

با توجه به اصل **Dry** تا می توان باید از نوشتن کدهای تکراری خودداری کرد و کدها را تا جایی که ممکن است به قسمت هایی با قابلیت استفاده ی مجدد تبدیل کرد. حین کار کردن با ORM های معروف مثل NHibernate و EntityFramework زمان زیادی نوشتن کوئری ها جهت واکنشی داده ها از دیتابیس صرف می شود. اگر بتوان کوئری هایی با قابلیت استفاده ی مجدد نوشت علاوه بر کاهش زمان توسعه قابلیت هایی قدرتمندی مانند زنجیر کردن کوئری ها به دنبال هم به دست می آید. با یک مثال نحوه ی نوشتن و مزایای کوئری با قابلیت استفاده ی مجدد را بررسی می کنیم :

برای مثال دو جدول شهرها و دانش آموزان را در نظر بگیرید:

```
namespace ReUsableQueries.Model
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
        [ForeignKey("BornInCityId")]
        public virtual City BornInCity { get; set; }
        public int BornInCityId { get; set; }
    }

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

        public virtual ICollection<Student> Students { get; set; }
    }
}
```

در ادامه این کلاس ها را در معرض دید EF Code first قرار داده:

```
using System.Data.Entity;
using ReUsableQueries.Model;

namespace ReUsableQueries.DAL
{
    public class MyContext : DbContext
    {
        public DbSet<City> Cities { get; set; }
        public DbSet<Student> Students { get; set; }
    }
}
```

و همچنین تعدادی رکورد آغازین را نیز به جداول مرتبط اضافه می کنیم:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = true;
        AutomaticMigrationDataLossAllowed = true;
    }
    protected override void Seed(MyContext context)
    {
        var city1 = new City { Name = "city-1" };
        var city2 = new City { Name = "city-2" };
        context.Cities.Add(city1);
        context.Cities.Add(city2);
        var student1 = new Student() { Name = "Shaahin", LastName = "Kiassat", Age=22, BornInCity =
city1};
        var student2 = new Student() { Name = "Mehdi", LastName = "Farzad", Age = 31, BornInCity =
```

```
city1 };
= city2 };
    var student3 = new Student() { Name = "James", LastName = "Hetfield", Age = 49, BornInCity
    context.Students.Add(student1);
    context.Students.Add(student2);
    context.Students.Add(student3);
    base.Seed(context);
    }
}
```

فرض کنید قرار است یک کوئری نوشته شود که در جدول دانش آموزان بر اساس نام ، نام خانوادگی و سن جستجو کند :

```
var context = new MyContext();
var query= context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
    x => x.Age == age);
```

احتمالا هنوز کسانی هستند که فکر می کنند کوئری های LINQ همان لحظه که تعریف می شوند اجرا می شوند [اما اینگونه نیست](#) . در واقع این کوئری فقط یک Expression از رکوردهای جستجو شده است و تا زمانی که متد ToArray یا ToList روی آن اجرا نشود هیچ داده ای برگردانده نمی شود.

در یک برنامه ی واقعی داده های باید به صورت صفحه بندی شده و مرتب شده برگردانده شود پس کوئری به این صورت خواهد بود :

```
var query= context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
    x => x.Age == age).OrderBy(x=>x.LastName).Skip(skip).Take(take);
```

ممکن است بخواهیم در متد دیگری در لیست دانش آموزان بر اساس نام ، نام خانوادگی ، سن و شهر جستجو کنیم و سپس خروجی را اینبار بر اساس سن مرتب کرده و صفحه بندی نکنیم:

```
var query = context.Students.Where(x => x.Name.Contains(name)).Where(x =>
x.LastName.Contains(lastName)).Where(
    (
        x => x.Age == age).Where(x => x.BornInCityId == 1).OrderBy(x => x.Age);
```

همانطور که می بینید قسمت هایی از این کوئری با کوئری هایی که قبلا نوشتیم یکی است ، همچنین حتی ممکن است در قسمت دیگری از برنامه نتیجه ی همین کوئری را به صورت صفحه بندی شده لازم داشته باشیم.

اکنون نوشتن این کوئری ها میان کد های Business Logic باعث شده هیچ استفاده ی مجددی نتوانیم از این کوئری ها داشته باشیم. حال بررسی می کنیم که چگونه می توان کوئری هایی با قابلیت استفاده ی مجدد نوشت :

```
namespace ReUsableQueries.Queries
{
    public static class StudentQueryExtension
    {
        public static IQueryable<Student> FindStudentsByName(this IQueryable<Student> students, string
name)
        {
            return students.Where(x => x.Name.Contains(name));
        }
        public static IQueryable<Student> FindStudentsByLastName(this IQueryable<Student> students,
string lastName)
        {
            return students.Where(x => x.LastName.Contains(lastName));
        }
        public static IQueryable<Student> SkipAndTake(this IQueryable<Student> students, int skip , int
take)
        {
            return students.Skip(skip).Take(take);
        }
        public static IQueryable<Student> OrderByAge(this IQueryable<Student> students)
```

```

    {
        return students.OrderBy(x=>x.Age);
    }
}

```

همان طور که مشاهده می کنید به کمک متدهای الحاقی برای شیء `IQueryable<Student>` چند کوئری نوشته ایم . اکنون در محل استفاده از کوئری ها می توان این کوئری ها را به راحتی به هم زنجیر کرد. همچنین اگر روزی قرار شد منطق یکی از کوئری ها عوض شود با عوض کردن آن در یک قسمت برنامه همه جا اعمال می شود. نحوه ی استفاده از این متدهای الحاقی به این صورت خواهد بود :

```

var query =
context.Students.FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(skip,take);

```

فرض کنید قرار است یک سیستم جستجوی پیشرفته به برنامه اضافه شود که بر اساس شرطهای مختلف باید یک شرط در کوئری اعمال شود یا نشود ، به کمک این طراحی جدید به راحتی می توان بر اساس شرطهای مختلف یک کوئری را اعمال کرد یا نکرد :

```

var query = context.Students.AsQueryable();
if (searchByName)
{
    query= query.FindStudentsByName(name);
}
if (orderByAge)
{
    query = query.OrderByAge();
}
if (paging)
{
    query = query.SkipAndTake(skip, take);
}
return query.ToList();

```

همچنین این کوئری ها وابسته به ORM خاصی نیستند البته این نکته هم مد نظر است که Linq Provider بعضی ORM ها ممکن است بعضی کوئری ها را پشتیبانی نکند.

نظرات خوانندگان

نویسنده: محمد باقر سیف الهی
تاریخ: ۲۲:۲۳ ۱۳۹۱/۰۸/۱۸

ممنون از مطلب خوبتون... می خواستم بدونم اگه بخوام این متدها رو (در کلاس StudentQueryExtension) جوری بنویسم که با Anonymous Type هم قابل استفاده باشه چه راه حلی وجود داره؟ (یعنی تمام ستون ها رو برگردونم و فقط اونهایی رو که نیاز دارم نمایش بدم و این اعلام نیاز بتونه داینامیک باشه و از طریق پارامتر به تابع پاس داده بشه یا چیزی شبیه این!) . نوع خروجی متدها بهتره چجوری نوشته بشن؟

نویسنده: شاهین کیاست
تاریخ: ۲۲:۴۱ ۱۳۹۱/۰۸/۱۸

خواهش می کنم.
با توجه به این که متدهای الحاقی برای

IQueryable<Entity>

نوشته شده اند پس نوع خروجی هم باید از همین نوع باشد ، راه حلی که به نظر می آید اینه که برای برگرداندن چند ستون نوع برگشتی را از نوع یک CustomObject بگذارید مثلا StudentDTO در مورد داینامیک بودن نمی دانم چه کار باید کرد اما برای خودم هم جالب هست که آیا میشه این کار رو کرد یا خیر .

نویسنده: وحید نصیری
تاریخ: ۱:۲۵ ۱۳۹۱/۰۸/۱۹

- هیچ تغییری را در متدهای الحاقی همه منظوره ایجاد نکنید. این متدها رکوردی رو بر نمی گردونند (در متن لینک داده شده). فقط یک سری عبارت هستند. Select نهایی ویژه را پیش از ToList آخر کار انجام بدید.
- برای پویا کردن LINQ امکان استفاده از رشته ها وجود داره: (^)
- نوع خروجی متد در این حالت خاص می تونه object یا IEnumerable خالی باشد.

نویسنده: محسن د
تاریخ: ۱:۴۷ ۱۳۹۱/۰۸/۱۹

اول تشکر می کنم بابت مطلب خوبتون ..
اگر سوال جناب سیف الهی رو درست متوجه شده باشم ، ایشون می خوان که فیلدهایی رو که از یک تابع برگشت داده میشه خودشون انتخاب کنن و محدود به مقدار بازگشتی از نوع Student برای مثال نباشن .
ایده ای که به ذهن من رسید (بر اساس برداشتی که از سوال داشتم) استفاده از قابلیت بسیار کاربردی Func هستش . یک Func با ورودی از نوع Entity و مقدار بازگشتی از نوع anonymous Type . در هنگام فراخوانی هم میشه از نوع dynamic برای دریافت نتیجه استفاده کرد . یک نمونه از پیاده سازی همچین چیزی رو [اینجا](#) قرار دادم .

نویسنده: شاهین کیاست
تاریخ: ۲:۱۸ ۱۳۹۱/۰۸/۱۹

ممنونم.
نمونه کد خیلی خوبی بود تشکر.

نویسنده: ابراهیم
تاریخ: ۱۱:۴۶ ۱۳۹۱/۰۸/۱۹

سلام. ممنون از مطلب خوبتان. می خواستم نظرتان را در رابطه با الگوی [Repository](#) بدانم، به نظر من این الگو با اینکه محبوبیت زیادی هم پیدا کرده ولی به پیچیدگی نالازمی نسبت به روش شما دارد. سوالی نیز داشتم، امکان نداشت به شیوه ای از IEnumerable به جای IQueryable استفاده شود؟ به نظر من مزیت آن در این است که بتوان خارج از چارچوب ORM از این کوئری ها استفاده شود و برای آن ها تست ایجاد نمود.
باز هم ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۲:۱

- مطلب جاری نفی کننده وجود لایه سرویس در برنامه نیست و مکمل آن است.
- پیاده سازی الگوی مخزنی را که لینک دادید اشتباه است. دلایل اشتباه بودن آن را در این مطلب مطالعه کنید: ([^](#))

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۲:۲

سلام ؛ [استفاده از الگوی Repository اضافی در EF Code first؛ آری یا خیر؟!](#)

لطفا مطلب [تفاوت های IQueryable و IEnumerable](#) را مطالعه بفرمایید.
اگر از IEnumerable استفاده شود دیگر نمی توان کوئری ها را به هم زنجیر کرد .

نویسنده: محمد باقر سیف الهی
تاریخ: ۱۳۹۱/۰۸/۲۰ ۹:۴

بسیار ممنون از تمام دوستان...

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۷:۹

در قسمت زنجیر کردن کوئری ها نباید

```
var query =  
context.Students.FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(skip,take);
```

به

```
var query =  
context.Students.AsQueryable().FindStudentsByName(name).FindStudentsByLastName(lastName).SkipAndTake(sk  
ip,take);
```

تغییر کند؟! اگر جواب منفی است چرا؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۸/۲۰ ۱۸:۴۷

نیازی نیست چون DbSet از یک سری کلاس منجمله IQueryable مشتق می شود.

نویسنده: کیا
تاریخ: ۱۳۹۱/۰۸/۲۱ ۹:۱۲

برای حالتی که بخواین بصورت دینامیک و Anonymous ستونها رو پاس بدین می تونین بصورت زیر عمل کنین. در سمت سرویس :

```
public IEnumerable<dynamic> GetCustomColumnsDynamic(Func<Student, dynamic> pColumns)
{
    return _entities.Select(pColumns).ToList();
}
```

و برای استفاده :

```
var resultDynamic = _serviceStudent.GetCustomColumnsDynamic
(
    x=> new { x.Id, x.LastName, x.Age }
);
MessageBox.Show(resultDynamic.LastName);
```

و البته همونطور که می‌دونین چون نتیجه بصورت dynamic در اختیار شما قرار می‌گیره از امکانات کامپایلر بی نصیب هستید

نویسنده: محمد جواد تواضعی
تاریخ: ۱۷:۳۰ ۱۳۹۱/۰۸/۲۹

سلام

شاهین جان بابت مطلب بسیار عالی بود.

می‌خواستم نظرت در مورد اینکه برای گرفتن کوئری با قابلیت مجدد از این روش استفاده بشود چیست ؟ [Expression tree](#)

و برای کوئری با قابلیت مجدد کدام روش بهینه‌تر می‌باشد ؟

نویسنده: کوروش شاهی
تاریخ: ۱۳:۱۷ ۱۳۹۳/۰۲/۲۸

با توجه به مطلبی که در میحث « [تفاوت بین IQueryable و IEnumerable در حین کار با ORMs](#) » بیان شده، خروجی متد یا باید List و یا باید IEnumerable باشد ؟
اگه مثالی هم بیان بشه این مهم بیشتر قابل درک است و یا لینکی که با مثال این رو توضیح داده باشه.
متشکر.

نویسنده: محسن خان
تاریخ: ۱۳:۳۴ ۱۳۹۳/۰۲/۲۸

بستگی داره در چه لایه‌ای کار می‌کنید و این خروجی قراره در چه لایه‌ای استفاده بشه. خروجی لایه سرویس قراره در لایه UI نمایش داده بشه؟ خروجی لایه سرویس نباید IQueryable باشه. داخل لایه سرویس می‌خواهید کوئری‌ها را با هم ترکیب کنید؟ باید IQueryable باشه.

نویسنده: کوروش شاهی
تاریخ: ۱۵:۱۳ ۱۳۹۳/۰۲/۲۸

با توجه به موارد و بستگی هایی که بیان کردین، فقط در لایه سرویس(بیزینس) باید IQueryable بودن یا نبودن خروجی متد رو مشخص کنیم و یا همچنین در لایه Repository یا همون DAL هم باید این موارد رو در نظر بگیریم ؟
با تشکر.

نویسنده: کوروش شاهی
تاریخ: ۱۶:۵۷ ۱۳۹۳/۰۲/۲۹

اگر منبع معتبری هم باشه که این موارد رو در قبال مثال توضیح داده باشه، میتونه خیلی بیشتر مثر ثمر واقع بشه. من خیلی گوگل کردم ولی روش ها بسیار متنوع بود و آدم سردرگم میشه بیشتر. متشکرم.

نویسنده: شاهین کیاست

تاریخ: ۱۷:۳۰ ۱۳۹۳/۰۲/۲۹

من یک دور بازخوردهای شما را خواندم اما متوجه موردی که برای شما ابهام ایجاد کرده نشدم. آیا شما از Entity Framework استفاده می کنید؟ اگر پاسخ مثبت است، خود EF لایه ی Repository را پیاده سازی کرده است، و این پیاده سازی یک IQueryable جهت انجام Queryهای متفاوت در اختیار شما قرار می دهد. شما می توانید مستقیما از DbContext سمت لایه ی سرویس استفاده کنید و داده ها را جهت استفاده برای استفاده کننده ی لایه ی سرویس فراهم کنید. لایه ی سرویس باید داده ها را درون حافظه برگرداند، نه اینکه یک IQueryable برگرداند که استفاده کننده آن را اجرا کند. از Repository در لایه ی سرویس استفاده کنید.

نویسنده: احمدعلی شفیعی

تاریخ: ۱۸:۱۴ ۱۳۹۳/۰۹/۱۲

آیا با این روش ORM هم می فهمه که فقط اون اطلاعات رو باید از دیتابیس بگیره؟

نویسنده: محسن خان

تاریخ: ۱۸:۳۶ ۱۳۹۳/۰۹/۱۲

همیشه میشه خروجی دقیق ORM رو بدون حدس و گمان لاگ و بررسی کرد: [نمایش خروجی SQL کدهای Entity framework 6 در کنسول دیباگ ویژوال استودیو](#)