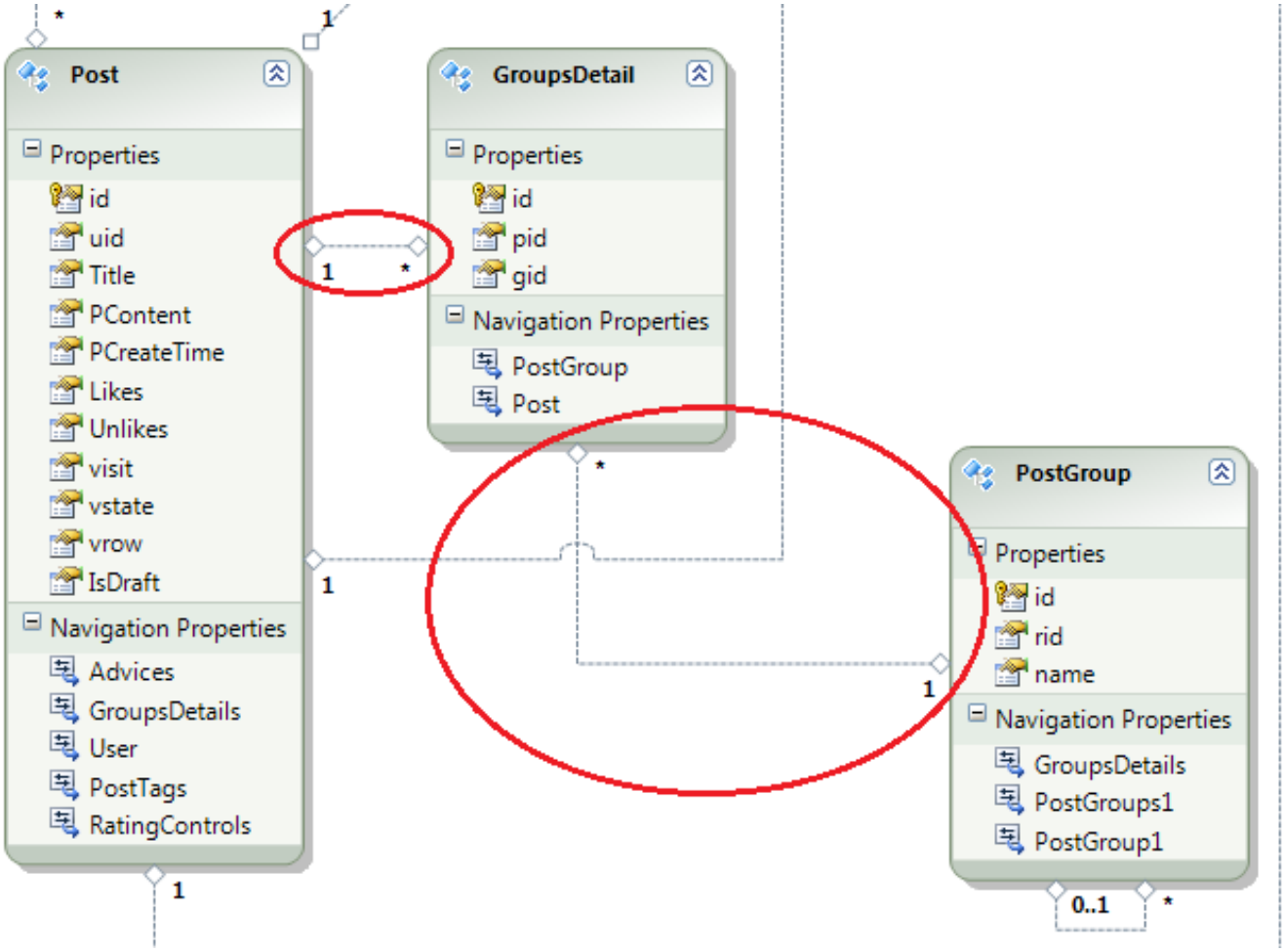
عنوان: **ارسال انواع بی نام (Anonymous) بازگشتی توسط Entity framework به توابع خارجی**
نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۱ ۱۶:۵۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Entity framework, ASP.Net MVC, Anonymous Type

فرض کنید ساختار زیر را در مدل ساخته شده به وسیله‌ی Entity framework در پروژه‌ی خود داریم .



جدول Post با جدول GroupsDetail ارتباط یک به چند و در مقابل ان جدول GroupsDetail با جدول PostGroup ارتباط چند به یک دارد . به زبان ساده ما تعدادی گروه بندی برای مطالب داریم (در جدول PostGroup) و میتوانیم برای هر مطلب تعدادی از گروه‌ها را در جدول GroupsDetail مشخص کنیم ...
فکر میکنم همین مقدار توضیح به اندازه‌ی کافی برای درک روابط مشخص شده در مدل رابطه ای مفروض کفایت کند.
حالا فرض کنید ما میخواهیم لیستی از عنوان مطالب موجود به همراه [نام] گروه‌های هر مطلب را داشته باشیم .
توجه شما رو به قطعه کد ساده‌ی زیر جلب میکنم:

```
var context = new Models.EntitiesConnection();
var query = context.Posts.Select(pst => new {
             id = pst.id,
             Title = pst.Title,
             GNames = pst.GroupsDetails.Select(grd => new { Name = grd.PostGroup.name })
             })
             .OrderByDescending(c => c.id)
             .ToList();
```

همانطور که شاهد هستید در قطعه کد بالا توسط خواص راهبری (Navigation Properties) به صورت مستقیم نام گروه‌های ثبت شده برای هر مطلب در جدول GroupsDetail را از جدول PostGroup استخراج کردیم . خب تا اینجا مسئله ای وجود نداشت (البته با ORMها این کار بسیار سهل و اسان شده است تا جایی که در حالت عادی برای همین کار باید کلی join نویسی کرد و .... در کل خدارو شکر که از دست کارهای تکراری و خسته کننده توسط این موجودات مهربان (ORM's) نجات یافتیم)...

خب میرسیم به ادامه‌ی مطلبمون . نتیجه‌ی این query چه خواهد بود ؟ کاملا واضح است که ما تعداد دلخواهی از فیلدها رو برای واکشی مشخص کردیم پس در نتیجه نوع داده ای که توسط این query بازگشت داده خواهد شد یک لیست از یک نوع بی نام میباشد ... چگونه از نتیجه‌ی بازگشتی این query در صورت ارسال اون به عنوان یک پارامتر به یک تابع استفاده کنیم ؟ اگر ما تمامی فیلدهای جدول Post را واکشی میکریم مقدار بازگشتی یک لیست از نوع Post بود که به راحتی قابل استفاده بود مثل مثال زیر

```
public bool myfunc(List<Post> query)
        {
            foreach (var item in query)
            {
            .....  string title =  item.Title;
            }
...return true;
        }
.
.
.

var context = new Models.EntitiesConnection();
var queryx = context.Posts.ToList();
.... myfunc(queryx );
```

اما حالا با یک لیست از یک نوع بی نام رو به رو هستیم ... چند راه مختلف برای دسترسی به این گونه از مقادیر بازگشتی داریم .
1 : استفاده از Reflection برای دسترسی به فیلدهای مشخص شده .
2 : تعریف یک مدل کامل بر اساس فیلدهای مشخص شده‌ی بازگشتی و ارسال یک لیست از نوع تعریف شده به تابع . (به نظرم این روش خسته کنندست و زیاد شکیل نیست چون برای هر query خاص مجبوریم یک مدل کلی تعریف کنیم و این باعث میشه تعداد زیادی مدل در نهایت داشته باشیم که استفاده‌ی زیادی از اونها نمیشه عملا)
3 : استفاده از یکی از روش‌های خلاقانه‌ی تبدیل نوع Anonymousها .

روش سوم روش مورد علاقه‌ی من هست و انعطاف بالایی داره و خیلی هم شیکو مجلسیه ! در ضمن این روش که قراره ذکر بشه کاربردهای فراوانی داره که این مورد فقط یکی از اونهاست
خب بریم سر اصل مطلب . به کد زیر توجه کنید :

```
public static List<T> CreateGenericListFromAnonymous<T>(object obj, T example)
        {
            return (List<T>)obj;
        }
        public static IEnumerable<T> CreateEmptyAnonymousIEnumerable<T>(T example)
        {
            return new List<T>(); ;
        }

////

        public bool myfunc(object query)
        {
            var cquery = CreateGenericListFromAnonymous(query,
                                new {
                                        id = 0,
                                        Title = string.Empty,
                                        GNames = CreateEmptyAnonymousIEnumerable(new { Name =
string.Empty })
                                });
            foreach (var item in cquery)
            {
```

```
                string title = item.Title;
                foreach (var gname in item.GNames)
                {
                    string gn = gname.Name;
                }
.
.
.
            }
            return false;
        }

.
.
.
var context = new Models.EntitiesConnection();
var query = context.Posts.Select(pst => new
                                {
                                     id = pst.id,
                                     Title = pst.Title,
                                     GNames = pst.GroupsDetails.Select(grd => new { Name =
grd.PostGroup.name })
                                }).OrderByDescending(c => c.id);
            myfunc(query.ToList());
```

در کد بالا به صورت تو در تو از انواع بی نام استفاده شده تا مطلب براتون خوب جا بیوفته . یعنی یک نوع بی نام که یکی از فیلدهای اون یک لیست از یک نوع بی نام دیگست ... خب میبینید که خیلی راحت با دو تابع ما تبدیل انواع بی نام رو به صورت inline انجام دادیم و ازش استفاده کردیم . بدون اینکه نیاز باشه ما یک مدل مجزا ایجاد کنیم ...

تابع CreateGenericListFromAnonymous : یک object رو میگیره و اون رو به یک لیست تبدیل میکنه بر اساس نوعی که به صورت inline براش مشخص میکنیم .

تابع CreateEmptyAnonymousIEnumerable یک لیست از نوع IEnumerable رو بر اساس نوعی که به صورت inline براش مشخص کردیم بر میگردونه . دلیل اینکه من در اینجا این تابع رو نوشتم این بود که ما در query یک فیلد با نام GNames داشتیم که مجموعه ای از نام گروه‌های هر مطلب بود که از نوع IEnumerable هستش . در واقع ما در اینجا نوع بی نامی داریم که یکی از فیلدهای اون یک لیست از یک نوع بی نام دیگست . امیدوارم مطلب براتون جا افتاده باشه.

دوستان عزیز سوالی بود در قسمت نظرات مطرح کنید.

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۳۹۱/۰۹/۲۱ ۱۷:۱۱

با تشکر. روش‌های دیگری هم برای بازگشت انواع بی‌نام و نشان وجود دارند:

- خروجی متد را object تعریف کنیم

- خروجی متد را یک لیست از نوع dynamic (سی شارپ 4) تعریف کنیم

- خروجی متد را فقط ienumerable تعریف کنیم (نیازی به ذکر t ندارد الزاما)

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۱/۰۹/۲۱ ۲۰:۴۷

ممنون از نکاتی که ذکر کردید . از نکات ذکر شده برجسته ترینش استفاده از انواع dynamic هستش که کار رو ساده میکنه ولی خب مشکلاتی رو هم به وجود میاره مثلا اشکال زدایی رو سخت میکنه. هر چند که این نوع کار خودش رو با Reflection در زمان اجرا انجام میده و استفاده از اون رو ساده میکنه ولی خب استفاده مستقیم از روش‌های سطح پایین‌تر مزایایی رو هم داره مثلا مثال زیر رو در نظر بگیرید

```
public bool sample(object query)
        {
            System.Reflection.BindingFlags flags = System.Reflection.BindingFlags.Public |
                System.Reflection.BindingFlags.NonPublic |
                System.Reflection.BindingFlags.Static |
                System.Reflection.BindingFlags.Instance |
                System.Reflection.BindingFlags.DeclaredOnly;
            foreach (var item in query.GetType().GetProperties(flags))
            {

                string test = item.GetValue(query,null).ToString();
                // Do Something ...
            }
            return false;
        }
```

خب حلا فرض کنید ما میخوایم یک Table Generator بسازیم که بر اساس لیست نتایج بازگشتی توسط query بازگشت داده شده توسط Entity framework یک جدول رو برامون ایجاد میکنه . طبیعی هست که هر بار ما یک نوع داده ای بی نام رو براش ارسال میکنیم اگه ما بخوایم نوع dynamic رو به عنوان پارامتر تابع تعریف کنیم نمیتونیم به فیلدهای نوع بی نام دسترسی داشته باشیم چرا که هر بار فیلدها فرق میکنه و این تابع قراره در زمان اجرا فیلدها رو تشخیص بده ولی در انواع dynamic ما نام فیلد رو در زمان طراحی مشخص میکنیم و در زمان اجرا توسط نوع dynamic بسط داده میشه که این موضوع واسه همچین مواردی کارایی نداره ... و باید در نظر داشته که انواع dynamic فقط میتونن به فیلدهای public دسترسی داشته باشن ولی ما با reflection میتونیم محدوده‌ی دسترسی رو مشخص کنیم...
و اما در رابطه با مطلب بالا : بر فرض ما مجموعه ای از داده‌ها رو توسط Entity framework واکشی کردیم و یک نوع بی نام داریم و حالا میخوایم مثلا با Table Generator فرضی دو جدول رو از همین یک بار واکشی ایجاد کنیم که هر کدوم شامل یک سری فیلدها هستن و یک سری فیلدها رو از نوع بی نام واکشی شده شامل نمیشن . ما میتونیم یک تابع داشته باشیم که لیست نوع بی نام مرجع رو براش ارسال کنیم و یک نوع بی نام هم به عنوان یک پارامتر به صورت inline براش بفرستیم که فیلدهای مورد نظرمون رو از نوع بی نام مرجع شامل میشه . حالا با کمی توسعه CreateGenericListFromAnonymous و مپ کردن نوع بی نام مرجع با نوع بی نام ارسال شده توسط پارامتر و استفاده از Reflection میتونیم فقط فیلدهایی رو که به صورت inline مشخص کردیم داشته باشیم و با یک بار واکشی اطلاعات چندین بار اون رو با شکل‌های مختلف پردازش کنیم و .... این فقط یک مثال بود و مطلب بالا صرفا ایده ای بود که دوستان بتونن کارهای خلاقانه ای رو از طریق اون انجام بدن ....

نویسنده: ایلیا
تاریخ: ۲۰:۵۹ ۱۳۹۱/۰۹/۲۱

عالی بود. من همیشه یک مدل کامل ایجاد می‌کردم. ولی حالا خیلی راحت شد. تشکر فراوان.

نویسنده: سجاد
تاریخ: ۱۷:۴۴ ۱۳۹۱/۰۹/۲۲

بسیار عالی ! اگه در مورد " **حالا با کمی توسعه CreateGenericListFromAnonymous و مپ کردن نوع بی نام مرجع با نوع بی نام ارسال شده توسط پارامتر و استفاده از Reflection میتونیم فقط فیلدهایی رو که به صورت inline مشخص کردیم داشته باشیم** " بیشتر توضیح بدین ممنون میشم

نویسنده: سید مهران موسوی
تاریخ: ۲۲:۱۹ ۱۳۹۱/۰۹/۲۲

خواهش میکنم قابل نداشت . دوست عزیز این مطلب واقعا پیچیده و تخصصی هستش من یک نمونه واستون نوشتم که کد رو براتون میزارم ولی برای فهم کاملش نیاز به اشنایی عمقی با ساختار دات نت و کار با رفلکشن داره که توضیحش در چند خط نمیگنجه بحثه یک کتاب کامله. این نمونه کد که نوشتم دقیقا همون چیزی هست که درخواست توضیحش رو دادید .

```
public static List<T> CreateGenericListFromAnonymous<T>(object obj, T example)
        {
            var newquery = new List<T>();
            var constructor = typeof(T).GetConstructors(
            System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance
            ).OrderBy(c => c.GetParameters().Length).First();
            foreach (var item in ((IEnumerable<object>)obj))
            {
                var mapobj = new object[example.GetType().GetProperties().Count()];
                int counter = 0;
                foreach (var itemmap in example.GetType().GetProperties())
                {

                    object value = item.GetType().GetProperty(itemmap.Name).GetValue(item, null);
                    Type t = itemmap.PropertyType;
                    mapobj[counter] = Convert.ChangeType(value, t);
                    counter++;
                }
                newquery.Add((T)constructor.Invoke(mapobj));
            }

            return newquery;
        }


var context = new Models.EntitiesConnection();
var query = context.Posts.Select(pst => new
                                {
                                     id = pst.id,
                                     Title = pst.Title,
                                      Likes = pst.Likes,
                                      Unlikes = pst.Unlikes
                                }).OrderByDescending(c => c.id);

object test_custom_casting = CreateGenericListFromAnonymous(query.ToList(), new { id = 0, Title =
string.Empty });
```

همینطور که میبینید نتیجه‌ی بازگشتی از یک query مرجع یک list شامل **فقط و فقط** فیلدهایی هست که به صورت inline توسط انواع بی نام مشخص شده ! امیدوارم مفید واقع شده باشه . یا حق

نویسنده: ناصر فرجی
تاریخ: ۱۲:۲۳ ۱۳۹۲/۰۱/۱۸

من همیشه از viewmodel استفاده میکنم و فک میکنم روش استانداردتری باشه. ممنون بابت اشتراک گذاری این مطلب.

نویسنده: pjimax
تاریخ: ۱۳۹۲/۰۲/۲۱ ۲۱:۹

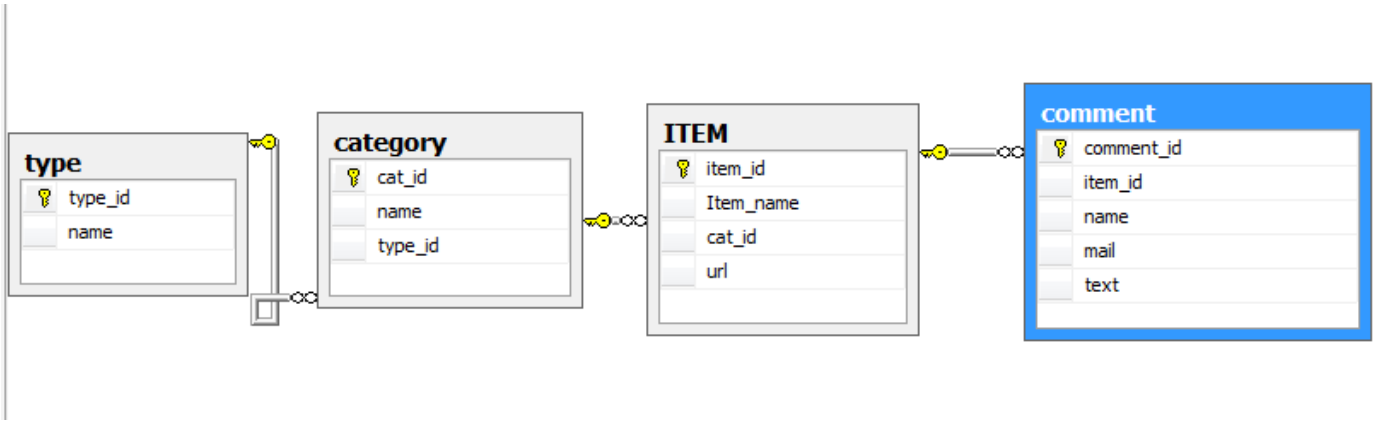
با سلام و تشکر من مقاله شما رو در پروژه خودم پیاده سازی کردم.فقط جای دوتا چهارتا جدول join کردم اما
return(List<T>)obj; خطا میده
لطفا راهنمایی کنید

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۱ ۲۱:۲۱

خطا رو اگر نوشته بودی الان جواب گرفته بودی!

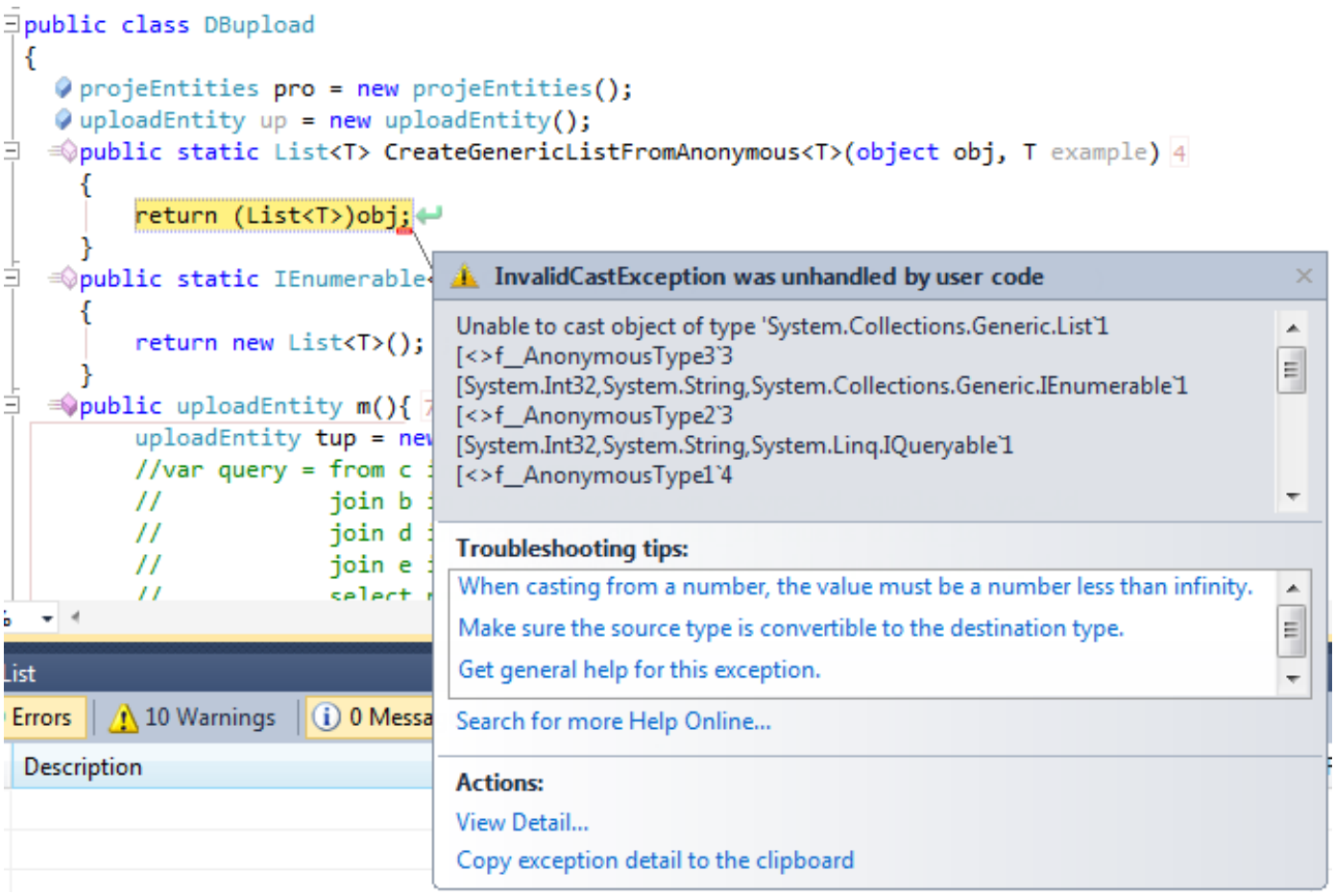
نویسنده: pjimax
تاریخ: ۱۳۹۲/۰۲/۲۲ ۱۰:۵۳

من میخام جداول زیر رو به روش بالا با هم join و استفاده کنم




```
var query = pro.types.Select(ty => new
       {
           typID = ty.type_id,
           typ_nam = ty.type_nam,
           cat_names = ty.categories.Select(cat => new
           {
               catgID = cat.cat_id,
               catg_name = cat.cat_nam,
               items = pro.items.Select(itm => new
               {
                   item_ID = itm.item_id,
                   item_nam = itm.item_nam,
                   item_path = itm.url,
                   coments = pro.comments.Select(cmm => new
                   {
                       comm_title = cmm.comment_nam,
                       comm_mail = cmm.mail,
                       comm_text = cmm.text,
                   })

               })


           })

       }).ToList();

 bool l = myfunc(query);

  protected bool myfunc(object q)
```

```
    {
        var cquery = CreateGenericListFromAnonymous(q,
            new
            {
                typID = 0,
                typ_nam = string.Empty,
                cat_names = CreateEmptyAnonymousIEnumerable(new
                {
                    catgID = 0,
                    catg_name = string.Empty,
                    items = CreateEmptyAnonymousIEnumerable(new
                    {
                        item_ID = 0,
                        item_name = string.Empty,
                        item_path = string.Empty,
                        coments = CreateEmptyAnonymousIEnumerable(new
                        {
                            comm_title = string.Empty,
                            comm_mail = string.Empty,
                            comm_text = string.Empty,

                        })
                    })
                })
            });
        foreach (var item in cquery)
        {
            int ID = item.typID;
            string type_title = item.typ_nam;
            foreach (var Cname in item.cat_names)
            {
                int categoryID = Cname.catgID;
                string category_name = string.Empty;

                foreach (var Iname in Cname.items)
                {
                    int ItmID = Iname.item_ID;
                    string ItmName = Iname.item_name;
                    string ItmPath = Iname.item_path;
                    foreach (var cm in Iname.coments)
                    {
                        string com_title = cm.comm_title;
                        string com_mail = cm.comm_mail;
                        string com_text = cm.comm_text;
                    }
                }
            }
        }
        return true;
    }
```



نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۲ ۱۱:۳۷

```
IList<Types> typesList = context.Types.Include(x=>x.Categories)
                                      .Include(x=>x.Items).Include(x=>x.Comments).ToList();
```

اگر از EF استفاده می‌کنید، استفاده از متد Include کار جوین رو برای شما انجام می‌ده. بعدش نیازی به استفاده از متد CreateGenericListFromAnonymous که فقط یک سطح رو بررسی می‌کنه نیست. برای بررسی بیشتر از یک سطح باید متد بازگشتی نوشته بشه ولی در حالت شما واقعا نیازی نیست. ضمنا متد تصویر شما با متد نوشته شده یکی نیست.