

فرض می‌کنیم که یک Enum بصورت زیر داریم :

```
[Flags]
public enum Gender : byte
{
    None=0, Male=1, Female=2,
};
```

حال می‌خواهیم از این Enum در یک مدل ساده استفاده کنیم. از آنجا که EF هنوز قادر به پشتیبانی از Enum نمی‌باشد باید به روش زیر عمل کنیم:

(1) توسط data Annotation

```
public class User
{
    public int UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Username { get; set; }

    [Column(Name="Gender")]
    public int InternalGender { get; set; }
    [NotMapped]
    public Gender Gender
    {
        get { return (Gender)this.InternalGender; }
        set { this.InternalGender = (int)value; }
    }

    public DateTime DateOfBirth { get; set; }
}
```

(2) توسط Fluent API

```
modelBuilder.Entity<Participant>().Ignore(p => p.Gender);
modelBuilder.Entity<Participant>().Property(p => p.InternalGender).HasColumnName("Gender");
```

## نظرات خوانندگان

نویسنده: رضا

تاریخ: ۲۱:۱۰ ۱۳۹۱/۰۷/۰۶

ممنون مطلب مفیدیه. فقط به سوال: ما اگر به سری Enum مثلاً برای Priority داشته باشیم، بعد بخوایم در زمان استفاده توی UI به صورت فارسی نمایش داده بشه باید چکار کرد؟

نویسنده: وحید نصیری

تاریخ: ۲۲:۰۶ ۱۳۹۱/۰۷/۰۶

از ویژگی [Description](#) استفاده کنید.

نویسنده: اردلان شاه قلی

تاریخ: ۲۳:۵۶ ۱۳۹۱/۰۸/۲۰

سلام.

بسیار سپاسگزارم از زحمات شما.

در خصوص کد فوق یک سوال داشتم.

در قسمت تعریف خصوصیات مطابق کد زیر

[("Column(Name="Gender]

در هنگام کامپایل برنامه با خطای زیر روبرو می‌شوم.

'System.ComponentModel.DataAnnotations.Schema.ColumnAttribute' does not contain a definition for 'name'

نویسنده: وحید نصیری

تاریخ: ۱۰:۳۶ ۱۳۹۱/۰۸/۲۱

Column در اسمبلی EntityFramework.dll تعریف شده؛ هرچند فضای نام آن در اصل متعلق به اسمبلی EF نیست.

نویسنده: شاهین کیاست

تاریخ: ۱۱:۱۵ ۱۳۹۱/۰۸/۲۱

به کمک ReSharper فضای نام‌ها و اسمبلی‌هایی که از قلم افتاده را سریعتر پیدا می‌کنید.

نویسنده: کیا

تاریخ: ۱۴:۵۰ ۱۳۹۱/۰۸/۲۱

بهترین راه برای تعریف Enum در بانک و البته در راستای استفاده از اون در ef CodeFirst چیه؟

تعریف جدول بصورت id, title (که مثلاً id دقیقاً مقدار متناظر با چیزی که سمت کلاینت برای enum در نظر گرفتیم باشه) و

سپس سمت ef CF مثل یک جدول معمولی باهاش کار بشه؟

تعریف اعداد در یک جدول و map کردن با مقدار متناظر enum در سمت کلاینت؟

راه دیگه ای؟

خودتون با enumها به چه صورت برخورد می‌کنید و کار می‌کنید

اگر ممکن است زیر ef5 صحبت کنید چون اون خودش enum support داره ولی بعضیا هنوز مثل من مجبورن زیر نسخه 5 کار کنن

فرض کنید Stored Procedure ی با چند مقدار برگشتی را می‌خواهیم در EF CodeFirst مورد استفاده قرار دهیم. برای مثال Stored Procedure زیر را در نظر بگیرید:

```
CREATE PROCEDURE [dbo].[GetAllBlogsAndPosts]
AS
SELECT * FROM dbo.Blogs
SELECT * FROM dbo.Posts
```

ی Stord Procedure که توسط این دستور ساخته می‌شود تمام رکوردهای جدول Blogs و تمامی رکوردهای جدول Posts را واکشی کرده و به عنوان خروجی برمیگرداند (دو خروجی متفاوت). روش فراخوانی و استفاده از داده‌های این StoredProcedure در EF CodeFirst به صورت زیر است :

تعریف دو کلاس مدل Blog و Post به ترتیب برای نگهداری اطلاعات وبلاگ‌ها و پست‌ها در زیر آمده است. در ادامه نیز تعریف کلاس BloggingContext را مشاهده می‌کنید.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

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

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

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Objects;

namespace Sproc.Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BloggingContext())
            {
                db.Database.Initialize(force: false);

                var cmd = db.Database.Connection.CreateCommand();
                cmd.CommandText = "[dbo].[GetAllBlogsAndPosts]";

                try
                {
                    // اجرای پروسیجر
                    db.Database.Connection.Open();
                    var reader = cmd.ExecuteReader();

                    // خواند رکوردهای blogs
                }
            }
        }
    }
}
```

```

        var blogs = ((ObjectContextAdapter)db)
            .ObjectContext
            .Translate<Blog>(reader, "Blogs", MergeOption.AppendOnly);

        foreach (var item in blogs)
        {
            Console.WriteLine(item.Name);
        }

        // پخش به نتایج بعدی (همان)
        reader.NextResult();
        var posts = ((ObjectContextAdapter)db)
            .ObjectContext
            .Translate<Post>(reader, "Posts", MergeOption.AppendOnly);

        foreach (var item in posts)
        {
            Console.WriteLine(item.Title);
        }
    }
    finally
    {
        db.Database.Connection.Close();
    }
}
}
}

```

در کدهای بالا ابتدا یک Connection به بانک اطلاعاتی باز می‌شود:

```
db.Database.Connection.Open();
```

و پس از آن نوبت به اجرای Stored Procedure می‌رسد:

```
var reader = cmd.ExecuteReader();
```

در کد بالا پس از اجرای Stored Procedure نتایج بدست آمده در یک reader ذخیره می‌شود. شئی reader از نوع DBDataReader می‌باشد. پس از اجرای Stored Procedure و دریافت نتایج و ذخیره سازی در شئی reader، نوبت به جداسازی رکوردها می‌رسد. همانطور که در تعریف Stored procedure مشخص است این Stored Procedure دارای دو نوع خروجی از نوع‌های Blog و Post می‌باشد و این دو نوع باید از هم جدا شوند. برای انجام این کار از متد Translate شئی Context استفاده می‌شود. این متد قابلیت کپی کردن نتایج موجود از یک شئی DBDataReader به یک شئی از نوع مدل را دارد. برای مثال:

```

var blogs = ((ObjectContextAdapter)db)
    .ObjectContext
    .Translate<Blog>(reader, "Blogs", MergeOption.AppendOnly);

```

در کدهای بالا تمامی رکوردهایی از نوع Blog از شئی reader خوانده شده و پس از تبدیل به نوع Blog درون شئی Blogs ذخیره می‌شود.

پس از آن توسط حلقه foreach محتویات Blogs پیمایش شده و مقدار موجود در فیلد Name نمایش داده می‌شود.

```

foreach (var item in blogs)
{
    Console.WriteLine(item.Name);
}

```

با توجه به اینکه حاصل اجرای این Stored Procedure دو خروجی متفاوت بوده است، پس از پیمایش رکوردهای Blogs باید به سراغ نتایج بعدی که همان رکوردهای Post می‌باشد برویم. برای اینکار از متد NextResult شئی reader استفاده می‌شود:

```
reader.NextResult();
```

در ادامه برای خواندن رکوردهایی از نوع Post نیز به همان روشی که برای Blog انجام شد عمل می‌شود. [مطالعه بیشتر](#)

## نظرات خوانندگان

نویسنده: ashi mashi  
تاریخ: ۱۴:۳۶ ۱۳۹۲/۰۲/۱۱

لطفا نحوه اجرا کردن پروسیجرها با مقادیر مختلف در ورودی هم کمی توضیح می‌دادید ممنون می‌شدم.

با تشکر از توضیحات خوبتون،

نویسنده: محسن خان  
تاریخ: ۱۴:۵۴ ۱۳۹۲/۰۲/۱۱

[استفاده مستقیم از عبارات SQL در EF Code first](#) و مثال void runSp آن.

نویسنده: بهمن آبادی  
تاریخ: ۱۶:۱۷ ۱۳۹۲/۰۲/۱۱

منظور من روش مشابه parameterAttribute ها در mapping کردن ورودی‌های پروسیجرها در linq می‌باشد. نه اینکه صرفا از دستورات sql بصورت command استفاده شود.

مانند استفاده از پروسیجرها با چند ورودی و multiResult بودن آن در linq

نویسنده: محسن خان  
تاریخ: ۱۶:۲۷ ۱۳۹۲/۰۲/۱۱

در مورد نحوه اجرای رویه‌های ذخیره شده با ورودی‌های مختلف پرسیدید، روش اجرایش تا EF 5.0 به همین صورتی است که [ملاحظه می‌کنید](#).

[قرار است در EF 6.0](#) این مساله با Fluent API به نحو دیگری پوشش داده شود.

نویسنده: بهمن آبادی  
تاریخ: ۱۶:۴۱ ۱۳۹۲/۰۲/۱۱

دقیقا منظورم همون لینکه EF 6.0 که گفتین هستش.

Road Map برای انتشار نسخه جدید EF 6.0 وجود داره که قراره نهایتا تا کی انتشار پیدا بکنه ؟

بازم ممنون.

نویسنده: سید مهدی فاطمی  
تاریخ: ۲۱:۴۷ ۱۳۹۲/۰۷/۰۴

تشکر دوست عزیز

اما در مدل من که از دیتابیس گرفتم شی رویه ذخیره شده هم به عنوان یک تابع شبیه سازی شده و مثل شما عمل نکردم اما نمی‌تونم مقادیر رو از این شی بگیرم.

```
var sp1=_db.splogin(user,pas,value1,value2);
```

در توضیح این کد هم بگم که این اس پی 4 پارامتر داره و قراره دو مقدار رو برای من برگردونه .

نویسنده: سانای رحیمی  
تاریخ: ۱۹:۳۹ ۱۳۹۳/۰۴/۲۲

سلام

اگر SP ما بدون پارامتر out باشد و ما در آخر دستورات آن از دستور SCOPE\_IDENTITY () استفاده کرده باشیم چگونه می‌توانیم مقدار آن را به دست بیاوریم؟

نویسنده: محسن خان  
تاریخ: ۱۱:۰ ۱۳۹۳/۰۴/۲۳

در مطلب [استفاده مستقیم از عبارات SQL در EF Code first](#) بحث شده. از متد جنریک `db.Database.SqlQuery` باید استفاده کنید.

در کنار کتابخانه [elmah](#) که وظیفه ثبت تمامی خطاهای برنامه را دارد کتابخانه [MiniProfiler](#) امکان یافتن مشکلات کارایی و تنگنای وب سایت را در اختیارمان قرار می‌دهد. دو قابلیت عمده که این ابزار فراهم می‌نماید امکان مشاهده و بررسی کوئری‌های خام ADO.NET از قبیل SQL Server, Oracle و LINQ-to-SQL و EF/First Code و ... نمایش زمان اجرای عملی صفحات

برای استفاده از این ابزار کفایت تا آن را از nuget دریافت نمایید

```
PM> Install-Package MiniProfiler
```

در ASP.NET MVC در صفحه \_Layout\_ قبل از بسته شدن تگ body تابع RenderIncludes را مانند زیر صدا بزنید تا در همه صفحات نمایش داده شود

```
@using StackExchange.Profiling;
<head>
...
</head>
<body>
...
@MiniProfiler.RenderIncludes()
</body>
```

در کلاس global کد زیر را برای اجرای MiniProfiler اضافه نمایید

```
protected void Application_BeginRequest()
{
    if (Request.IsLocal)
    {
        MiniProfiler.Start();
    }
}

protected void Application_EndRequest()
{
    MiniProfiler.Stop();
}
```

برای پیکربندی MiniProfiler در web.config کد زیر را اضافه نمایید

```
<system.webServer>
...
<handlers>
  <add name="MiniProfiler" path="mini-profiler-resources/*" verb="*"
        type="System.Web.Routing.UrlRoutingModule"
        resourceType="Unspecified"
        preCondition="integratedMode" />
</handlers>
</system.webServer>
```

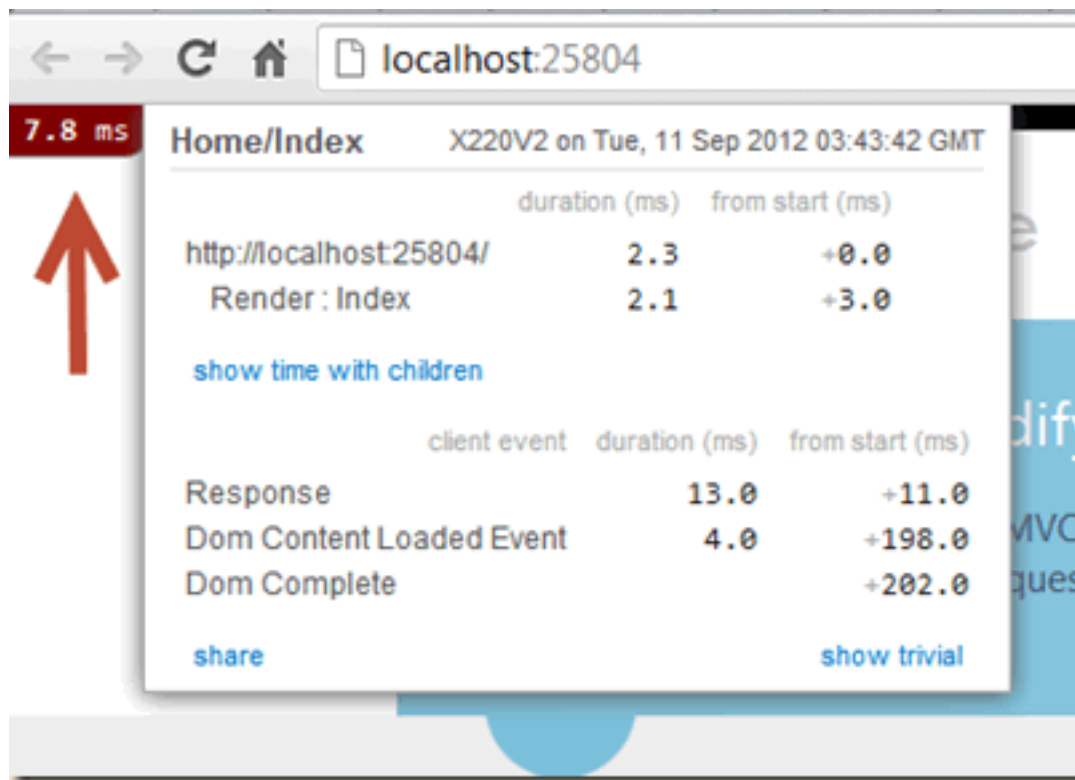
یا کتابخانه MiniProfiler.MVC را از nuget دریافت نمایید

```
PM> Install-Package MiniProfiler.MVC
```

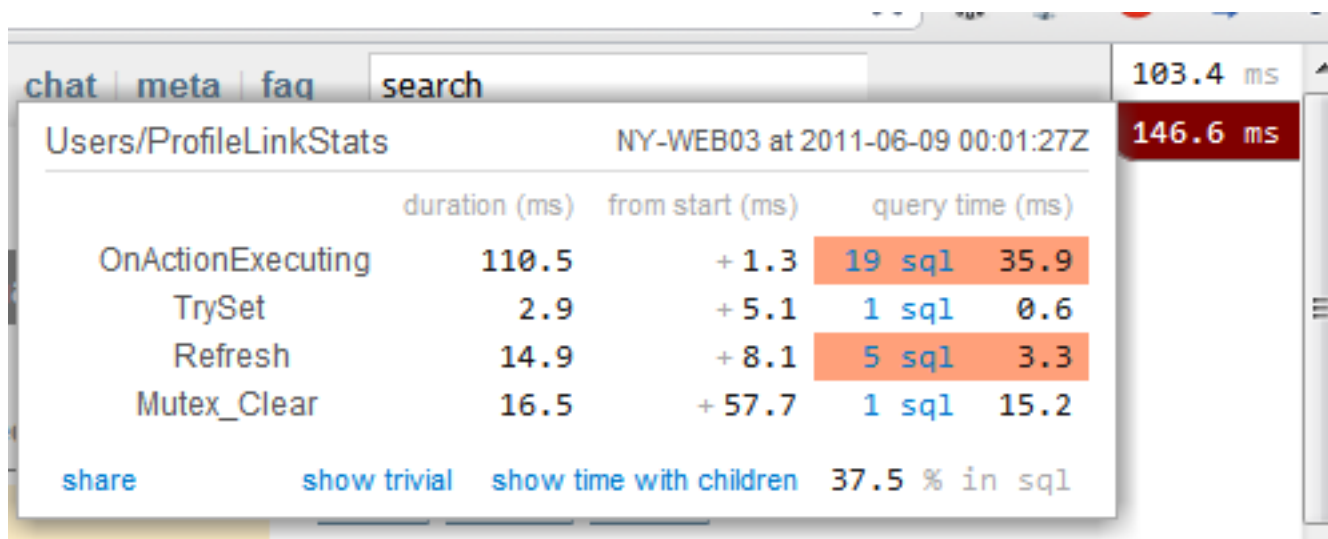
با اضافه شدن این کتابخانه همه پیکربندی بصورت صورت خودکار انجام می‌گیرد. حال وب سایت را اجرا کنید در بالای صفحه



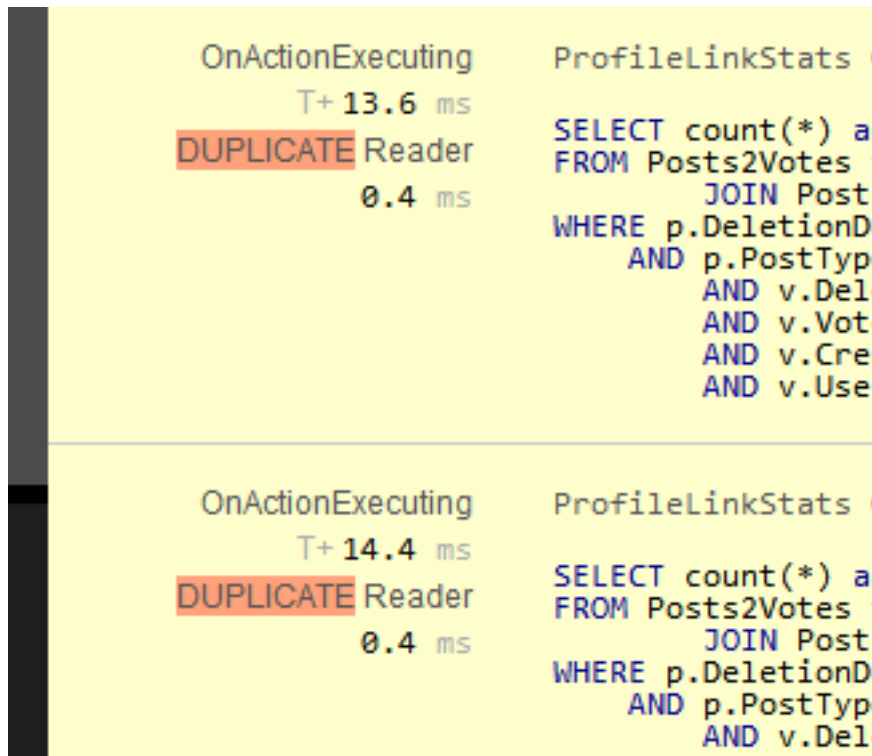
مانند شکل زیر مدت زمان بارگذاری صفحه نمایش داده می‌شود که با کلیک بر روی آن اطلاعات بیشتری را مشاهده می‌نمایید



اگر در اکشن اجرا شده کوئری اجرا شد باشد ستونی به نام query times نمایش داده می‌شود که تعداد کوئری‌ها و مدت زمان آن را نمایش می‌دهد



حال بر روی گزینه sql کلیک کنید که صفحه دیگری باز شود و کوئری خام آن را مشاهده نمایید اگر کوئری تکرار شده باشد در کنار آن با DUPLICATE متمایز شده است



برای مشاهده کوئری‌های Entity Framework/First Code کتابخانه MiniProfiler.EF را اضافه نمایید

```
PM> Install-Package MiniProfiler.EF
```

اگر بصورت دستی MiniProfiler را پی‌کربندی کرده باشید می‌بایست در Application\_Start دستور زیر را اجرا نمایید

```
protected void Application_BeginRequest()
{
    if (Request.IsLocal)
    {
        MiniProfiler.Start();
        MiniProfilerEF.Initialize();
    }
}
```

در حالت پیشرفته‌تر اگر قصد داشته باشید زمان یک قطعه کد را جداگانه محاسبه نمایید بصورت زیر عمل نمایید

```
public ActionResult Index()
{
    var profiler = MiniProfiler.Current;
    using (profiler.Step("Step 1"))
    {
        //code 1
    }
    using (profiler.Step("Step 2"))
    {
        //code 2
    }
    return View();
}
```

با این کار زمان هر step را بصورت جداگانه محاسبه می‌نماید. در ASP.NET Webforms به همین صورت استفاده می‌شود فقط

کافیست در masterpage اصلی یا اگر از masterpage استفاده نمی‌کنیم در صفحه مورد نظر تابع RenderIncludes را بصورت زیر صدا بزنیم

```
<%= StackExchange.Profiling.Miniprofiler.RenderIncludes() %>
```

امیدوارم مفید واقع شده باشد.

## نظرات خوانندگان

نویسنده: سعید

تاریخ: ۱۸:۳ ۱۳۹۱/۱۱/۱۵

ممنون از شما. آخرین باری که miniprofiler.ef رو تست کردم با ef5 کار نمی کرد و خطا می داد. شما تست کردید با آخرین نگارش ef code first؟

نویسنده: مجتبی کاویانی

تاریخ: ۱۸:۱۴ ۱۳۹۱/۱۱/۱۵

نه هیچ مشکلی ندارد من خودم با 5 ef code first تست کردم

نویسنده: Alireza Godazchian

تاریخ: ۸:۴۶ ۱۳۹۱/۱۱/۱۸

با سلام؛ آقای مهندس خیلی عالی توضیح دادید و بنده هم تست کردم و جواب گرفتم.  
با تشکر فراوان

نویسنده: امیر

تاریخ: ۲۰:۵۲ ۱۳۹۱/۱۱/۱۸

سلام چرا تو دانت ای اس پی کار نمیکنه فقط تو ام وی سی جواب میده  
MiniProfiler نمایشناسه

نویسنده: مجتبی کاویانی

تاریخ: ۲۲:۸ ۱۳۹۱/۱۱/۱۸

در ASP.NET هم بخوبی جواب می دهد آیا Handler را در Web.config اضافه نموده اید؟

نویسنده: هیمن صادقی

تاریخ: ۱۶:۵۳ ۱۳۹۳/۰۴/۲۷

با سلام و درود بی کران  
من کار که فرمودید انجام دادم اما زمانی که اجرا می کنم خطا  
"No Entity Framework provider found for the ADO.NET provider with invariant name 'System.Data.Odbc'. 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"

میدهد

از entity framework6 , mvc4 استفاده می کنیم  
در صورت امکان روش رفع خطا می دانید بیان نماید تا بتوانم از این ابزار استفاده کنم

نویسنده: وحید نصیری

تاریخ: ۱۸:۲۲ ۱۳۹۳/۰۴/۲۷

برای EF 6 یک بسته ی نیوگت جداگانه دارد. [اطلاعات بیشتر](#)

**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 ORM ها (فرقی نمی‌کند؛ 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 می‌کند این Id رو به مصرف کننده برگردونه (چون برنامه نقشی در تعیین اون نداره). یعنی عملاً با یک insert و بلافاصله با یک select مواجه خواهیم بود. البته این مورد هم باید اضافه بشه که ORM ها برای فرآیندها و تراکنش‌هایی کوتاه طراحی شدن و این مساله در حالت متداول کار با ORM ها اهمیتی نداره و اصلاً 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 مثلاً کلید اصلی توسط خود برنامه مدیریت بشه (مثلاً 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 می‌کند.

- 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 می‌کنه 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 مستقیماً 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.

## 1) رفتار متصل و غیر متصل در EF چیست؟

اولین نکته ای که به ذهنم می‌رسد اینه که برای استفاده از EF حتما باید درک صحیحی از رفتارها و قابلیت‌های اون داشته باشیم. نحوه استفاده از EF رو به دو رفتار متصل و غیر متصل تقسیم می‌کنیم.

حالت پیش فرض EF بر مبنای رفتار متصل می‌باشد. در این حالت شما یک موجودیت رو از دیتابیس فرا می‌خوانید 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 بياره.

رفتار متصل شاید در ویندوز اپلیکیشن کاربرد داشته باشه ولی در حالت وب اپلیکیشن کاربردی نداره چون با هر درخواستی 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 باشه 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 باین 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 روابط آن در EF چگونه است؟

با کد زیر می‌تونیم وضعیت یک موجودیت 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
```

اگر 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 جلوگیری 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) Include و AsNoTracking دو ابزار مهم در رفتار غیر متصل:

در صورتیکه 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 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 کاملاً 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: ۱۶:۹ ۱۳۹۲/۰۳/۰۹

سلام ... مطلب بسیار کاربردی بود ...  
در برنامه‌های وب با استفاده از ابزارهایی مَثِ Structuremap 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: ۱۷:۳ ۱۳۹۲/۰۳/۰۹

در حالت Detached (مثل ایجاد یک شیء CLR ساده)

در متد Update ایی 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 Find اش 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 سربرار 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 کارایی 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 { get; set; }
    public TimeSpan ListNormal { get; set; }
    public TimeSpan DetachedEntityDisabledTracking { get; set; }
    public TimeSpan DetachedEntityNormal { get; set; }
}

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(); // در هر بار باید محاسبه صورت گیرد
            //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 دارد 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). مثلاً 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 شیء اتچ شده بود که به اشتباه متد آپدیت رو قراردادام. (اصلاح شد)

```

T attachedEntity = set.Find(entity.Id);
var attachedEntry = dbContext.Entry(attachedEntity);
attachedEntry.CurrentValues.SetValues(entity);

```

نویسنده:

ایمان محمدی

تاریخ:

۱۷:۴۸ ۱۳۹۲/۰۳/۰۹

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. مثلاً 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:

۱۶:۲۸ ۱۳۹۲/۰۳/۳۰

سلام . من به راهنمایی می‌خواهم ممنون میشم کمک کنین برا آپدیت مشکل دارم، گذش 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:

۱۷:۴۹ ۱۳۹۲/۰۳/۳۰

احتمالاً رابطه‌های PersonelAction با دیگر موجودیت باعث این ارور شده، کدی که اینجا نوشتید برای پاسخ دادن خیلی ناقصه.

نویسنده:

arezoo

تاریخ:

۲۲:۲۴ ۱۳۹۲/۰۳/۳۰

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: ۲۳:۲۱ ۱۳۹۲/۰۳/۳۰

- ممکنه نتونسته باشید unit of work رو درست مدیریت کنید و در پاس دادن اون به لایه‌های مختلف، چند وهله ارزش ساخته شده. در این حالت خطای فوق رو می‌گیرد.
- ممکنه شئی 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: ۳:۷ ۱۳۹۲/۰۳/۳۱

- 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 به توضیحی بدین؟ ما توی هر فرم برای ذخیره‌ی تغییرات آیا یک instance از unit of work می‌سازیم؟

نویسنده: محسن جمشیدی  
تاریخ: ۸:۴۵ ۱۳۹۲/۰۳/۳۱

- 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 Modified قرار بدید. بنابراین در else نیاز هست که چک کنید personelAction.Id قبلا در DbContext موجود است یا خیر

نویسنده: محسن جمشیدی  
تاریخ: ۸:۵۲ ۱۳۹۲/۰۳/۳۱

- بستگی به فرم داره. اگر شما دو فرم داشته باشید که یکی Master هست 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: ۹:۹ ۱۳۹۲/۰۳/۳۱

- 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 سؤال 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: ۱۸:۵۶ ۱۳۹۲/۰۴/۰۱

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: ۱۸:۵۸ ۱۳۹۲/۰۴/۰۱

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: ۲۳:۲۲ ۱۳۹۲/۰۴/۰۱

به نظر من عنوان صحیح نیست. اصولا EF نامتصل هست. اینی که اینجا بحث شده در حقیقت دو حالت Dispose شده با نشده Context هست نه چیزی به عنوان Connected یا Disconnected

نویسنده: علیرضا  
تاریخ: ۲۳:۳۳ ۱۳۹۲/۰۴/۰۱

اگر این توضیح صحیح باشه باید برای هر تعداد از عناصر هم صحیح باشه. یعنی با هر تعداد شیئی در لیست. خوب حالا فرض کنید لیست ما 1 عنصر داره. این یعنی همون حالت "کار با یک شیئی ساده". چرا در این حالت توضیحی که شما گفتید صادق نیست؟

نویسنده: وحید نصیری  
تاریخ: ۲۳:۵۱ ۱۳۹۲/۰۴/۰۱

- 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 واکنشی 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 تغییری رخ داده و چگونه سریعاً باید محاسبات رو انجام بده. اما در حالت منقطع توسط متد DetectChanges تازه شروع به اتصال و محاسبه می‌کند.

نویسنده: وحید نصیری  
تاریخ: ۰۰:۰۰ ۱۳۹۲/۰۴/۰۲

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: ۱:۱۷ ۱۳۹۲/۰۴/۰۲

احتمالاً برداشت شما متصل بودن به دیتابیس است، اینجا منظور متصل بودن یک شیء به DbContext است. که این متصل بودن مزایایی همچون ردگیری تغییرات توسط DbContext را دارد.

نویسنده: علیرضا  
تاریخ: ۱۱:۴۸ ۱۳۹۲/۰۴/۰۲

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: ۱۱:۵۷ ۱۳۹۲/۰۴/۰۲

درسته. من کد رو با دقت نخونده بودم مخصوصا متد CustomUpdate. ممنون از توضیح دوباره.

نویسنده: مسعود2  
تاریخ: ۱۴:۴۵ ۱۳۹۲/۰۴/۲۱

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: ۱۸:۳۴ ۱۳۹۲/۰۴/۲۱

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 uncommment شود؟ طبق این [لینک](#).

نویسنده: وحید نصیری  
تاریخ: ۱۸:۵۳ ۱۳۹۲/۰۴/۲۱

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 objectی 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: ۱۳:۴۹ ۱۳۹۲/۰۷/۰۹

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: ۸:۳۸ ۱۳۹۲/۰۷/۱۱

[Using WCF Data Services 5.6.0 with Entity Framework 6](#)

### این ویدئو

به مدت حدوداً یکساعت و حجمی حدود 50 مگابایت، مربوط به یک کلاس آموزشی است که در ضمن آن به بررسی مقدماتی چگونگی بکاربردن روش CodeFirst برای تولید دیتابیس و جداول آن، پرداخته‌ام. پیشاپیش از نواقص و نارسایی‌های احتمالی آن، پوزش می‌طلبم.

سرفصل مطالبی که بطور مختصر مطرح شده‌اند، عبارتند از:

معرفی EF CodeFirst و کاربرد آن

استفاده از Nuget Package Manager برای افزودن EntityFramework

ایجاد کلاس نمونه User و معرفی DbContext جهت معرفی کلاس User به عنوان جدولی از دیتابیس

ایجادConnectionString و نکات مربوط به آن برای ایجاد صحیح جداول در SQL Server

چگونگی ایجاد فیلد کلیدی

روش ذخیره سازی اطلاعات در جدول

روش ایجاد رابطه یک به چند با ایجاد دو جدول کمکی Log و Work و مرتبط با جدول User

روش جستجو در جداول بدون استفاده مستقیم از SQL Query

## نظرات خوانندگان

نویسنده: صادق  
تاریخ: ۱۷:۵۶ ۱۳۹۲/۰۴/۰۶

با سلام و عرض ادب  
می‌خواستم بدونم ادامه این ویدئو را چطوری می‌تونم تهیه کنم . با تشکر

نویسنده: محسن خان  
تاریخ: ۱۸:۳۱ ۱۳۹۲/۰۴/۰۶

روی لینک «این ویدیو» در ابتدای مطلب کلیک کنید.

نویسنده: sorosh  
تاریخ: ۱۸:۴۹ ۱۳۹۲/۰۴/۰۶

با سلام؛ می‌خواستم بدونم ویدئو دیگه‌ای آیا این مجموعه داره یا همینه فقط ؟ مرسی

نویسنده: سیدمجتبی حسینی  
تاریخ: ۱۳:۰۰ ۱۳۹۲/۰۴/۰۷

سلام و درود.  
در بینابین جلسات آموزش سی شارپ پیشرفته به جهت تبیین یکی از کاربردهای نوین کدنویسی شیء‌گرا و همچنین تقویت انگیزه یادگیرندگان در یادگیری همه مطالب ، چه انتزاعی و چه کاربردی، به طرح این بحث فقط در یک جلسه اکتفا کردم.  
باتشکر

نویسنده: آگاه  
تاریخ: ۱۵:۳۶ ۱۳۹۲/۱۱/۲۱

نمیشه آموزش ویدیویی این بحث رو ادامه بدین؟

خیلی شیوا کارامد و مفید بود.. مخصوصا اینکه کلی جزئیات هم بینش گفته میشد..:)