

دو نوع حالت کلی کارکردن با EF وجود دارند: متصل و منقطع.

در حالت متصل مانند برنامه‌های متداول دسکتاپ، Context مورد استفاده در طول عمر صفحه‌ی جاری زنده نگه داشته می‌شود. در این حالت اگر شیء‌ای اضافه شود، حذف شود یا تغییر کند، توسط EF ردیابی شده و تنها با فراخوانی متد SaveChanges، تمام این تغییرات به صورت یکجا به بانک اطلاعاتی اعمال می‌شوند. در حالت غیرمتصل مانند برنامه‌های وب، طول عمر Context در حد طول عمر یک درخواست است. پس از آن از بین خواهد رفت و دیگر فرصت ردیابی تغییرات سمت کاربر را نخواهد یافت. در این حالت به روز رسانی کلیه تغییرات انجام شده در خواص و همچنین ارتباطات اشیاء موجود، کاری مشکل و زمانبر خواهد بود. برای حل این مشکل، کتابخانه‌ای به نام GraphDiff طراحی شده‌است که صرفاً با فراخوانی متد UpdateGraph آن، به صورت خودکار، محاسبات تغییرات صورت گرفته در اشیاء منقطع و اعمال آن‌ها به بانک اطلاعاتی صورت خواهد گرفت. البته ذکر متد SaveChanges پس از آن نباید فراموش شود.

## اصطلاحات بکار رفته در GraphDiff

برای کار با GraphDiff نیاز است با یک سری اصطلاح آشنا بود:

### Aggregate root

گرافی است از اشیاء به هم وابسته که مرجع تغییرات داده‌ها به شمار می‌رود. برای مثال یک سفارش و آیتم‌های آن را در نظر بگیرید. بارگذاری آیتم‌های سفارش، بدون سفارش معنایی ندارند. بنابراین در اینجا سفارش aggregate root است.

### AssociatedCollection/AssociatedEntity

حالت‌های Associated به GraphDiff اعلام می‌کنند که اینگونه خواص راهبری تعریف شده، در حین به روز رسانی aggregate root نباید به روز رسانی شوند. در این حالت تنها ارجاعات به روز رسانی خواهند شد. اگر خاصیت راهبری از نوع ICollection است، حالت AssociatedCollection است، و اگر صرفاً یک شیء ساده است، از AssociatedEntity استفاده خواهد شد.

### OwnedCollection/OwnedEntity

حالت‌های Owned به GraphDiff اعلام می‌کنند که جزئیات و همچنین ارجاعات اینگونه خواص راهبری تعریف شده، در حین به روز رسانی aggregate root باید به روز رسانی شوند.

## دریافت و نصب GraphDiff

برای نصب خودکار کتابخانه‌ی GraphDiff می‌توان از دستور نیوگت ذیل استفاده کرد:

```
PM> Install-Package RefactorThis.GraphDiff
```

## بررسی GraphDiff در طی یک مثال

مدل‌های برنامه آزمایشی، از سه کلاس ذیل که روابط many-to-many و one-to-many با یکدیگر دارند، تشکیل شده‌است:

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

```
namespace GraphDiffTests.Models
{
    public class BlogPost
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public virtual ICollection<Tag> Tags { set; get; } // many-to-many

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

        public BlogPost()
        {
            Tags = new List<Tag>();
        }
    }

    public class Tag
    {
        public int Id { set; get; }

        [StringLength(maximumLength: 450), Required]
        public string Name { set; get; }

        public virtual ICollection<BlogPost> BlogPosts { set; get; } // many-to-many

        public Tag()
        {
            BlogPosts = new List<BlogPost>();
        }
    }

    public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<BlogPost> BlogPosts { get; set; } // one-to-many
    }
}
```

- یک مطلب می‌تواند چندین برچسب داشته باشد و هر برچسب می‌تواند به چندین مطلب انتساب داده شود.
- هر کاربر می‌تواند چندین مطلب ارسال کند.

در این حالت، Context برنامه چنین شکلی را خواهد یافت:

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

namespace GraphDiffTests.Config
{
    public class MyContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<BlogPost> BlogPosts { get; set; }
        public DbSet<Tag> Tags { get; set; }

        public MyContext()
            : base("Connection1")
        {
            this.Database.Log = sql => Console.WriteLine(sql);
        }
    }
}
```

به همراه تنظیمات به روز رسانی ساختار بانک اطلاعاتی به صورت خودکار:

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

```
using System.Linq;
using GraphDiffTests.Models;

namespace GraphDiffTests.Config
{
    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            if(context.Users.Any())
                return;

            var user1 = new User {Name = "User 1"};
            context.Users.Add(user1);

            var tag1 = new Tag { Name = "Tag1" };
            context.Tags.Add(tag1);

            var post1 = new BlogPost { Title = "Title...1", Content = "Content...1", User = user1};
            context.BlogPosts.Add(post1);

            post1.Tags.Add(tag1);

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

در متد Seed آن یک سری اطلاعات ابتدایی ثبت شده‌اند؛ یک کاربر، یک برچسب و یک مطلب.

SQLQuery6.sql - (I...VahidPC\Vahid (60)) X SQLQuery5.sql - (I...VahidPC\Vahid (59))

```
/****** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [Id]
, [Name]
FROM [testdb_2013].[dbo].[Users]
```

100 %

Results Messages

	Id	Name
1	1	User 1

SQLQuery6.sql - (I...VahidPC\Vahid (60))    SQLQuery5.sql - (I...VahidPC\Vahid (59))    X

```

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [Id]
, [Name]
FROM [testdb_2013].[dbo].[Tags]

```

100 %    Results    Messages

	Id	Name
1	1	Tag1

SQLQuery6.sql - (I...VahidPC\Vahid (60))    SQLQuery5.sql - (I...VahidPC\Vahid (59))    SQLQuery4.sql - (I...VahidPC\Vahid (58))

```

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [Id]
, [Title]
, [Content]
, [UserId]
FROM [testdb_2013].[dbo].[BlogPosts]

```

100 %    Results    Messages

	Id	Title	Content	UserId
1	1	Title...1	Content...1	1

در این تصاویر به Id هر کدام از رکوردها دقت کنید. از آن‌ها در ادامه استفاده خواهیم کرد.  
در اینجا نمونه‌ای از نحوه‌ی استفاده از GraphDiff را جهت به روز رسانی یک Aggregate root ملاحظه می‌کنید:

```

using (var context = new MyContext())
{
    var user1 = new User { Id = 1, Name = "User 1_1_1" };
    var post1 = new BlogPost { Id = 1, Title = "Title...1_1", Content = "Body...1_1",
        User = user1, UserId = user1.Id };
    var tags = new List<Tag>
    {
        new Tag { Id = 1, Name = "Tag1_1"},
        new Tag { Id=12, Name = "Tag2_1"},
        new Tag {Name = "Tag3"},
        new Tag {Name = "Tag4"},
    };
    tags.ForEach(tag => post1.Tags.Add(tag));

    context.UpdateGraph(post1, map => map
        .OwnedEntity(p => p.User)
        .OwnedCollection(p => p.Tags)
    );

    context.SaveChanges();
}

```

پارامتر اول UpdateGraph، گرافی از اشیاء است که قرار است به روز رسانی شوند.  
پارامتر دوم آن، همان مباحث Owned و Associated بحث شده در ابتدای مطلب را مشخص می‌کنند. در اینجا چون می‌خواهیم هم

برچسب‌ها و هم اطلاعات کاربر مطلب اول به روز شوند، نوع رابطه را Owned تعریف کرده‌ایم. در حین کار با متد UpdateGraph، ذکر Idهای اشیاء منقطع از Context بسیار مهم هستند. اگر دستورات فوق را اجرا کنیم به خروجی ذیل خواهیم رسید:

```
SQLQuery6.sql - (I...VahidPC\Vahid (60))  SQLQuery5.sql - (I...VahidPC\Vahid (59))  SQLQuery4.sql - (I...VahidPC\Vahid (58))

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [Id]
, [Name]
FROM [testdb_2013].[dbo].[Users]
```

100 %

Results Messages

	Id	Name
1	1	User 1_1_1

```
SQLQuery6.sql - (I...VahidPC\Vahid (60))  SQLQuery5.sql - (I...VahidPC\Vahid (59))

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [Id]
, [Name]
FROM [testdb_2013].[dbo].[Tags]
```

100 %

Results Messages

	Id	Name
1	1	Tag1_1
2	2	Tag3
3	3	Tag4
4	4	Tag2_1

```
SQLQuery6.sql - (I...VahidPC\Vahid (60))  SQLQuery5.sql - (I...VahidPC\Vahid (59))  SQLQuery4.sql - (I...VahidPC\Vahid (58))

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [Id]
, [Title]
, [Content]
, [UserId]
FROM [testdb_2013].[dbo].[BlogPosts]
```

100 %

Results Messages

	Id	Title	Content	UserId
1	1	Title...1_1	Body...1_1	1

- همانطور که مشخص است، چون id کاربر ذکر شده و همچنین این Id در post1 [نیز درج گردیده است](#)، صرفاً نام او ویرایش گردیده است. اگر یکی از موارد ذکر شده رعایت نشوند، ابتدا کاربر جدیدی ثبت شده و سپس رابطه‌ی مطلب و کاربر به روز رسانی خواهد شد (userId آن به userId آخرین کاربر ثبت شده تنظیم می‌شود).
- در حین ثبت برچسب‌ها، چون Id=1 از پیش در بانک اطلاعاتی موجود بوده، تنها نام آن ویرایش شده‌است. در سایر موارد، برچسب‌های تعریف شده صرفاً اضافه شده‌اند (چون Id مشخصی ندارند یا Id=12 در بانک اطلاعاتی وجود خارجی ندارد).
- چون Id مطلب مشخص شده‌است، فیلدهای عنوان و محتوای آن نیز به صورت خودکار ویرایش شده‌اند.

و ... تمام این کارها صرفاً با فراخوانی متدهای UpdateGraph و سپس SaveChanges رخ داده‌است.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

[GraphDiffTests.zip](#)

## نظرات خوانندگان

نویسنده: مسعود سنائی  
تاریخ: ۲۲:۴۹ ۱۳۹۳/۰۵/۲۰

در مثال فوق چنانچه بخواهیم تنها Title شیء BlogPost را ویرایش کنیم، بایستی ابتدا کل Aggregation را Load کنیم و بعد Title را تغییر دهیم، آیا راهی غیر از این روش وجود دارد؟

نویسنده: وحید نصیری  
تاریخ: ۲۳:۵ ۱۳۹۳/۰۵/۲۰

« [وارد کردن یک شیء به سیستم Tracking](#) » انتهای مطلب.  
کاری هم که GraphDiff انجام می‌دهد انجام همین کار در چند سطح وابسته و مرتبط است به صورت بهینه و خودکار.

نویسنده: محمد بنزاده  
تاریخ: ۱۴:۳۰ ۱۳۹۳/۰۶/۱۹

ضمن تشکر از مطلب خوبتون  
آیا امکانش هست که بدون داشتن AssociatedEntity ها هم از GraphDiff استفاده کرد؟ منظور وقتیست که با نوع Generic کار می‌کنیم نه با یک Entity مشخص

نویسنده: وحید نصیری  
تاریخ: ۱۵:۵۹ ۱۳۹۳/۰۶/۱۹

[اینجا بحث شده](#)

نویسنده: md  
تاریخ: ۰:۹ ۱۳۹۳/۰۶/۲۹

با سلام؛ اگر امکان دارد صرفاً جهت مقایسه، کد آخرین قسمت را بدون استفاده از GraphDiff بنویسید.

نویسنده: وحید نصیری  
تاریخ: ۰:۱۲ ۱۳۹۳/۰۶/۲۹

یک مثال: « [بررسی تفصیلی رابطه Many-to-Many در EF Code first](#) »

نویسنده: صابر فتح الهی  
تاریخ: ۲۲:۴۹ ۱۳۹۳/۱۰/۰۵

با سلام؛ نحوه استفاده‌ی از آن با الگوی کار چطوریه؟

نویسنده: وحید نصیری  
تاریخ: ۲۲:۵۷ ۱۳۹۳/۱۰/۰۵

- در کدهای فوق بجای context از \_uow استفاده خواهید کرد. UpdateGraph در حقیقت یک متد الحاقی است برای آن.  
- این نوع متدهای الحاقی خاص را داخل همان کلاس Context هم می‌شود اضافه کرد (this همان context خواهد بود در این حالت). بعد متد تعریف کننده‌ی آن را به اینترفیس IUnitOfWork اضافه کنید.

نویسنده: صابر فتح الهی  
تاریخ: ۹:۳۳ ۱۳۹۳/۱۰/۰۷

من از این پکیج استفاده کردم در زمان ثبت مقاله به خوبی کار میکنه اما در زمان بروزرسانی وقتی تگی از مقاله کم میشه اون تگ به صورت فیزیکی از دیتابیس حذف میشه حتی وقتی AssociatedCollection انتخاب میکنم با خطا مواجه میشه

```
public void Update(Article entity)
{
    var item = articles.Find(entity.Id);
    item.Author = entity.Author;
    item.CategoryId = entity.CategoryId;
    item.Content = entity.Content;
    item.Source = entity.Source;
    item.Title = entity.Title;
    if (entity.FileStream != null)
        item.FileStream = new File
        {
            ContentType=entity.FileStream.ContentType,
            Id=entity.Id,
            Size = entity.FileStream.Size,
            FileBytes = entity.FileStream.FileBytes
        };
    var allTag = tags.ToCacheableList().ToList();
    var tagsList = entity.Tags.ToList().Select(x => x.Name.ToPersianContent(true)).ToArray();

    if (entity.Tags != null && entity.Tags.Any())
    {
        entity.Tags.Clear();
        entity.Tags = new Collection<Tag>();
    }

    var listOfTags = tagsList.Select(tag =>
        allTag.Any(x => x.Name == tag) ?
        allTag.FirstOrDefault(x => x.Name == tag) :
        new Tag { Name = tag }).ToList();

    listOfTags.ForEach(tag => entity.Tags.Add(tag));

    unitOfWork.PwsUpdateGraph(entity, map => map
        .OwnedEntity(p => p.FileStream)
        .OwnedCollection(p => p.Tags));
}
```

نویسنده:

صابر فتح الهی

تاریخ:

۱۰:۱۲ ۱۳۹۳/۱۰/۰۷

در زمان استفاده از این پکیج زمانی که در کلاس پایه موجودیت فیلدی مزین شده با Timestamp داشته باشیم در زمان بروز رسانی با خطای زیر مواجه می‌شوم.

RowVersion failed optimistic concurrency

نویسنده:

وحید نصیری

تاریخ:

۱۰:۲۳ ۱۳۹۳/۱۰/۰۷

خطاهای دریافتی را در [issue tracker](#) این پروژه به نویسنده‌ی اصلی آن گزارش کنید.