفحه یا آدرس یافت نشده، وجود ندارد.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۲/۰۴/۲۷ ۴:۲۴

این یعنی به ازاء تمام درخواستهایی که سمت سرور ارسال میشه context تشکیل میشه (نیاز به context باشه یا نه ، مثلا یه تصویر قرار هست در صفحه نمایش داده شود و نیاز به کنترلر هم نداره. خطای گزارش شده نمایش آیکون بود) ؟
آیا از لحاظ Performance مشکلی نداره ؟

ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۷ ۸:۵

- Context زمانی تشکیل خواهد شد که کار وهله سازی کنترلر متناظر و موجودی با موفقیت انجام شده باشد.
+ زمانیکه runAllManagedModulesForAllRequests در وب کانفیگ به true تنظیم شده تمام درخواست‌ها به موتور ASP.NET نگاشت می‌شوند. می‌شود این وضعیت را کنترل کرد؛ مراجعه کنید به قسمت « تنظیمات ثانویه پس از فعال سازی RouteExistingFiles »
در کل تهیه یک برنامه ASP.NET MVC نیاز به رعایت یک سری موارد دارد که پیشتر در این سایت بحث شده: « چک لیست تهیه یک برنامه ASP.NET MVC ». یکی از نکات آن هم «پوشه‌های Content و Scripts از سیستم مسیریابی تعریف شده در Global.asax خارج شوند» است.

نویسنده: hossein101211
تاریخ: ۱۳۹۲/۰۵/۰۵ ۲۱:۸

با سلام و خسته نباشید
من یه کلاس استاتیک دارم میخواستم ببینم لزومی داره داخل متدهای استاتیک هم الگوی unit of work استفاده بشه یا نه؟
چون به نظرم متدی که استاتیک تعریف میشه یه جورایی الگوهایی که باید رعایت بشه(مخصوصا unit of work) رو دور میزنه!
نمیدونم تا چه حد درست فکر میکنم یانه؟

```
private  static  readonly ICatHotellService _catHotellService;
  private static readonly ICatTourismService _catTourismService;
  private static readonly ICatTourService _catTourService;
  private static readonly IUnitOfWork _uow;
  public DropDownList(ICatHotellService CatHotellService, IUnitOfWork ouw, ICatTourService
CatTourService, ICatTourismService CatTourismService)
  {
_uow=ouw;
_catHotellService = CatHotellService;
_catTourismService = CatTourismService;
_catTourService = CatTourService;
  }
```

ولی دیگه نمیشه داخل سازنده DropDownList اونا رو بهم نسبت داد
لطفا کمی راهنمایی کنید با تشکر

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۰۵/۰۵ ۲۱:۴۰

اگر برنامه وب است، به هیچ عنوان نباید از سرویس‌هایی که به صورت یک فیلد استاتیک تعریف شدند استفاده کنید:
بررسی واژه کلیدی static
متغیرهای استاتیک و برنامه‌های ASP.NET

- اگر برنامه دسکتاپ است و نیاز دارید که اطلاعات یک سرویس خاص را در طول عمر برنامه زنده نگه دارید، برای نمونه در StructureMap حالت طول عمر Singleton هم وجود دارد برای مدیریت این نوع سرویس‌ها و نیازی نیست باز هم متغیر استاتیک تعریف کنید. (یک نمونه آن در دوره « طراحی یک فریم ورک برای کار با WPF و EF Code First توسط الگوی MVVM » بحث شده به همراه مثال کاربردی)
- ضمنا زنده نگه داشتن اطلاعات یک سرویس در طول عمر یک برنامه، باید با آگاهی کامل صورت گیرد. در اینجا و در حالت استفاده از EF، به این ترتیب Context ایجاد شده Dispose نخواهد شد و همین مساله مشکلات زیادی مانند خطاهای ثبت اطلاعات جدیدی که پیشتر در صفحه‌ای دیگر به Context وارد شدن را سبب می‌شود. همچنین در محیط‌های چندکاربری مانند وب، یک Context به اشتراک گذاشته بین تمام کاربران (مفهوم متغیرهای استاتیک)، thread safe نیست و مشکلات تداخل اطلاعات و یا حتی تخریب آن‌ها را شاهد خواهید بود.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۲/۰۵/۱۳ ۱۵:۹

سلام
در مثال مربوط به MVC برای controllerها از یک کلاس به عنوان base استفاده کردم و متد ExceuteCore رو جهت تنظیمات Globalization تحریف کردم . در این حالت متد ExceuteCore اجرا نشد .چند تا سوال داشتم ممنون میشم راهنمایی کنید
1- آیا تحریف متد ExcuteCore برای تنظیمات Globalization جای مناسبی هست ؟
2- مشکل مربوط به اجرا نشدن ExecuteCore رو با توجه به اینکه StructureMap وهله جدیدی از controller ایجاد میکنه رو چه طور میشه برطرف کرد .
ممنون

نویسنده: امیر
تاریخ: ۱۳۹۲/۰۵/۱۴ ۱۹:۱۱

سلام. اگر بخواهیم با استفاده از الگوی واحد کار یک کوئری دستی ایجاد کنیم روش کار به چه صورتی خواهد بود؟ یعنی من به صورت عادی از کد زیر استفاده میکنم:

```
using (var context = new MyContext())
{
context.Database.SqlQuery......
}
```

حالا با استفاده از UnitOfWork چگونه باید این کار رو انجام داد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۵ ۹:۵۸

شما محدود نیستید به چند متد اولیه‌ای که در اینترفیس Uow ذکر شده. یک متد با امضای اجرای SQL به آن اضافه کنید و پیاده سازی آن‌را در کلاس Context خود که از اینترفیس Uow مشتق می‌شود، انجام دهید (مانند this.Database الی آخر). بعد کلاس‌های استفاده کننده از Uow به آن دسترسی خواهند داشت.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۵ ۱۳:۲۸

- خیر. ASP.NET MVC یک فریم ورک AOP سر خود است. این مسایل رو باید با فیلترها پیاده سازی کنید.

- StructureMap وهله جدیدی را ایجاد می‌کند، اما ... کار استفاده (یا عدم استفاده) از آن به عهده ASP.NET MVC است و StructureMap دخالتی در آن ندارد.
- این مورد (عدم فراخوانی ExecuteCore تحریف شده) تغییری است که در MVC4 اعمال شده

```
public class MyBaseController : Controller
    {
        /// <summary>
        /// from
http://forums.asp.net/t/1776480.aspx/1?ExecuteCore+in+base+class+not+fired+in+MVC+4+beta
        /// </summary>
        protected override bool DisableAsyncSupport
        {
            get { return true; }
        }

        protected override void ExecuteCore()
        {
            base.ExecuteCore();
        }
    }
```

باید DisableAsyncSupport را اضافه کنید.

نویسنده: محمد تیموری بادله
تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۹:۴۰

با سلام و تشکر از مطالب مفید تون

اگر بخواهیم تحت هر شرایطی به کلاس DbContext دسترسی داشته باشیم به چه شکلی خواهد بود؟
در این حالت اگر بخواهیم یکی از Entityها را ویرایش کنیم به چه صورتی هست؟من هر کاری م کنم با ارور برخورد می‌کنم و فکری غیر از اینکه DbContext را داخل uow بیارم به ذهنم نمی‌رسه!

به چه شکلی می‌شود این مشکل را حل کرد؟

با احترام.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۲۱ ۲۱:۵۳

سیستم مدیریت محتوای IRIS از الگوی واحد کار استفاده کرده. از کدهاش برای انجام یک پروژه واقعی ایده بگیرید. برای برنامه‌های دسکتاپ هم دوره طراحی یک فریم ورک برای کار با WPF و EF Code First توسط الگوی MVVM از الگوی واحد کار استفاده می‌کنه.

نویسنده: رضا گرمارودی
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۶:۱۲

با خوندن مطالب فوق این طور بر میاد که در Ef یک Context به صورت Singleton ایجاد بشه تا هم بهینه باشه و هم مباحث مدیریت Transactionها و غیره به راحتی مدیریت بشه.

اما در اینجا StackOverflow در این خصوص خوندم که بهتره برای هر thread یک Context مجزا ایجاد کرد و سیستم Pools کانکشن تکراری و ارتباط متعدد را چک می‌کند.

بنده اشتباه برداشت کردم و یا سیستم Pool استفاده دیگره ای دارد .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۸:۴۶

- یک Context در EF باید طول عمر کوتاهی داشته باشد. سینگلتون تعریف کردن آن یعنی زنده نگه داشتن آن در طول عمر برنامه. در یک برنامه وب به این ترتیب اگر جایی کاربری به مشکلی برخورد، چون یک Context در این حالت بیشتر وجود نخواهد داشت، تمام خطاهای آن کاربر به سایر کاربران نیز منعکس می‌شود. همچنین Context به صورت thread safe طراحی نشده و به این ترتیب به مشکلات تخریب اطلاعات در برنامه‌های چند کاربره نیز برخواهید خورد.
- در مطلب فوق به ازای هر درخواست یک Context ایجاد می‌شود (مقدمه بحث). Context در طول عمر یک درخواست کاربری خاص، زنده نگه داشته شده و بعد در پایان کار آن درخواست Dispose می‌شود. نه فقط بحث مدیریت اتصالات در اینجا مطرح است، بحث مدیریت یک تراکنش واحد در طول عمر یک درخواست نیز باید درنظر گرفته شود.

چند پیشنهاد:
- یکبار مطلب جاری را مطالعه کنید.
- یکبار وقت بگذارید نظرات آن‌را هم بررسی کنید. در مورد سینگلتون یا حتی Contextهای استاتیک و امثال آن قبلا بحث شده.
- یکبار دوره پیشنیاز این مطلب را مطالعه کنید: « بررسی مفاهیم معکوس سازی وابستگی‌ها و ابزارهای مرتبط با آن »
- یکبار دوره خاص برنامه‌های دسکتاپ طراحی شده با این الگو را هم مطالعه کنید: « طراحی یک فریم ورک برای کار با WPF و EF Code First توسط الگوی MVVM »
- نگاهی هم به یک پروژه کامل ASP.NET MVC که با درنظر گرفتن الگوی مطرح شده در این بحث تهیه شده، داشته باشید: « سیستم مدیریت محتوای IRIS »

این مبحث باز شده‌اش بالای 20 قسمت است.

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۷:۱۲

یکی از اشکالات repository این بود که در عمل و دنیای واقعی، قابلیتی برای تعویض ORM ایجاد نمی‌کرد و گفتید در صورتی می‌توان این کار را کرد که دست و پای ORM را ببندیم و از مشترکات استفاده کنیم. اما در الگویی که معرفی کردید یعنی IUnitOfWork هم ظاهرا دست و پایمان برای پیاده سازی متدهای خاص بسته است مانند متد ساده زیر

```
public void ChangeState<T>(T entity, EntityState state) where T : class
{
     Entry<T>(entity).State = state;
}
```

یا متدی که در بالا ذکر شد یعنی:

```
DbEntityEntry<TEntity> Entry<TEntity>(TEntity entity) where TEntity : class;
```

اینطور نیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۸:۲۸

برای «پیاده سازی متدهای خاص» متد اضافه کن؛ اینترفیس رو تغییر بده. در *مثالی* که زده شده، من دو تا متد تعریف کردم. شما بسطش بده. مثلا:

```
public interface IUnitOfWork
{
   //...
```

```
    DbEntityEntry<TEntity> Entry<TEntity>(TEntity entity) where TEntity : class;
}
```

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۴:۲۰

در مدلی که ارایه کردید هم نمی‌توان متدهای خاص که از کلاسهای خاصی استفاده کرده اند را بکار برد مثل DbEntityEntry یا EntityState

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱۵:۳۱

عرض کردم، اینترفیس را بسط دهید؛ مثلا مانند کدهای زیر. DbEntityEntry را داخل یک متد مانند MarkAsChanged هم می‌شود محصور کرد با خروجی void. به عبارتی نحوه کار با base.Entry را بهتر می‌شود از دید مصرف کننده مخفی کرد تا حتما او نیازی نداشته باشد خودش مستقیما base.Entry(entity).State = EntityState.Modified را در کدهای نهایی مورد استفاده قرار دهد. فقط کافی باشد تا متد عمومی MarkAsChanged را که در پشت صحنه از base.Entry(entity).State استفاده می‌کند، بکارگیرد.

```
namespace EF_Sample07.DataLayer.Context
{
    public interface IUnitOfWork
    {
        IDbSet<TEntity> Set<TEntity>() where TEntity : class;
        int SaveAllChanges();
        void MarkAsChanged<TEntity>(TEntity entity) where TEntity : class;
    }
}

namespace EF_Sample07.DataLayer.Context
{
    public class Sample07Context : DbContext, IUnitOfWork
    {
        public DbSet<Category> Categories { set; get; }
        public DbSet<Product> Products { set; get; }

        public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
        {
            return base.Set<TEntity>();
        }

        public int SaveAllChanges()
        {
            return base.SaveChanges();
        }

        public void MarkAsChanged<TEntity>(TEntity entity) where TEntity : class
        {
            base.Entry<TEntity>(entity).State = EntityState.Modified;
        }
    }
}
```

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۴:۴۴

آیا امکان ثبت یک گراف از اشیائ مرتبط، بصورت یکجا در طی یک درخواست وجود دارد؟(در یک برنامه multi-tier وب یا ویندوزی) منظورم این است که فرضا مدلی داریم به شکل زیر:

آیا امکان این وجود دارد که کاربر یک سفارش را بصورت زیر ویرایش کند:

ویرایش تاریخ سفارش در کلاس Order

اضافه نمودن یک OrderLine جدید به Order
حذف یکی از OrderLineهای موجود
ویرایش Product مربوط به یک OrderLine

و سپس دکمه Save را بزند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۵:۶

اخیرا کتابی منتشر شده به نام Entity Framework 6 Recipes . این کتاب فوق العاده است. جزو معدود کتاب‌های چند سال اخیر است که ارزش یکبار خواندن را دارد. فصل 9 آن دقیقا مرتبط است به موضوع «Using the Entity Framework in N-Tier Applications».

نویسنده: حسین
تاریخ: ۱۳۹۲/۱۰/۰۵ ۲۰:۳۳

سلام
من دقیقا طبق همین الگویی که عرض کردید درحال نوشتن پروژه هستم ، در بعضی از نقاط پروژه باید از jquery ajax استفاده کنم ، به عنوان مثال ورود کاربر ، حالا مسئله ای که هست من باید در webmethod مربوطه یک method static رو صدا بزنم ، توی این الگو چطور می‌تونم مثلا ثبت یک رکورد رو بصورت یک method static انجام بدم ؟

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۵ ۲۱:۱

- باید از الگوی Service locator استفاده کنید در این موارد خاص فناوری‌های قدیمی که برای تزریق وابستگی‌ها طراحی نشده‌اند. پیشنیاز این بحث دوره « بررسی مفاهیم معکوس سازی وابستگی‌ها و ابزارهای مرتبط با آن » است.
- ضمن اینکه الان با بودن ASP.NET Web API که هم با وب فرم‌ها سازگار است و هم با MVC، دلیلی برای استفاده از وب متدهای استاتیک عهد عتیق وجود ندارد. ASP.NET Web API طوری طراحی شده تا تزریق وابستگی‌ها در آن ممکن و آزمون پذیری آن بالا باشد.

نویسنده: مسعود2
تاریخ: ۱۳۹۲/۱۰/۰۶ ۲۲:۹

فرض کنید که بخواهیم در این مثال این کارها رو انجام بدیم:
در یک صفحه لیست کالاها و دسته بندی اونها رو نشون بدیم.

کاربر قادر باشه در همون صفحه مشخصات یک کالا شامل گروه کالا یا نام کالا رو ویرایش کنه.

در این حالت برای ویرایش آیا بایستی از همون وهله DbContext که جهت گرفتن لیست کالاها استفاده شده، استفاده بشه؟ یا برای ویرایش بایستی یک وهله جدید DbContext ساخته بشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۶ ۲۳:۴۰

برنامه وب هست یا برنامه ویندوز؟ اگر برنامه وب هست که پس از پایان نمایش صفحه Context شما Dispose شده در سرور. اگر برنامه ویندوزی هست، بله می‌توانید از همان وهله استفاده کنید. چون تا زمانیکه فرم باز است، آن وهله هم می‌تواند باز باشد، یا زنده نگه داشته شود. نمونه آن در مثال « طراحی یک فریم ورک برای کار با WPF و EF Code First توسط الگوی MVVM » مطرح شده. یکبار مثال آن‌را اجرا کنید. «لطفا»

نویسنده: محمدرضا برنتی
تاریخ: ۱۳۹۲/۱۰/۰۷ ۲۳:۱۵

```
private static void initStructureMap()
        {
            ObjectFactory.Initialize(x =>
            {
                x.For<IUnitOfWork>().HttpContextScoped().Use(() => new Sample07Context());
                x.ForRequestedType<ICategoryService>().TheDefaultIsConcreteType<EfCategoryService>();
                x.ForRequestedType<IProductService>().TheDefaultIsConcreteType<EfProductService>();
            });
            //Set current Controller factory as StructureMapControllerFactory
            ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
        }
```

دقیقا در هر خط چه کاری انجام می‌دهند ؟ امکان داره یه مثال حقیقی بزنید ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۰۷ ۲۳:۲۳

در دوره « بررسی مفاهیم معکوس سازی وابستگی‌ها و ابزارهای مرتبط با آن » مفصل بحث شده و جزئیات آن کاملا بررسی شده‌اند.

نویسنده: vici
تاریخ: ۱۳۹۲/۱۱/۰۴ ۱۵:۲۴

ممنون؛ در برنامه ای که از linq to sql استفاده شده. بخواهیم همین روش (Unit of work) رو پیاده کنیم چه کاری باید انجام بشه؟ با تشکر

نویسنده: vici
تاریخ: ۱۳۹۲/۱۱/۰۵ ۲۲:۴۱

سلام؛ من از روش شما استفاده کردم در این خط کد

```
x.For<IUnitOfWork>().CacheBy(InstanceScope.Hybrid).Use<Context>();
```

یه هشدار میده که به این صورت

```
Warning 1 '....CacheBy(StructureMap.InstanceScope)' is obsolete:
```

این جوری که من متوجه شدم میگه این روش absolute شده درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۵ ۲۲:۵۰

- ویژگی absolute یعنی متد CacheBy در نگارش بعدی احتمالا حذف خواهد شد؛ نه اینکه در نگارش فعلی قابل استفاده نیست.
- مبحث تزریق وابستگی‌ها و به روز شده‌ی این مطالب در دوره‌ای به همین نام در سایت ارائه شده

```
x.For<IUsersService>().HybridHttpOrThreadLocalScoped().Use<UsersService>();
```

نویسنده: ناظم
تاریخ: ۱۳۹۲/۱۱/۱۲ ۲۳:۷

سلام؛ در اینترفیس IUnitofWork ما 2 تا متد زیر روداریم

```
IDbSet<TEntity> Set<TEntity>() where TEntity : class;
int SaveChanges();
```

که هر 2 تا تو dbcontext پیاده سازی شدن، ولی تو context خودمون متد Set دوباره پیاده سازی شده
که همون متد dbcontext رو اجرا میکنه، چه نیازی به این کار بود؟ و با متد savechanges چرا اینکارو نکردیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۲ ۲۳:۱۳

جهت نیازهای آموزشی مفاهیم ارث بری در کلاس‌ها؛ مثلا اگر متدی هم نام با یکی از متدهای کلاس پایه DbContext را خواستید بازنویسی کنید، چکار باید کرد و امثال آن. این بازنویسی‌ها ممکن است به همراه یک سری کد اضافی از طرف شما هم باشند. مثلا متد Save کلاس پایه را بازنویسی کنید و قبل از آن خودتان اعتبارسنجی خاصی را اضافه کنید. ضمنا ضرورتی هم در بکارگیری متدهای هم نام با کلاس پایه، نیست. الان حداقل مشاهده کرده‌اید که با کدام متدهای کلاس پایه باید کار کرد و به چه صورتی.

نویسنده: ح مراداف
تاریخ: ۱۳۹۲/۱۱/۲۵ ۱۵:۷

سلام، با تشکر از مقاله عالیتون. بنده 2 تا علامت سوال توی ذهنم پدید اومده که اگر کمکم کنین ، بسیار ممنون میشم :
1- در تمامی کدها خبری از try catch نیست ، آیا نیاز نیست ؟ یعنی اگر جایی مشکلی پیش بیاد کاربر صفحه ارور معروف asp رو

نمی‌بینه ؟
اگر نیاز هست ، آیا در داخل کنترلر باید استفاده شود ؟
{تا جایی که من می‌دونم گویا در لایه سرویس باید اطلاعات رو درست در نظر بگیریم و بررسی اطلاعات ورودی و مدیریت خطا باید در کنترلر باشه}

2- در بخش زیر ، برای بنده که از First Database استفاده می‌کنم ، باید چکار کنم ؟

```
public class Sample07Context : DbContext, IUnitOfWork
```

آیا باید یک کلاس دیگه بسازم و از کانتکس اصلیم ارث بدم ؟
با تشکر از وقتتون.
یا حق

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۵ ۱۷:۳۳

- نیازی نیست. کرش بسیار پدیده‌ی نیکویی است و مشخصه‌ی وجود مشکل در سیستم. ELMAH را به پروژه اضافه کنید برای ثبت خودکار جزئیات استثناهای رخ داده و همچنین صفحه‌ی عمومی نمایش خطایی رخ داده‌است (کاربر به دلایل امنیتی نباید هیچ صفحه‌ی دیگری را در این حالت مشاهده کند). همین کافی است. پس از یک مدت کار کردن با ELMAH به ارزش آن در بالا بردن کیفیت برنامه پی‌خواهید برد.
- من راه حل آماده‌ای برای سایر حالات EF ندارم. این سری فقط code first بوده.

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۲/۱۲/۱۰ ۱۱:۳۱

با سلام
در زمان پیاده سازی الگوی واحد کار در برنامه‌های MVC زمانی که در کنترلر می‌خواهیم یک view بسازیم اگر گزینه create a strongly typed view انتخاب شود از بخش model Class باید کدام مدل را انتخاب کنیم؟
فایل‌های داخل پروژه Model که به عنوان ViewModel تعریف شده اند یا ...! ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۰ ۱۲:۱۶

این مساله ارتباطی به الگوی واحد کار ندارد. شما به عنوان برنامه نویس باید پس از بررسی تشخیص دهید که آیا خطر mass assignment در حین کار با شیء در حال دریافت از کاربر (هر نامی که دارد)، برنامه را تهدید می‌کند یا خیر. همچنین آیا View در حال استفاده نیاز به چند Model برای کار کردن دارد یا خیر. در این حالات استفاده از ViewModel توصیه می‌شود. در غیراینصورت استفاده از Domain modelها نه مشکل امنیتی را به همراه خواهند داشت و نه برای صرفا گزارش گیری، کم و کسری دارند.

نویسنده: رضا شش
تاریخ: ۱۳۹۲/۱۲/۱۸ ۱۷:۱۹

با الگوی uow ظاهرا می‌توان کانتکس‌های مختلف تعریف کرد اما سوال من این است که مثلا به دلایلی کلاس مدل من مثل Order به دلیل ساختار مختلف دیتابیس در سالهای مختلف فرق می‌کند قبلا برای رفع این مشکل از الگوی Factory استفاده می‌کردم اما در EF CodeFirst چگونه باید این تنظیمات در کانتکس تعریف شود. ظاهرا همه این کارها در پشت صحنه و زمان اجرا انجام می‌شود و خودکار DbSetها پر می‌شوند.

نویسنده: وحید نصیری

تاریخ: ۱۳۹۲/۱۲/۱۸ ۱۸:۴

« استفاده از چندین Context در EF 6 Code first »

نویسنده: سنائی
تاریخ: ۱۳۹۲/۱۲/۱۹ ۷:۲۵

چنانچه بخواهیم از BoundedContext استفاده کنیم، نحوه استفاده از الگوی واحد کار به چه صورت است؟ فرضا چنانچه SaleDbContext, ShippmentDbContext داشته باشیم که هر دو از IUnitOfWork به ارث رفته اند و در یک سرویس بخواهیم عملیاتی را در هر دو Context و طی یک transaction انجام دهیم، Ioc container هنگام وهله سازی IUnitOfWork، چه کلاسی را بایستی new کند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۱۹ ۹:۳۷

« آشنایی با TransactionScope » مطلب + نظرات
« استفاده از چندین Context در EF 6 Code first » نکات قسمت داشتن چندین Context در برنامه و مدیریت تراکنش‌ها
« Shrink EF Models with DDD Bounded Contexts » برای روش مدیریت بانک اطلاعاتی

نویسنده: رضا شش
تاریخ: ۱۳۹۲/۱۲/۲۰ ۹:۴۶

با عرض معذرت ، من بخش چندین Context را مطالعه کردم ولی هنوز به پیاده سازی درست نرسیدم. نمونه مثال ساده ای که گذاشته ام فکر می‌کنم سوالم را واضح‌تر کند.

```
//search for person with ID = 1 in year 92.
            using (var context = new TestContextNew())
            {
                // در اینجا هم باید بنحوی بتوان با مشخص کردن سال مورد نظر اطلاعات از جدول مربوطه لود
شود
                //Info_92 مثلا برای سال 92 از جدول
                var result = from h in context.Info_News where h.ID == 1 select h;

                dataGridView1.DataSource = result.ToList();
            }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۰:۲۵

- می‌شود به ازای هر سال یک Context مجزا با Entityهای مجزا درست کرد. فایل مثالی که با دو Context کار می‌کند در نظرات همان مطلب « استفاده از چندین Context در EF 6 Code first » پیوست شده‌است: Sample25.cs
ولی این روش سبب خواهد شد مجبور شوید به ازای هر سال، کوئری‌های LINQ مختلفی را هم بنویسید. یعنی لایه سرویس برنامه را باید هربار بازنویسی کنید، فقط برای اینکه نمی‌خواهید ساختار بانک اطلاعاتی را به روز کنید. چرا؟
- EF با استفاده از امکانات Migration به سادگی ساختار بانک‌های اطلاعاتی را به صورت خودکار می‌تواند به روز کند. باید هم اینکار را انجام بدهید چون کوئری‌های مختلف LINQ شما نهایتا به SQL ترجمه شده و چون یک سری از فیلدها در بانک اطلاعاتی سال قبل حضور ندارند، عملا برنامه کار نخواهد کرد. یعنی قسمت عمده‌ای از برنامه شما (کل لایه سرویس) از کار می‌افتد. کامپایل شدن برنامه در این حالت مهم نیست. آیا مثلا تنها کوئری GetAll ایی که تهیه شده، بر روی تمام سال‌ها و با ساختارهای مختلف اجرا می‌شود؟ خیر.
- سپس برای کار با بانک‌های اطلاعاتی دارای یک ساختار و مربوط به سال‌های مختلف، امکان تعیین رشته اتصالی به ازای هر Context هست:

```
context.Database.Connection.ConnectionString = "...";
```

نویسنده: reza

تاریخ: ۱۳۹۲/۱۲/۲۷ ۱۷:۱۰

من می‌خواهم در پروژه ام از uow استفاده کنم و نیاز به متد زیر دارم

```
context.Database.SqlQuery(type, sql, parameters)
```

ولی نوعی که برمی گرداند از نوع DbRawSqlQuery می‌باشد. چگونه باید متدی سازگار با متد فوق را در uow بازنویسی کنم که uow وابسته به EF خاصی نشود.
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۲۷ ۱۷:۲۵

یک ToList به آخر آن اضافه کنید:

```
public class MyContext : DbContext, IUnitOfWork
    {
        // ...
        public IList<T> GetRows<T>(string sql, params object[] parameters) where T: class
        {
            return this.Database.SqlQuery<T>(sql, parameters).ToList();
        }
    }
```

نویسنده: reza
تاریخ: ۱۳۹۳/۰۱/۰۱ ۱:۳۸

ضمن تبریک سال نو.
منظورم متد غیر جنریک SqlQuery بود . نوع انتیتی در زمان اجرا مشخص می‌شود. این متد ToList ندارد.
کلا آیا روشی وجود دارد که بتوان در درون متد غیر جنریک نوع جنریک آن را صدا زد. مثلا همین SqlQuery هم جنریک دارد و هم غیر جنریک
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۱ ۹:۴۰

در این حالت خاص، خروجی متد را کلمه‌ی کلیدی dynamic قرار دهید. یک مثال

نویسنده: میرزایی
تاریخ: ۱۳۹۳/۰۱/۰۱ ۱۳:۵۱

با سلام و عرض تشکر فراوان به خاطر مطلب مفید تون
سوالی که با خواندن این مطلب برای من پیش آمده اینه که فرض بگیرید بعد از مدتی شما تصمیم می‌گیرید ORM تون رو عوض کنید و تبدیل کنید به Nhibernate ولی شما چون در همه جا از جمله درUI و Application Service از IUnitOfWork استفاده کرده اید چه طور می‌خواهید این کار را انجام دهید
با تشکر فراوان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۱ ۱۳:۵۸

- من قصد ندارم چنین تعویضی را انجام دهم. علتش را در اینجا توضیح دادم.
- در لایه UI فقط از اینترفیس‌های لایه سرویس استفاده شده و این لایه از جزئیات پیاده سازی‌ها بی‌اطلاع است. کلاس‌های مورد نیاز از طریق تزریق وابستگی‌ها در اختیار آن قرار می‌گیرند. هر زمان که نیاز به تعویض بود، فقط پیاده سازی‌های لایه سرویس را

تغییر دهید.

نویسنده: behrouz
تاریخ: ۱۳۹۳/۰۱/۰۲ ۲۰:۴۰

سلام
در طراحی لایه سرویس شما کدامیک را پیشنهاد می‌کنید؟
به ازای هر موجودیت در لایه دومین یک کلاس سرویس داشته باشیم
یا
به ازای چندین موجودیت به هم وابسته در دومین یک کلاس سرویس.
آیا استانداردی در این زمینه وجود دارد؟

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۳/۰۱/۱۹ ۱۲:۵۲

با سلام
آیا نحوه کار با StructureMap در VS 2013 و MVC5 با .NET 4.5 متفاوت است؟
آخه من از نوگت StructureMap را نصب نمودم و در فایل global نیز کدهای شما را وارد نمودم ولی در این خط
;(()x.For<IUnitOfWork>().HttpContextScoped().Use(() => new Sample07Context

به HttpContextScoped و در این خط ;()ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects
به ReleaseAndDisposeAllHttpScopedObjects گیر میده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۹ ۱۳:۳

خیر. نگارش سوم structure map تغییراتی داشته که در انتهای مطلب « تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET Web forms » به صورت یک نکته‌ی تکمیلی مستند شده. مطلب جاری برای نگارش *2.6* تهیه شده بود.

نویسنده: مجتبی فخاری
تاریخ: ۱۳۹۳/۰۱/۲۰ ۰:۳۳

با سلام
الان باید اول structuremap , structuremap.web رو نصب کنم وبعد هم کدهای زیر را در فایل global بنویسم:

```
private static void initStructureMap()
        {
            ObjectFactory.Initialize(x =>
            {
                x.For<IPhoneTypeService>().Use<EFPhoneTypeService>();

                x.Policies.SetAllProperties(y =>
                {
                    y.OfType<IPhoneTypeService>();
                });
            });
            //Set current Controller factory as StructureMapControllerFactory
            ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
        }

        protected void Application_EndRequest(object sender, EventArgs e)
        {
            HttpContextLifecycle.DisposeAndClearAll ();
        }
    }

    public class StructureMapControllerFactory : DefaultControllerFactory
    {
        protected override IController GetControllerInstance(RequestContext requestContext, Type
```

```
controllerType)
        {
            return ObjectFactory.GetInstance(controllerType) as Controller;
        }
    }
```

نویسنده: رضایی
تاریخ: ۱۳۹۳/۰۱/۲۴ ۲۳:۱

با سلام؛ موقع اجرا با خطای زیر مواجه می‌شم:

```
StructureMap Exception Code:  202
No Default Instance defined for PluginFamily Iris.Datalayer.Context.IUnitOfWork, baran.Datalayer,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۴ ۲۳:۱۶

«No Default Instance» یعنی در تنظیمات اولیه IoC Container مورد استفاده، برای اینترفیس خاصی، کلاس پیاده سازی کننده‌ای را تعریف نکرده‌اید. برای مشاهده بحث مشابهی در این مورد به نظرات مطلب « تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET MVC » مراجعه کنید.

نویسنده: سارا کیانی
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۳:۹

سلام آقای نصیری من در حال نوشتن یک پروژه ویندوزی با روشهای گفته شده در این سایت هستم، منظور استفاده از EF Code First و StructureMap و ... می‌باشد. Base برنامه نوشته شده و شامل یک پروژه مرکزی- پروژه حسابداری مالی – حقوق دستمزد- خرید وفروش و.... می‌باشد، کاربران فقط و فقط از طریق اجرای سیستم مرکزی قادر به ورود به سیستم‌های نرم افزاری حوزه خود می‌باشند، با توجه به مطالب فوق الذکر و پرسش و پاسخ‌های کلیه دوستان، متوجه شدم که فقط باید از یک Context استفاده کرد، درسته؟ در حالیکه از بین این چند نرم افزار شاید شرکتی تنها قصد خرید واستفاده سیستم مالی وشرکتی دیگر سیستم مالی و n تای دیگر از برنامه‌ها رو خرید و مورد استفاده قرار دهد، و چون فعلا کلیه عملیات مرتبط با بانک اطلاعاتی در یک Context ودر MainProject انجام میشه، شرکتی که از یک نرم افزار من استفاده خواهد کرد همان ساختار جداول و بانک اطلاعاتی را دارد که شرکتی دیگر از چند نرم افزار دیگر از همین پروژه استفاده میکند. درکل نکته مبهم برام اینست که با چه تکنیک و روشی می‌توان از uow استفاده و پیروی کرد ولی هر پروژه Context خودش رو داشته باشد که با ورود به آن زیرسیستم، عملیات ساخت یا ویرایش DataBase و جداول مرتبط فقط بر روی آن زیر سیستم انجام شود و هیچ تاثیری رو جداول سیستم‌های دیگر نداشته باشد؟ (درضمن کلیه سیستم‌های از یک DataBase استفاده خواهند کرد). با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۳:۱۴

مطلب + نظرات « استفاده از چندین Context در EF 6 Code first »

نویسنده: بهنام
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۴:۶

با عرض سلام و خسته نباشید
من از روش ذکر شده در اجرای یک پروژه استفاده کردم و خیلی مفید بود.
به همین خاطر روش‌های مشابه رو سرچ کردم و به دو تا لینک زیر رسیدم : http://www.pr0g33k.com/blog/2003/Getting-Started-MVC-4-Entity-Framework-5-Code-First-and-Unity
http://geekswithblogs.net/danemorgridge/archive/2010/06/28/entity-framework-repository-amp-unit-of-work-t4-template-on.aspx
میخواستم ببینم که آیا این‌ها الگوی واحد کار رو نقض میکنن همونطور که گفته بودید یا خیر؟

و اینکه بین این دو کدوم روش بهتر هست؟ (در صورت عدم نقض الگوی واحد کار)
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۴:۲۴

هیچکدام. قسمت 11 را ابتدا مطالعه کنید (در مورد اینکه چرا نیازی به استفاده از الگوی مخزن با EF نیست).
سپس در سایت به مطالب تکمیلی زیر مراجعه کنید:
به الگوی Repository در لایه DAL خود نه بگویید!
پیاده سازی generic repository یک ضد الگو است
ایجاد Repositories بر روی UnitOfWork
نگاهی به generic repositories
بدون معکوس سازی وابستگی‌ها، طراحی چند لایه شما ایراد دارد

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۱ ۹:۶

به روز رسانی
کدهای قسمت جاری را به روز شده جهت استفاده از EF 6 و StructureMap 3 در VS 2013، از اینجا می‌توانید دریافت کنید:
EF_Sample07.zip

نویسنده: ارم وب
تاریخ: ۱۳۹۳/۰۷/۱۲ ۱۶:۱۹

سلام؛ تو این قسمت تمام کارهای که شما گفتید را انجام دادم با Structuremap جوای میگیرم ولی با Ninject فقط اطلاعات اولیه را که در متد Seed وارد کردم نشون میده یعنی برای نمایش جواب میده ولی زمانی که می‌خواهم عملیات درج را انجام بدم انجام نمیده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۲ ۱۷:۵۰

مفاهیم یکی هست. فقط تنظیمات اولیه IoC Containerها متفاوت است. برای Ninject یک افزونه‌ی خاص MVC تهیه شده: در اینجا
. پوشه‌ی MVC3 آن تا MVC 5 را هم پوشش می‌دهد. این افزونه یک مثال آماده هم دارد.
ضمنا تنظیم طول عمر یک وهله از UoW در طول یک درخواست توسط متد InRequestScope آن انجام می‌شود.

نویسنده: امیر
تاریخ: ۱۳۹۳/۰۷/۱۳ ۱۸:۴۵

در صورتی که بخواهیم Context را به ازای هر ویندوز فرم زنده نگه داریم در WinForm، پیاده سازی زیر صحیح است:

```
static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Database.SetInitializer<LoanContext>(null);

            // تنظیمات اولیه برنامه که فقط یکبار باید در طول عمر برنامه انجام شود
            ObjectFactory.Initialize(x =>
            {
                x.For<IUnitOfWork>().Use<LoanContext>();
                x.For<IEmployeeService>().Use<EmployeeService>();
```

```
            });

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmMain());
        }
    }
```

و در frmMain به شکل زیر پیاده سازی را انجام دهیم:

```
public partial class frmMain : Form
    {
        private IContainer _container;
        private IUnitOfWork _uow;

        public frmMain()
        {
            InitializeComponent();

            _container = ObjectFactory.Container.GetNestedContainer();
            _uow = _container.GetInstance<IUnitOfWork>();
        }

        private void btnHomeLoanRequest_Click(object sender, System.EventArgs e)
        {
            var employeeService = _container.GetInstance<IEmployeeService>();
            .
            .
            .
            _uow.SaveChanges();
        }
        private void btnEditLoan_Click(object sender, System.EventArgs e)
        {
            var categoryService = container.GetInstance<ICategoryService>();
            .
            .
            .
            _uow.SaveChanges();
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۳ ۱۹:۲۹

تزریق وابستگی‌های اصولی تا حد امکان از وجود خود IoC Container خالی است (^). یک نمونه پیاده سازی آن برای WinForms: (^)

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۷/۱۳ ۲۲:۱۹

باتوجه به توضیح شما، پیاده سازی صحیح‌تر بصورت زیر است؟

```
Program.cs:

Application.Run(ObjectFactory.GetInstance<frmMain>());
```

```
frmMain:
       private IContainer _container;
       private IUnitOfWork _uow;

       public frmMain(IUnitOfWork uow, IContainer container)
       {
           InitializeComponent();

           _container = container;
           _uow = uow;
       }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۳ ۲۲:۵۲

وابستگی‌ها (سرویس‌های مورد نیاز)، از طریق سازنده کلاس دریافت می‌شوند و نیازی نیست از طریق IContainer یا ObjectFactory که در اصل یک وابستگی اضافی داخل کلاسی خاص هستند، دریافت شوند. کلاس‌های شما باید *تا حد امکان* از GetInstance ها خالی باشند. نباید *تا جایی که ممکن است* بالای یک کلاس using StructureMap مشاهده شود (^ و ^) .

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۷/۱۴ ۸:۸

یک سوال:

در نگارش سوم Structure map، شما از دستور زیر استفاده کردید:

```
private void btnHomeLoanRequest_Click(object sender, System.EventArgs e)
        {
            using (var container = ObjectFactory.Container.GetNestedContainer()) // کانتکست را به صورت خودکار دیسپوز می‌کند
            {
                var uow = container.GetInstance<IUnitOfWork>();
                var employeeService = container.GetInstance<IEmployeeService>();
```

که با توجه به توضیح شما (^) نیازی به استفاده آن در متد فوق نیست و با تزریق frmMain در program.cs این وابستگی‌ها بوسیله container تزریق و تخریب می‌شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۴ ۹:۱۲

این نکته برای برنامه‌های کنسول و سرویس ویندوز است. در برنامه‌های WinForms و WPF با بسته شدن فرم، به صورت خودکار این اشیاء هم تخریب می‌شوند.

```
public class MyContext : DbContext
    {
        protected override void Dispose(bool disposing)
        {
            Debug.WriteLine("MyContext Dispose() is called.");
            base.Dispose(disposing);
        }
    }
```

برای بررسی بهتر این موضوع، متد فوق را به کلاس Context برنامه اضافه کنید. یک breakpoint روی آن قرار دهید و بعد فرم را باز کرده، با بانک اطلاعاتی کار کنید و سپس فرم را ببندید.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۷/۱۴ ۱۸:۳۴

آیا لازم است در ویندوز فرم‌ها بصورت صریح حین معرفی IUnitOfWork از متد CacheBy با پارامتر InstanceScope.ThreadLocal استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۴ ۱۸:۳۹

« نکته‌ای در مورد مدیریت طول عمر اشیاء در حالت HybridHttpOrThreadLocalScoped در برنامه‌های دسکتاپ »

نویسنده: رضا میر داماد
تاریخ: ۱۳۹۳/۰۹/۰۳ ۱۹:۵۰

عرض سلام. در مثالی که قرار دادین در متد جنریک Set در کلاس Context عمل رفع Shadowing انجام شده :

```
public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
```

ولی متد SaveChanges عمل Shadowing نشده!
چرا ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۰۳ ۲۰:۳۱

چون نام متد آن Save **All** Changes است و با نام متد کلاس پایه یکی نیست.

نویسنده: رضا میر داماد
تاریخ: ۱۳۹۳/۰۹/۰۳ ۲۰:۵۶

- اینطوری متد SaveChanges کلاس DbContext در لایه بالاتر(بعنوان مثال DataLayer) قابل دسترسی است , ولی اگر متد SaveChanges را Shadowing کنیم در کلاس Context , لایه بالاتر به متد SaveChanges کلاس DbContext دسترسی ندارد و فقط به متد SaveChanges در کلاس Context دسترسی دارد.
- چرا باید لایه بالاتر به متد SaveChanges کلاس DbContext دسترسی داشته باشد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۰۳ ۲۱:۳۳

- نیازی به بازنویسی SaveChanges اصلی نیست؛ چون تعریف و پیاده سازی آن در کلاس پایه DbContext قرار دارد.
- در اینجا متد جدید SaveAllChanges تعریف شد تا بدانید اگر خواستید پیش از SaveChanges اصلی، مثلا کار اعتبارسنجی دستی را انجام دهید (برای نمونه در برنامه‌های دسکتاپ مثلا)، چگونه می‌توان به آن دسترسی داشت.
- DbContext در DataLayer قرار دارد. زمانیکه با یک ORM کار می‌کنید، خود ORM همان DataLayer شما است. لایه‌ای که از آن استفاده می‌کند (لایه سرویس)، صرفا با اینترفیس IUnitOfWork کار می‌کند و تعاریف موجود در آن و نه چیزی بیشتر از آن.
- زمانیکه با IUnitOfWork کار می‌کنید، در هیچ قسمتی از برنامه قرار نیست مستقیما new MyContext مشاهده شود. تامین این وهله صرفا از طریق تزریق وابستگی‌ها صورت می‌گیرد (کل بحث جاری یا همان پیاده سازی الگوی Context Per Request بر همین مبنا است؛ یک وهله از Context در طول یک درخواست. نه اینکه هر جایی علاقمند بودیم یک new MyContext دستی نوشته شود).

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۱۰/۰۱ ۱۱:۵۶

عرض سلام ؛
آقای نصیری روش پیشنهادیتون برای Context Per Request در مدل DataBase First به چه صورته ؟
انتزاع IUnitOfWork به چه صورت باشه و Context به چه صورت پیاده سازی بشه با توجه به اینکه موجودیت‌های دامنه به صورت خودکار توسط EF تولید شده ...
کلا یک روش پیشنهادی برای Context Per Request در DataBase First توسط **Structure Map** مد نظرم هست
نظرتون راجع به این شکل چیه ؟

```
ObjectFactory.Initialize(x =>
            {
                x.For<Entities>().HttpContextScoped().Use(() => new Entities());
            });
```

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۱ ۱۴:۴۳

اگر از VS 2013 و EF 6 استفاده می‌کنید، حالت DB First آن در حقیقت مهندسی معکوس دیتابیس موجود به حالت Code first است. برای مثال اگر این فایل DbModel.edmx را تولید کند، ذیل آن فایل **tt**. DbModel.Context مشخص است که حاصل آن تولید خودکار یک چنین کلاسی است:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace EFDbFirstDependencyInjection.DataLayer
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class TestDbIdentityEntities : DbContext
    {
        public TestDbIdentityEntities()
            : base("name=TestDbIdentityEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Category> Categories { get; set; }
        public virtual DbSet<Product> Products { get; set; }
    }
}
```

این کلاس دقیقا از DbContext حالت Code first استفاده می‌کند و کلا ObjectContext قدیمی را کنار گذاشته‌اند (حتی برای حالت DB First).

بنابراین تمام نکات مطلب جاری در مورد حالت DB First موجود در VS 2013 صادق است. فقط باید فایل **tt**. DbModel.Context آن‌را اصلاح کنید تا IUnitOfWork را به صورت خودکار به انتهای تعریف کلاس Context اضافه کند. مابقی مسایل آن یکی است.

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۱۰/۰۱ ۱۵:۵۶

سپاس جناب نصیری .

در تکمیل بحث آقای نصیری ، برای افزوده شدن خودکار IUnitOfWork به DbContext ایجاد شده به صورت خودکار ، دوستانی که با T4 Templates آشنایی ندارن ، دقیقا خطوط زیر رو در مکان‌های مشخص شده اضافه کنید تا IUnitOfWork به صورت خودکار به DbContext اضافه بشه ...

ابتدا :

```
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext, IUnitOfWork
{
    public <#=code.Escape(container)#>()
        : base("name=<#=container.Name#>")
    {
<#
if (!loader.IsLazyLoadingEnabled(container))
{
#>
        this.Configuration.LazyLoadingEnabled = false;
<#
}
```

و سپس بعد از نوشته شدن FunctionImports کدهای زیر رو اضافه کنید و در نهایت بر روی Template راست کلیک کرده و run custom tool و در نهایت congratulation :)

```
    foreach (var edmFunction in container.FunctionImports)
    {
        WriteFunctionImport(typeMapper, codeStringGenerator, edmFunction, modelNamespace, includeMergeOption: false);
    }
#>

    public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
    {
      return base.Set<TEntity>();
    }

}

public interface IUnitOfWork
{
   IDbSet<TEntity> Set<TEntity>() where TEntity : class;
   int SaveChanges();
}
```

عنوان: **پیاده سازی UnitOfWork به وسیله MEF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۱/۱۲/۲۴ ۲۲:۵
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, MEF, UnitOfWork

در این پست قصد دارم یک UnitOfWork به روش MEF پیاده سازی کنم. ORM مورد نظر EntityFramework CodeFirst است. در صورتی که با MEF , UnitOfWork آشنایی ندارید از لینک‌های زیر استفاده کنید:

MEF

UnitOfWork

برای شروع ابتدا مدل برنامه رو به صورت زیر تعریف کنید.

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

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

سپس فایل Map رو برای مدل بالا به صورت زیر تعریف کنید.

```
public class CategoryMap : EntityTypeConfiguration<Entity.Category>
    {
        public CategoryMap()
        {
            ToTable( "Category" );

            HasKey( _field => _field.Id );

            Property( _field => _field.Title )
            .IsRequired();
        }
    }
```

برای پیاده سازی الگوی واحد کار ابتدا باید یک اینترفیس به صورت زیر تعریف کنید.

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

namespace DataAccess
{
    public interface IUnitOfWork
    {
        DbSet<TEntity> Set<TEntity>() where TEntity : class;
        DbEntityEntry<TEntity> Entry<TEntity>() where TEntity : class;
        void SaveChanges();
        void Dispose();
    }
}
```

DbContext مورد نظر باید اینترفیس مورد نظر را پیاده سازی کند و برای اینکه بتونیم اونو در CompositionContainer اضافه کنیم باید از Export Attribute استفاده کنیم.

چون کلاس DatabaseContext از اینترفیس IUnitOfWork ارث برده است برای همین از InheritedExport استفاده می‌کنیم.

```
[InheritedExport( typeof( IUnitOfWork ) )]
    public class DatabaseContext : DbContext, IUnitOfWork
    {
        private DbTransaction transaction = null;

        public DatabaseContext()
        {
            this.Configuration.AutoDetectChangesEnabled = false;
            this.Configuration.LazyLoadingEnabled = true;
        }
```

```
        protected override void OnModelCreating( DbModelBuilder modelBuilder )
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

            modelBuilder.AddFormAssembly( Assembly.GetAssembly( typeof( Entity.Map.CategoryMap ) ) );
        }

        public DbEntityEntry<TEntity> Entry<TEntity>() where TEntity : class
        {
            return this.Entry<TEntity>();
        }
    }
```

نکته قابل ذکر در قسمت OnModelCreating این است که یک Extension Methodبه نام AddFromAssembly (همانند NHibernate) اضافه شده است که از Assembly مورد نظر تمام کلاس‌های Map رو پیدا می‌کنه و اونو به ModelBuilder اضافه می‌کنه. کد متد به صورت زیر است:

```
public static class ModelBuilderExtension
    {
        public static void AddFormAssembly( this DbModelBuilder modelBuilder, Assembly assembly )
        {
            Array.ForEach<Type>( assembly.GetTypes().Where( type => type.BaseType != null &&
type.BaseType.IsGenericType && type.BaseType.GetGenericTypeDefinition() == typeof(
EntityTypeConfiguration<> ) ).ToArray(), delegate( Type type )
            {
                dynamic instance = Activator.CreateInstance( type );
                modelBuilder.Configurations.Add( instance );
            } );
        }
    }
```

برای پیاده سازی قسمت BusinessLogic ابتدا کلاس BusiessBase را در آن قرار دهید:

```
public class BusinessBase<TEntity> where TEntity : class
    {
        public BusinessBase( IUnitOfWork unitOfWork )
        {
            this.UnitOfWork = unitOfWork;
        }

        [Import]
        public IUnitOfWork UnitOfWork
        {
            get;
            private set;
        }

        public virtual IEnumerable<TEntity> GetAll()
        {
            return UnitOfWork.Set<TEntity>().AsNoTracking();
        }

        public virtual void Add( TEntity entity )
        {
            try
            {
                UnitOfWork.Set<TEntity>().Add( entity );
                UnitOfWork.SaveChanges();
            }
            catch
            {
                throw;
            }
            finally
            {
                UnitOfWork.Dispose();
            }
        }
    }
```

تمام متدهای پایه مورد نظر را باید در این کلاس قرار داد که برای مثال من متد Add , GetAll را براتون پیاده سازی کردم.
UnitOfWork توسط ImportAttribute مقدار دهی می‌شود و نیاز به وهله سازی از آن نیست
کلاس Category رو هم باید به صورت زیر اضافه کنید.

```
public class Category : BusinessBase<Entity.Category>
    {
        [ImportingConstructor]
        public Category( [Import( typeof( IUnitOfWork ) )] IUnitOfWork unitOfWork )
            : base( unitOfWork )
        {
        }
    }
```

.در انتها باید UI مورد نظر طراحی شود که من در اینجا از Console Application استفاده کردم. یک کلاس به نام Plugin ایجاد کنید و کدهای زیر را در آن قرار دهید.

```
public class Plugin
    {
        public void Run()
        {
            AggregateCatalog catalog = new AggregateCatalog();

            Container = new CompositionContainer( catalog );

            CompositionBatch batch = new CompositionBatch();

            catalog.Catalogs.Add( new AssemblyCatalog( Assembly.GetExecutingAssembly() ) );

            batch.AddPart( this );

            Container.Compose( batch );
        }

        public CompositionContainer Container
        {
            get;
            private set;
        }
    }
```

در کلاس Plugin توسط AssemblyCatalog تمام Export Attributeهای موجود جستجو می‌شود و بعد به عنوان کاتالوگ مورد نظر به Container اضافه می‌شود. انواع Catalog در MEF به شرح زیر است:
AssemblyCatalog : در اسمبلی مورد نظر به دنبال تمام Export Attributeها می‌گردد و آن‌ها را به عنوان ExportedValue در Container اضافه می‌کند.
TypeCatalog: فقط یک نوع مشخص را به عنوان ExportAttribute در نظر می‌گیرد.
DirectoryCatalog : در یک مسیر مشخص تمام Assembly مورد نظر را از نظر Export Attribute جستجو می‌کند و آن‌ها را به عنوان ExportedValue در Container اضافه می‌کند.
ApplicationCatalog : در اسمبلی و فایل‌های (EXE) مورد نظر به دنبال تمام Export Attributeها می‌گردد و آن‌ها را به عنوان ExportedValue در Container اضافه می‌کند.
AggregateCatalog : تمام موارد فوق را Support می‌کند.

کلاس Program رو به صورت زیر بازنویسی کنید.

```
class Program
    {
        static void Main( string[] args )
        {
            Plugin plugin = new Plugin();
            plugin.Run();

            Category category = new Category(plugin.Container.GetExportedValue<IUnitOfWork>());
            category.GetAll().ToList().ForEach( _record => Console.Write( _record.Title ) );
```

```
        }
    }
```

پروژه اجرا کرده و نتیجه رو مشاهده کنید.

نظرات خوانندگان

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۲:۵

سلام
سپاس از مطلبتون
در قسمت زیر شما نام کلاس CategoriMap رو ذکر کردید
در این صورت به ازای هر کلاس باید در این قسمت نام Map اون ذکر شود؟
خوب اگر قرار باشه به ازای هر کلاس نام Map آن ذکر شود دیگر نیازی به AddFormAssembly نبود مستقیماً نام CategoryMap. در modelBuilder اضافه میکردیم

```
protected override void OnModelCreating( DbModelBuilder modelBuilder )
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

            modelBuilder.AddFormAssembly( Assembly.GetAssembly( typeof( Entity.Map.CategoryMap ) ) );
        }
```

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۲۵ ۸:۲۵

اگر به پیاده سازی AddFromAsm دقت کنید یک ForEach داره. یعنی فقط شما یک اسمبلی رو بهش می‌دی، خودش مابقی رو پیدا می‌کنه . حتی اضافه کردن DbSetها هم قابلیت خودکار سازی داره .

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۱:۱۱

یک نکته که در مورد پیاده سازی بالا وجود داره اینه که متد save رو در خود توابع مربوط به repositoryها قرار داده اید و این با الگوی unitOfWork همخوانی نداره .

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۱:۲۵

بله در جریانم
اما در اینجا به صورت مستقیم نام MaP مستقیم ذکر شده
برای این این سوال برام پیش امد . چون اگر قرار بود خود کار ذکر بشن پس دیگه به ذکر Entity.Map.CategoryMap نبود

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۲:۳

مهم نیست. همینقدر که ایده اینکار مطرح شده مابقی‌اش هنر Reflection مصرف کننده است. مثلا از یک رشته (ذخیره شده در تنظیمات برنامه) هم می‌شود این نام‌ها را دریافت کرد: Assembly.Load

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۲۳:۴۶

متد Savechange در interface IUnitOfWork قرار دارد .اما در DatabaseContext پیاده سازی نشده که اون هم احتمالا یادشون رفته باشه.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۱/۱۲/۲۵ ۲۳:۵۳

در مورد سوال اول که چرا CategoryMap رو به متد AddFromAssembly پاس دادم.
متد AddFromAssembly نیاز به یک Assembly دارد تا بتونه تمام کلاس هایی رو که از کلاس EntityTypeConfiguration ارث برده اند رو پیدا کنه و اونها رو به صورت خودکار به ModelBuilder اضافه کنه. به همین دلیل من Assembly کلاس CategoryMap رو به اون پاس دادم. دقت کنید که اگر من n تا کلاس Map دیگه هم توی این ClassLibrary داشتم باز توسط همین دستور این کار به صورت خودکار انجام می‌شد. (پیشنهاد می‌کنم تست کنید)
نکته: این متد برگرفته شده از متد AddFromAssembly در NHibernate Session Configuration است.
البته بهتر است که یک کلاس پایه برای این کار بسازید و اون کلاس رو به AddFromAssembly پاس بدید.

در مورد سوال دوست عزیز مبنی بر اینکه متد Save رو در خود توابع Repository قرار دادم.
اگر به کدهای نوشته شده دقت کنید من اصلا مفهومی به نام Repository رو پیاده سازی نکردم. به این دلیل که خود DbContext ترکیبی از Repository Pattern , UnitOfWork است. متد SaveChange صدا زده شده همان متد SaveChange در DbContext است.
فرض کنید من یک کلاس Business دیگر به صورت زیر داشتم.

```
public class MyBusiness : BusinessBase<Entity.MyEntity>
    {
        [ImportingConstructor]
        public MyBusiness  ( [Import( typeof( IUnitOfWork ) )] IUnitOfWork unitOfWork )
            : base( unitOfWork )
        {
        }
        public void Add(Entity.MyEntity entity)
        {
            UnitOfWork.Set<MyEntity>().Add(entity);
        }
    }
```

حالا کد کلاس Business Category به صورت زیر تغییر می‌کنه.

```
public class Category : BusinessBase<Entity.Category>
    {
        [ImportingConstructor]
        public Category( [Import( typeof( IUnitOfWork ) )] IUnitOfWork unitOfWork )
            : base( unitOfWork )
        {
        }

        public override void Add( Entity.Category entity )
        {
            new MyBusiness().Add( new Entity.MyEntity() );
            UnitOfWork.Set<Entity.Category>().Add( entity );
            UnitOfWork.SaveChanges();
        }
    }
```

همان طور که می‌بینید فقط یک بار متد SaveChange فراخوانی شده است. Virtual کردن متد BusinessBase دقیقا به همین دلیل است.

نویسنده: علي
تاریخ: ۱۳۹۱/۱۲/۲۶ ۰:۲۶

کلاس پایه DbContext پیاده سازی SaveChanges رو داره.

نویسنده: MehRad
تاریخ: ۱۳۹۱/۱۲/۲۷ ۳:۴۹

تشکر میکنم بابت مقاله خوب و کاملتون.

نویسنده: Masoud
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۸:۴۶

اگه به جای category که شما تعریف کردین . موجودیتهای مثل خبرها یا آخرین نظرات و ... داشتیم چطور میتونیم اینا رو گروه بندی کنیم جوری که بشه هر کدوم رو در صفحه اصلی نشون داد؟مثلا همه موجودیت‌ها رو برسی کنه و بر اساس گروهشون اونایی که گروه خاصی دارن در قسمت منوی صفحه اصلی نمایش بده.ایا این امکان در MEF هست؟

نویسنده: ج.زوسر
تاریخ: ۱۳۹۲/۰۱/۱۷ ۲۱:۵۹

مرسی از مقاله خیلی خوبتون .

چه جور می‌تونم این روش روی local تست کنم .که اثرش رو مشاهده کنم .

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۲:۱۵

مهندس سلام
من از MEF برای تزریق کامپوننت هام به یک الگوی کار در MVC استفاده کردم، کار به درستی انجام میشه اما Attributeهای کلاس‌ها مثل پیام‌های خطا، طول فیلد و ... روی خروجی نهایی اعمال نمیشه و دیتابیس طبق پیش فرض‌ها ساخته میشه.
البته اگر از یک فایل کانفیگ (fluent API) جدا استفاده کنم مشکل حل میشه اما در این فایلها نمی‌توان پیام خطا برای حالت‌های مختلف تعریف کرد (البته به نظرم)

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۷:۲۵

در کدهای این مطلب نکات خودکار کردن تعاریف DbSetها در EF Code first ذکر نشدند. فقط نکات افزودن خودکار کلاس‌های تنظیمات نگاشت‌ها در EF Code first پیاده سازی شدند.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۴ ۲۱:۱۷

اما این پست‌ها ربطی به سوال من نداره قبلا همش بررسی کردم مهندس. مشکل توی عدم تزریق Metadataهای کلاس مانند DisplayName، ErrorMessageها و ... است که در FluentApi ظاهرا قابل پیاده سازی نیست

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۴ ۲۲:۴۴

من الان یک ویژگی StringLength به طول 30 رو روی خاصیت Title کلاس Category مقاله جاری اضافه کردم. با همین کدهای فوق، این فیلد با طول 30 الان در دیتابیس قابل مشاهده است. آغاز دیتابیس اصلا کاری به MEF نداره.

این هم فایل مثالی که در آن ویژگی طول رشته اعمال شده برای آزمایش:

MefSample.cs

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۵ ۹:۹

من کلاسهام به این شکله:

کلاس کانتکس‌های من

```
public class VegaContext : DbContext, IUnitOfWork, IDbContext
    {
#region Constructors (2)

        /// <summary>
        /// Initializes the <see cref="VegaContext" /> class.
        /// </summary>
        static VegaContext()
        {
            Database.SetInitializer<VegaContext>(null);
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="VegaContext" /> class.
        /// </summary>
        public VegaContext() : base("LocalSqlServer") { }

#endregion Constructors

#region Properties (2)

        /// <summary>
        /// Gets or sets the languages.
        /// </summary>
        /// <value>
        /// The languages.
        /// </value>
        public DbSet<Language> Languages { get; set; }

        /// <summary>
        /// Gets or sets the resources.
        /// </summary>
        /// <value>
        /// The resources.
        /// </value>
        public DbSet<Resource> Resources { get; set; }

#endregion Properties

#region Methods (2)

// Public Methods (1)

        /// <summary>
        /// Setups the specified model builder.
        /// </summary>
        /// <param name="modelBuilder">The model builder.</param>
        public void Setup(DbModelBuilder modelBuilder)
        {
            //todo
            modelBuilder.Configurations.Add(new ResourceMap());
            modelBuilder.Configurations.Add(new LanguageMap());
            modelBuilder.Entity<Resource>().ToTable("Vega_Languages_Resources");
            modelBuilder.Entity<Language>().ToTable("Vega_Languages_Languages");
            //base.OnModelCreating(modelBuilder);
        }
// Protected Methods (1)

        /// <summary>
        /// This method is called when the model for a derived context has been initialized, but
        /// before the model has been locked down and used to initialize the context.  The default
        /// implementation of this method does nothing, but it can be overridden in a derived class
        /// such that the model can be further configured before it is locked down.
        /// </summary>
        /// <param name="modelBuilder">The builder that defines the model for the context being
created.</param>
        /// <remarks>
        /// Typically, this method is called only once when the first instance of a derived context
        /// is created.  The model for that context is then cached and is for all further instances of
        /// the context in the app domain.  This caching can be disabled by setting the ModelCaching
        /// property on the given ModelBuidler, but note that this can seriously degrade performance.
        /// More control over caching is provided through use of the DbModelBuilder and
DbContextFactory
```

```
        /// classes directly.
        /// </remarks>
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new ResourceMap());
            modelBuilder.Configurations.Add(new LanguageMap());
            modelBuilder.Entity<Resource>().ToTable("Vega_Languages_Resources");
            modelBuilder.Entity<Language>().ToTable("Vega_Languages_Languages");
            base.OnModelCreating(modelBuilder);
        }

#endregion Methods

        #region IUnitOfWork Members
        /// <summary>
        /// Sets this instance.
        /// </summary>
        /// <typeparam name="TEntity">The type of the entity.</typeparam>
        /// <returns></returns>
        public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
        {
            return base.Set<TEntity>();
        }
        #endregion
    }
```

در تعاریف کلاسهایی که از IDBContext ارث می‌برن اکسپورت شدن (این یک نمونه از کلاس‌های منه)
در طرف دیگر برای لود کردن کلاس زیر نوشتم

```
public class LoadContexts
    {
        public LoadContexts()
        {
            var directoryPath = HttpRuntime.BinDirectory;//AppDomain.CurrentDomain.BaseDirectory;
//"Dll folder path";

            var directoryCatalog = new DirectoryCatalog(directoryPath, "*.dll");

            var aggregateCatalog = new AggregateCatalog();
            aggregateCatalog.Catalogs.Add(directoryCatalog);

            var container = new CompositionContainer(aggregateCatalog);
            container.ComposeParts(this);
        }

        //[Import]
        //public IPlugin Plugin { get; set; }

        [ImportMany]
        public IEnumerable<IDbContext> Contexts { get; set; }
    }
```

و در کانتکس اصلی برنامه این پلاگین هارو لود می‌کنم

```
public class MainContext : DbContext, IUnitOfWork
    {
        public MainContext() : base("LocalSqlServer") { }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            var contextList = new LoadContexts(); //ObjectFactory.GetAllInstances<IDbContext>();
            foreach (var context in contextList.Contexts)
                context.Setup(modelBuilder);

            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MainContext, Configuration>());
            //Database.SetInitializer(new DropCreateDatabaseAlways<MainContext>());
        }

        /// <summary>
        /// Sets this instance.
        /// </summary>
        /// <typeparam name="TEntity">The type of the entity.</typeparam>
        /// <returns></returns>
        public IDbSet<TEntity> Set<TEntity>() where TEntity : class
```

```
        {
            return base.Set<TEntity>();
        }
    }
```

با موفقیت همه پلاگین‌ها لود میشه و مشکلی در عملیات نیست. اما Attributeهای کلاس هارو نمیشناسه. مثلا پیام خطا تعریف شده در MVC نمایش داده نمیشه چون وجود نداره ولی وقتی کلاس مورد نظر از IValidatableObject ارث میبره خطای‌های من نمایش داده میشه. می‌خوام از خود متادیتاهای استاندارد استفاده کنم.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۵ ۹:۵۱

- بنابراین روی ساختار دیتابیس تاثیر داره. مثالش هم پیوست شد برای آزمایش.

- عمل نکردن خطاهای اعتبارسنجی به بود و نبود یک سری از تعاریف لازم در View هم بر می‌گرده. توضیح شما یعنی عمل نکردن اعتبارسنجی سمت کلاینت ولی عمل کردن اعتبارسنجی سمت سرور. به ترتیب باید jquery.validate.min.js ، jquery.min.js و jquery.validate.unobtrusive.min.js به View الحاق شده باشند. تنظیمات ClientValidationEnabled و UnobtrusiveJavaScriptEnabled در وب کانفیگ فعال باشند. از متدهایی مانند ValidationMessageFor استفاده شده باشد. این متدها یک سری ویژگی‌های خاص unobtrusive رو به عناصر HTML برای شناسایی توسط jquery.validate اضافه می‌کنند و بدون این‌ها عملا اعتبارسنجی سمت کاربر رخ نمی‌ده.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۵ ۲۰:۳۵

ممنون از پاسخ شما.
اما مهندس توی کامنت قبلی گفتم "با موفقیت همه پلاگین‌ها لود میشه و مشکلی در عملیات نیست. اما Attributeهای کلاس هارو نمیشناسه. مثلا پیام خطا تعریف شده در MVC نمایش داده نمیشه چون وجود نداره ولی وقتی کلاس مورد نظر از IValidatableObject ارث میبره خطای‌های من نمایش داده میشه. می‌خوام از خود متادیتاهای استاندارد استفاده کنم. "

پس خطا نمایش داده میشه و مشکلی توی طرف کلاینت ندارم.
در هر صورت ممنون از اینکه وقت گذاشتید و پاسخ دادید.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۵ ۲۱:۵۳

پردازش IValidatableObject سمت سرور هست. فقط نمایش نتیجه این نوع اعتبار سنجی سمت سرور، در سمت کلاینت بعد از post back کامل نمایش داده میشه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۶ ۱۰:۵۸

بله درسته
بعد از تست متوجه شدم وقتی خودم متا دیتا تعریف می‌کنم(ارث بری از متادیتای استاندارد) خطای طرف کلاینت عمل نمی‌کنه اما وقتی از متادیتای استاندارد خود دات نت استفاده میکنم خطای طرف کلاینت فعال نمیشه

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۶ ۱۱:۴

مطلب چطور باید سؤال پرسید رو اگر از ابتدا رعایت کرده بودید بحث به درازا نمی‌کشید. (سؤالی که در هر مرحله داره صورت مساله توضیح داده نشده‌اش عوض میشه؛ مثالی که نمی‌تونی از راه دور سریع تستش کنی و جزئیات متغیرش مشخص نیست)

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۷:۱۴

اگر امکان دارد سورس مثال را در سایت قرار دهید.
با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۸:۳۴

من کمی بالاتر ارسالش کردم.