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);
        }
    }
}
```

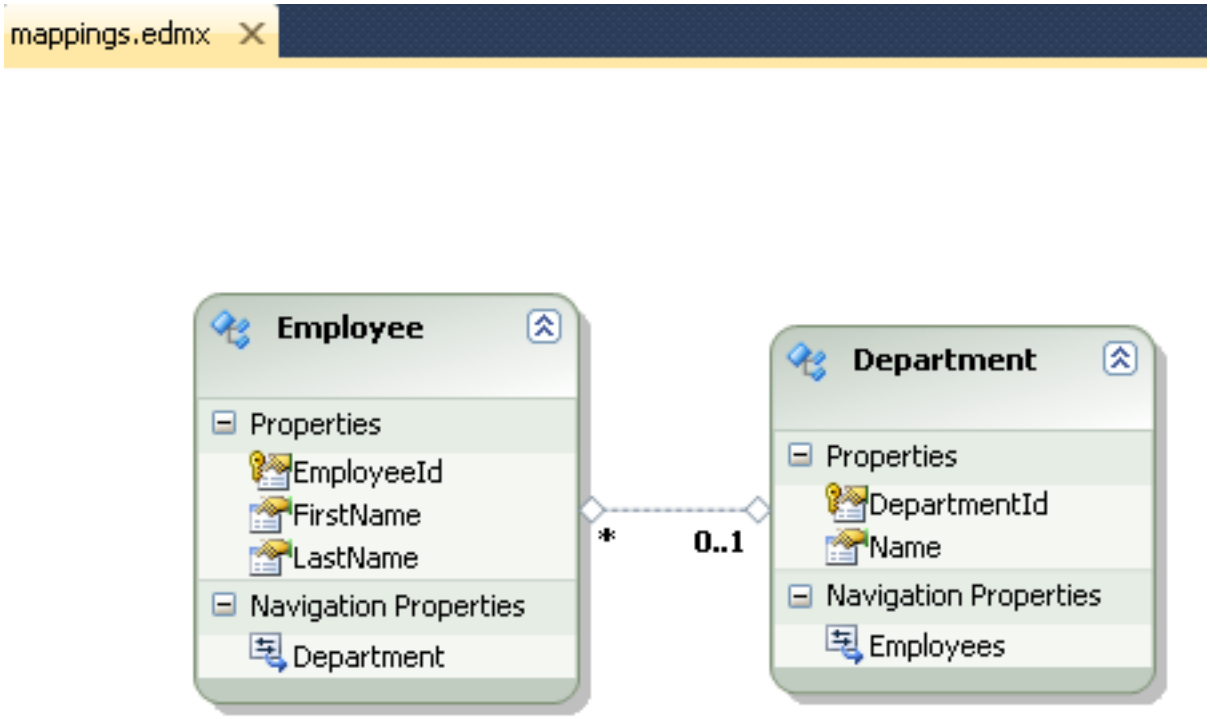
نکته: تهیه خروجی XML از نگاشت‌های خودکار تهیه شده

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:

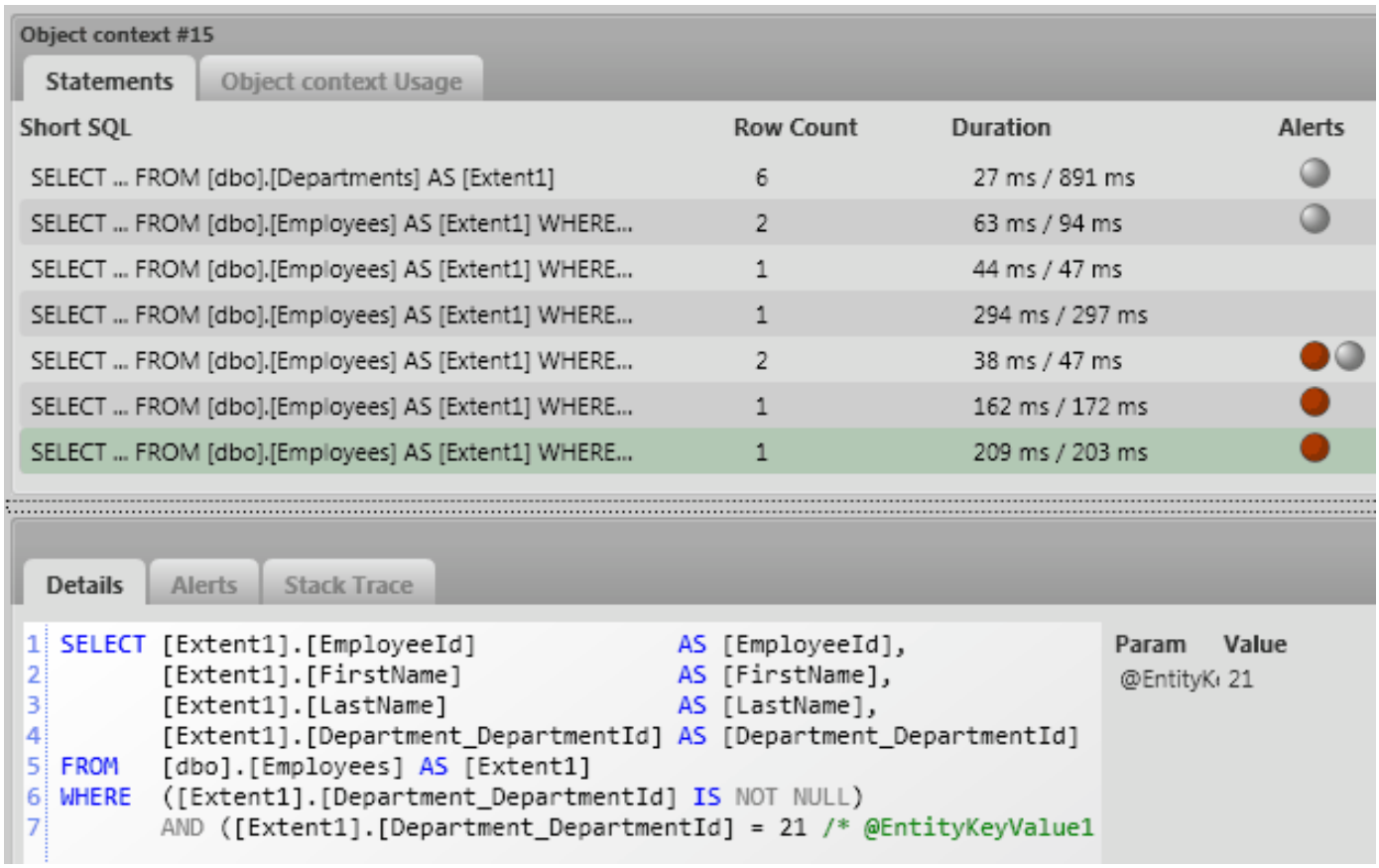
```
There is already an open DataReader associated with this Command which must be closed first
```

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!



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.
```

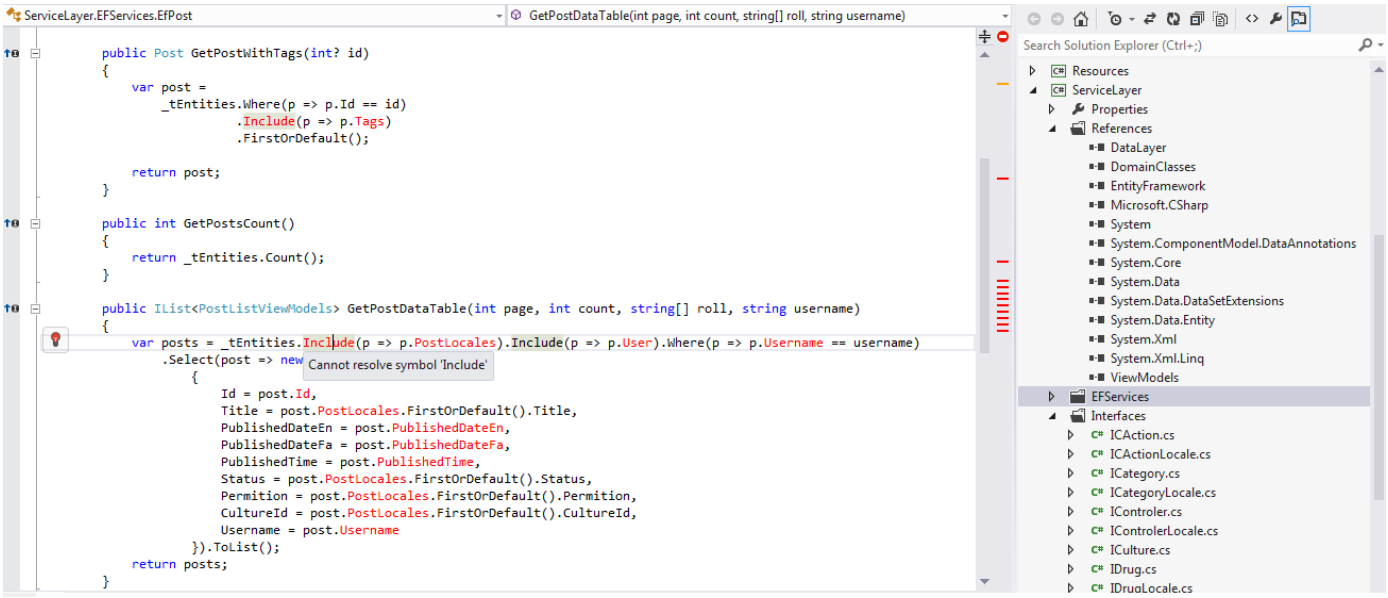
EF Prof با سیستم به روز رسانی بانک اطلاعاتی EF Code first تداخل ایجاد می‌کنه. اول دیتابیس رو به روز کنید. بعد تنظیمات EF Prof رو اضافه کنید و بعد هم آغاز دیتابیس رو با null مقدار دهی کنید.

- 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.