

## 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، TPC و TPT 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 مستقیماً 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'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 Context ها 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 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: ۱۰:۲۵ ۱۳۹۱/۰۴/۱۲

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: ۹:۵۹ ۱۳۹۱/۰۶/۱۲

با سلام .

- 1) کلاس PropertyReflector در کدام Namespace قرار دارد ؟
  - 2) چرا برای متد SaveChanges حالت EntityState.Deleted در نظر گرفته نشده است؟
- با تشکر.

نویسنده: وحید نصیری  
تاریخ: ۱۰:۳۰ ۱۳۹۱/۰۶/۱۲

- 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: ۱۸:۵ ۱۳۹۱/۰۶/۱۶

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: ۱۸:۱۶ ۱۳۹۱/۰۶/۱۶

ارث بری رو به صورت «is a» معرفی می‌کنند و می‌خوانند. یعنی هر دو حالتی که نام بردید یکی است و فرقی نمی‌کنه.

نویسنده: عرفان  
تاریخ: ۱۸:۲۴ ۱۳۹۱/۰۶/۱۶

گیج شدم آقای نصیری،میشه یکم موضوع رو بازش کنید؟  
پس اینجوری بگیم،به نظر شما کدوم استانداردتره(بهتره)1؟ 2 یا

نویسنده: وحید نصیری  
تاریخ: ۱۸:۴۰ ۱۳۹۱/۰۶/۱۶

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: ۱۸:۵۰ ۱۳۹۱/۰۶/۱۶

منم گیرما D:  
پس روش 2 استانداردتره؟درسته؟

نویسنده: وحید نصیری  
تاریخ: ۱۹:۱ ۱۳۹۱/۰۶/۱۶

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: ۱۹:۶ ۱۳۹۱/۰۶/۱۶

یه دنیا ممنون.

نویسنده: daneshjoo  
تاریخ: ۱:۳۸ ۱۳۹۲/۰۲/۱۰

مهندس عزیز , من در برنامه 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 یا نه.

چطوری می‌تونم این حالت رو چک کنم؟

برا ویرایش مشکلی نیست ، طبق گفته خودتون state رو چک می‌کنم اگه modified بود پیغام مایل به ذخیره تفرات رو می‌دم اما برا درج نمی‌دونم چه شرطی رو بنویسم

نویسنده: وحید نصیری  
تاریخ: ۸:۴۴ ۱۳۹۲/۰۲/۱۰

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: ۰:۲۳ ۱۳۹۲/۰۲/۱۱

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 اضافه 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 detected رو می‌ده و وقتی که ستونهای شی table1 رو مقدار دهی می‌کنم مقدار detected رو باز می‌ده و وقتی این شی رو با استفاده از متد savecheng در دیتابیس ذخیره می‌کنم بعد state رو چک می‌کنم مقدار unchanged رو بهم می‌ده

لطفا در این خصوص کمک کنید

نویسنده: وحید نصیری  
تاریخ: ۰:۳۹ ۱۳۹۲/۰۲/۱۱

- 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 مثلاً 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: ۲۲:۴ ۱۳۹۲/۰۲/۱۷

آقا وحید عزیز حرف شما درست بود و من تقریباً اشتباه فهمیده بودم .

من در برنامه اول میام یک شی از تیبل رو می‌سازم :

```
var t=new db.table1();
```



که اگر بیای state اونو بگیری بهت detached نشون میده  
و اگر بیای اونو به مدل اضافه کنی :

```
db.table1.add(t);
```

که اگر بیای state اونو بگیری بهت added نشون میده  
حالا سوال من اینه که اگر 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. صرفاً یک شیء معمولی تشکیل شده در حافظه است.  
+ لطفاً 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: مسعود 2  
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۸:۴۷

بهترین راه برای مدیریت تغییرات در یک سناریوی Disconnected چیست؟  
منظورم اینه که اگر از کلاسهای POCO استفاده کنیم و بخواهیم تغییرات سمت کلاینت (WinForm) روی Entityهای DbContext اعمال کنیم مناسب‌ترین راه چیست؟ (هم کشف تغییرات سمت کلاینت و هم اعمال اونها سمت سرور)

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۰۳/۰۴ ۱۹:۰۷

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: مسعود 2  
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 دقیقاً کدام یک از عملیات CRUD اتفاق افتاده.

نویسنده: مسعود م. پاکدل  
تاریخ: ۱۳۹۲/۰۳/۰۴ ۲۲:۰۸

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 دارد که با توجه به تغییرات و State هر Entity داده‌ها رو ذخیره می‌کند. 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 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 ۲۳:۰۰

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. نهایتاً 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 insert، update و delete رو محاسبه و اجرا می‌کند. نیازی به مدیریت خاصی بجز تعریف Context در سطح فرم نداره. به صورت خلاصه مرسوم نیست در مثلاً WinForms «متداول»، منقطع از Context کار کرد، چون اساساً می‌شود به سادگی، تا زمانی که یک فرم در حال نمایش است، Context و سیستم ردیابی خودکار آن را 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 نمی‌کند. در این حالت اصلاً Context سمت کلاینت وجود ندارد که بتونیم ردیابی خودکار اشیاء 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 می‌کند. 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 ( ^ )

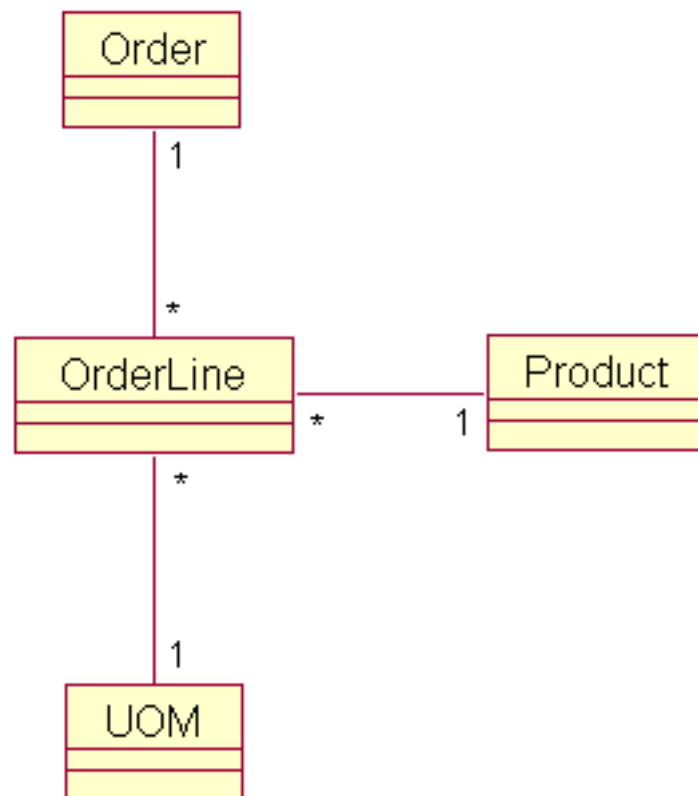
نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۰۳/۰۵ ۰:۴

- بحث Self tracking entities در حالت database first است و در Code first [پشتیبانی نمی‌شود](#) و احتمالاً هم [نخواهد شد](#).

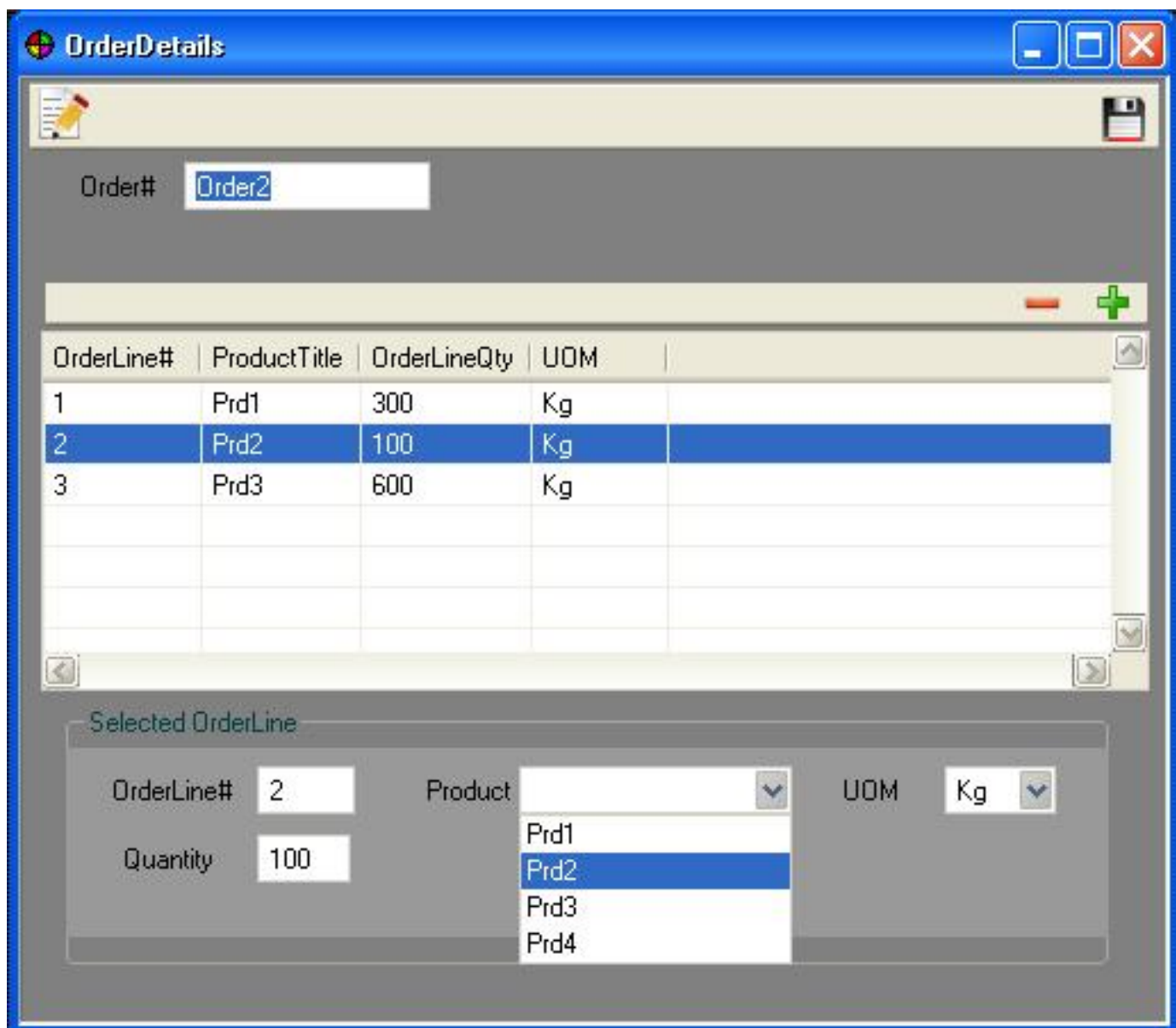
[در اینجا](#) رسماً پایین صفحه قید شده که دیگر از STE با EF 5.0 برای حالت‌های N-Tier استفاده نکنید. در EF Code first توصیه می‌شود که برای کار با WCF از WCF Data Services و یا RIA Services استفاده کنید. هر دوی این‌ها برای کار با EF Code first به روز شدن اخیراً. هر دوی این‌ها change tracking سمت کاربر رو هم پشتیبانی می‌کنند.

نویسنده: مسعود  
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:



Order#

OrderLine#	ProductTitle	OrderLineQty	UOM
1	Prd1	300	Kg
2	Prd2	100	Kg
3	Prd3	600	Kg

Selected OrderLine

OrderLine#  Product  UOM

Quantity

Product dropdown options: Prd1, Prd2, Prd3, Prd4

و کاربر پس از زدن دکمه ویرایش قادر به انجام کارهای زیر باشد:

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؟

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 عملاً 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

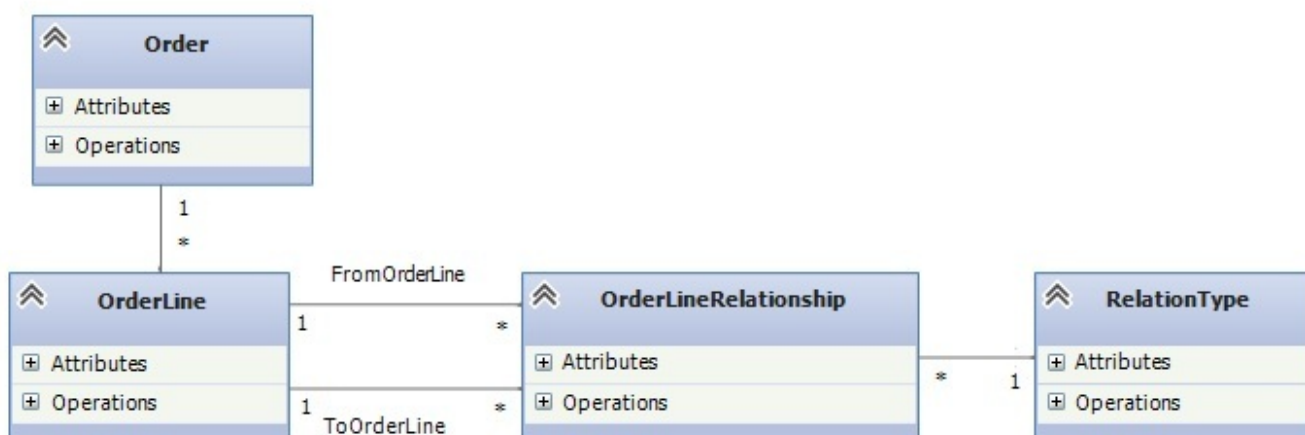
تاریخ: ۱۳۹۲/۱۰/۲۵ ۹:۵۳

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، داندولینکهای EditPostModel به جای آنها مینشینند، در واقع در صورت ویرایش داندولینکهای یک پست، ویرایش واقعی انجام نمیشود، بلکه این کار با حذف (sql Delete) کلیه داندولینکهای آن پست از DB و درج مجدد داندولینکهای 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 داندولینکها داده نشود باز هم حذف تمامی آنها و درج مجدد آنها صورت خواهد گرفت.

پایین آمدن کارایی وقتی که تعداد رکوردهای طرف چند رابطه 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 در دسرساز 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 زمانی که رکوردهای OrderLine را مستقیماً 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.

احتمالاً 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 تصمیم‌گیری 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 دقیقاً 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، آن را 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: مسعود  
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 دقیقاً 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: ۲۰:۵۹ ۱۳۹۲/۱۱/۱۰

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: ۲۱:۵۸ ۱۳۹۲/۱۱/۱۰

- 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: ۲۲:۱۷ ۱۳۹۳/۱۰/۰۱

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 مقداردی 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: ۲۳:۲۰ ۱۳۹۳/۱۰/۰۱

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: ۰:۱۱ ۱۳۹۳/۱۰/۰۲

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 مقداردی نشه داده تکراری ثبت میشه در هربار اجرا اگر هم فیلد کلید مقداردی بشه (به صورت دستی) خطای فوق الذکر رخ میده

نویسنده: صابر فتح الهی

تاریخ: ۲۰:۴۸ ۱۳۹۳/۱۰/۰۲

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: ۲۱:۴ ۱۳۹۳/۱۰/۰۲

از متد Any قبل از ثبت در متد Seed استفاده کنید:

```
if (!context.MemberSettings.Any())  
{  
    // ... add new MemberSettings  
}
```