

EF Code first هر بار در حین آغاز اجرای برنامه و اولین کوئری که به بانک اطلاعاتی ارسال می‌کند، کار تشخیص روابط بین کلاس‌ها و همچنین نگاشت آن‌ها را به بانک اطلاعاتی، انجام می‌دهد. این مورد شاید با تعداد کم کلاس‌ها آنچنان به نظر نرسد، اما اگر تعداد کلاس‌های شما به بالای 200 عدد رسید، زمان آغاز برنامه آزار دهنده خواهد شد. راه حلی برای این مساله وجود دارد به نام ایجاد Viewهای متناظر با نگاشت‌ها و سپس کامپایل آن به عنوان جزئی از برنامه، که در ادامه نحوه انجام این کار را مرور خواهیم کرد.

### بررسی ساختار pre-generated views

برای کامپایل نگاشت‌های EF در خود برنامه (بجای تولید پویای هر بار آن‌ها)، ابتدا باید فایل edmx متناظر با مدل‌ها و روابط بین آن‌ها تشکیل شود:

```
var ms = new MemoryStream();
using (var writer = XmlWriter.Create(ms))
{
    EdmxWriter.WriteEdmx(new Context(), writer);
}
```

پس از اینکه edmx تشکیل شد، باید از ساختار فشرده آن سه جزء زیر را استخراج کرد:

ssdl : storageModels (الف)

csdl : conceptualModels (ب)

msl : mappings (ج)

اینکار را به صورت زیر می‌توان انجام داد:

```
var xDoc = XDocument.Load(ms);

var ssdl = xDoc.Descendants("{http://schemas.microsoft.com/ado/2009/02/edm/ssdl}Schema").Single();
var csdl = xDoc.Descendants("{http://schemas.microsoft.com/ado/2008/09/edm}Schema").Single();
var msl =
xDoc.Descendants("{http://schemas.microsoft.com/ado/2008/09/mapping/cs}Mapping").Single();
```

پس از آن باید محتوای این سه جزء را توسط متد Save هر کدام، در فایل‌های xml ایی ذخیره کرد و توسط ابزاری به نام EdmGen.exe که جزئی از ویژوال استودیو است، فایل Context.Views.cs را تولید، به برنامه اضافه و سپس کامپایل کرد:

```
EdmGen.exe /mode:ViewGeneration /incsd1:Context.csdl /inmsl:Context.msl /inssdl:Context.ssdl
/outviews:Context.Views.cs
```

بهتر است این پروسه هر بار که قرار است ارائه نهایی برنامه صورت گیرد، انجام شود.

علاوه بر این‌ها اگر علاقمند باشید که کار فایل EdmGen را شبیه سازی کنید، کلاس زیر این کار را انجام داده و قادر است خروجی vb یا cs متناظری را نیز تولید کند:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Design;
using System.Data.Entity.Infrastructure;
using System.Data.Mapping;
using System.Data.Metadata.Edm;
using System.IO;
using System.Linq;
```

```

using System.Xml;
using System.Xml.Linq;

namespace EfUtils
{
    public static class PreGeneratedViewsWriter
    {
        public static void CreatePreGeneratedViewsFile(
            this DbContext contextInstance,
            LanguageOption language = LanguageOption.GenerateCSharpCode,
            string viewsFile = "Context.Views.cs",
            string edmxFile = "context.edmx",
            string ssdlFile = "context.ssdl.xml",
            string csdlFile = "context.csdl.xml",
            string mslFile = "context.msl.xml")
        {
            using (var contextViewsMemoryStream = new MemoryStream())
            {
                using (var edmxMemoryStream = new MemoryStream())
                {
                    var edmx = createEdmx(contextInstance, edmxFile, edmxMemoryStream);
                    var mappingItemCollection = createMappingItemCollection(ssdlFile, csdlFile,
mslFile, edmx);
                    generateViews(language, viewsFile, contextViewsMemoryStream,
mappingItemCollection);
                }
            }

            private static void generateViews(LanguageOption language, string viewsFile, MemoryStream
contextViewsMemoryStream, StorageMappingItemCollection mappingItemCollection)
            {
                var viewGenerator = new EntityViewGenerator // It's defined in
System.Data.Entity.Design.dll
                {
                    LanguageOption = language
                };
                using (var streamWriter = new StreamWriter(contextViewsMemoryStream))
                {
                    var errors = viewGenerator.GenerateViews(mappingItemCollection, streamWriter).ToList();

                    if (errors.Any())
                        throw new InvalidOperationException(errors.First().Message);

                    contextViewsMemoryStream.Position = 0;
                    using (var reader = new StreamReader(contextViewsMemoryStream))
                    {
                        var codeData = reader.ReadToEnd();
                        File.WriteAllText(viewsFile, codeData);
                    }
                }
            }

            private static StorageMappingItemCollection createMappingItemCollection(string ssdlFile, string
csdlFile, string mslFile, XDocument edmx)
            {
                var ssdl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2009/02/edm/ssdl}Schema").Single();
                ssdl.Save(ssdlFile);
                var storeItemCollection = new StoreItemCollection(new[] { ssdl.CreateReader() });

                var csdl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2008/09/edm}Schema").Single();
                csdl.Save(csdlFile);
                var edmItemCollection = new EdmItemCollection(new[] { csdl.CreateReader() });

                var msl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2008/09/mapping/cs}Mapping").Single();
                msl.Save(mslFile);

                var mappingItemCollection = new StorageMappingItemCollection(edmItemCollection,
storeItemCollection, new[] { msl.CreateReader() });
                return mappingItemCollection;
            }

            private static XDocument createEdmx(DbContext contextInstance, string edmxFile, MemoryStream
edmxMemoryStream)
            {
                var settings = new XmlWriterSettings { Indent = true };
                using (var writer = XmlWriter.Create(edmxMemoryStream, settings))
                {

```

```
        EdmxWriter.WriteEdmx(contextInstance, writer);
    }
    File.WriteAllBytes(edmxFile, edmxMemoryStream.ToArray());

    edmxMemoryStream.Position = 0;
    var edmx = XDocument.Load(edmxMemoryStream);
    return edmx;
}
}
```

در اینجا همان مراحل که عنوان شد، تکرار می‌شود. فایل edmx متناظر با وهله‌ای از DbContext برنامه، تولید شده و سه جزء آن استخراج می‌شوند. سپس این موارد به EntityViewGenerator موجود در اسمبلی System.Data.Entity.Design.dll ارسال شده و کد نهایی متناظر قابل کامپایل در برنامه تولید می‌گردد. پس از تولید فایل Context.Views.cs یا Context.Views.vb، آن‌را به پروژه اضافه کنید. اینبار نحوه استفاده از آن باید به صورت زیر باشد:

```
Database.SetInitializer<MyContext>(null);
```

از این جهت که تمام اطلاعات لازم جهت آغاز کار، در فایل تولیدی Context.Views وجود دارد و اکنون جزئی از فایل اجرایی برنامه است و نیازی به تکرار ساخت مجدد پویای آن نیست.

مرجع:

[Entity Framework Code First View Generation Templates On Visual Studio Code Gallery](#)

## نظرات خوانندگان

نویسنده: مهدی پایروند  
تاریخ: ۱۳۹۱/۰۷/۰۹ ۲۰:۳۸

در یکی از پروژه‌هایی که بجای تعداد 200 جدول رابطه زیادی هم داشت مجبور به استفاده از این روش شدم، البته سبک Model First

نویسنده: Petek  
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۱:۴۶

با سلام  
آیا امکان این هست که بشه از این در روش DataBase First استفاده کرد . با تشکر

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۲:۰۰

بله. همین روال رو می‌تونید توسط افزونه « [Entity Framework Power Tools](#) » هم انجام بدید (گزینه Generate Views را به منوی کلیک راست بر روی Entity Data Model/\*.EDMX اضافه می‌کند ).

نویسنده: Petek  
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۷:۵۰

با سلام  
ممنوم از راهنماییتون خیلی عالیه .  
با استفاده از این کار به view در کنار Model ام ایجاد شد و سرعت بارگذاری برای این اولین بار رو از 10 ثانیه به کمتر از 2 ثانیه تقلیل داد آیا چیز دیگه ای نیاز نیست و نیز من برای ایجاد دیتابیس با استفاده از Model در اولین واکشی به این صورت عمل می‌کنم آیا مشکلی پیش نیاید در استفاده از این روش . با تشکر

```
if (!db_Context.DatabaseExists())  
    db_Context.CreateDatabase();
```

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۸:۵۵

- خیر. به تنظیم دیگری نیاز ندارد. این کلاس‌ها به صورت خودکار تشخیص داده شده و استفاده می‌شوند. البته به ازای هربار تغییر مدل‌ها نیاز است مجدداً تولید شوند.  
- اگر روش شما db first است که عنوان کردید، بررسی فوق (ایجاد بانک اطلاعاتی) کار اضافی است. اگر روش code first است، باز هم نیازی نیست چون در حالت خودکار migrations اینکار را انجام می‌دهد.  
در کل بهتر است تمام جوانب را بررسی و آزمایش کنید.  
کاری که در اینجا انجام می‌شود ایجاد یک [cached metadata](#) کامپایل شده است بجای تولید پویای هربار آن (تفاوت مهم و اصلی با روش‌های متداول).

نویسنده: امیر  
تاریخ: ۱۳۹۱/۱۱/۲۸ ۰۵:۵۲

سلام و خسته نباشید  
سوالی که من دارم اینکه بعد از ساختن فایل Context.Views.cs چطور باید ازش استفاده کرد . من این فایلو ساختم و کنار Context.cs تو پروژه‌ام گذاشتم . ولی فرق خاصی احساس نمی‌کنم . میشه بیشتر راهنمایی کنید

نویسنده: محسن  
تاریخ: ۱۳۹۱/۱۱/۲۸ ۰:۵۸

اگر فرقی احساس نکردید منتظر نگارش بعدی EF باشید. [این مورد رو](#) برطرف کردن:

«significantly improved warm up time (view generation), especially for large models»

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۰۷/۲۷ ۱:۰۶

[به روز شده این اطلاعات برای EF 6](#)  
[اطلاعات بیشتر](#)

نویسنده: سوين  
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۱:۵۰

آیا بعد از ایجاد View برای بالا بردن لود اولیه ، باز هم migration کار میکنه یا نه ؟

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۲:۱۱

بله. هم روش دستی migrations و هم روش خودکار در اینجا کار می‌کنند. اینکه در انتهای بحث عنوان شده مقدار را null قرار بدید، برای رسیدن به حداکثر سرعت بارگذاری برنامه است. در این حالت می‌توانید از روش دستی برای انجام migrations استفاده کنید. این نکته در انتهای [مطلب پنجم](#) سری EF سایت بحث شده.

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۰:۵۵

اگر علاقمند باشید که مدیریت تولید این View ها را خودکار کنید می‌توانید از پروژه Interactive Pre Generated Views استفاده نمائید:

پروژه: [Interactive Pre Generated Views for Entity Framework 6](#)  
[بسته نیوگت](#)

نحوه استفاده: [Using Pre-Generated Views Without Having To Pre-Generate Views](#)

نویسنده: امین  
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۸:۱۵

باسلام. این کد را در کجا و چطوری در مدل code first استفاده بکنیم؟

```
using (var ctx = new MyContext())
{
    InteractiveViews
        .SetViewCacheFactory(
            ctx,
            new FileViewCacheFactory(@"C:\MyViews.xml"));
}
```

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۸:۲۶

در آغاز برنامه (مثلا در application\_start برنامه‌های وب و یا ابتدای متد Main برنامه‌های دسکتاپ): [وادر کردن EF Code](#)

## [first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه](#)»

نویسنده: دانش پژوه  
تاریخ: ۱۴:۲۴ ۱۳۹۳/۰۹/۲۹

با سلام

میشه خواهش کنم یک نمونه پروژه برای استفاده از این روش بزارید.

نویسنده: وحید نصیری  
تاریخ: ۱۴:۳۳ ۱۳۹۳/۰۹/۲۹

نیازی به مثال آنچنانی ندارد. ابتدا بسته‌ی نیوگت [این پروژه](#) را نصب کنید:

```
PM> Install-Package EFInteractiveViews
```

بعد یکبار در ابتدای برنامه در اولین کوئری، متد `InteractiveViews.SetViewCacheFactory` آن را فراخوانی کنید. البته بهتر است آن را درون یک [سینگلتون thread safe](#) قرار دهید. بار اولی که فایل `xml` آن ایجاد می‌شود زمان خواهد برد. بار دوم اجرای برنامه سریع است.

```
private static bool _isPreGeneratedViewCacheSet;
private void InitializationPreGeneratedViews()
{
    if (_isPreGeneratedViewCacheSet) return;

    var precompiledViewsFilePath = new FileInfo(Assembly.GetExecutingAssembly().Location).DirectoryName
+ @"EF6PrecompiledViews.xml";
    InteractiveViews.SetViewCacheFactory(this, new FileViewCacheFactory(precompiledViewsFilePath));
    _isPreGeneratedViewCacheSet = true;
}
```

نویسنده: دانش پژوه  
تاریخ: ۱۰:۳۶ ۱۳۹۳/۱۰/۰۲

با سلام و ممنون از جوابتون

روشی رو گفتید رفتم اینجای بزارم هم دیگران استفاده کنند و اگه هم اشتباه کردم بفرمایید اصلاح کنم.  
پکیج رو تو پروژه ای که کلاس `context` هست نصب کردم و تابع زیر رو

```
private static bool _isPreGeneratedViewCacheSet;
private void InitializationPreGeneratedViews()
{
    if (_isPreGeneratedViewCacheSet) return;

    var precompiledViewsFilePath = new FileInfo(Assembly.GetExecutingAssembly().Location).DirectoryName
+ @"EF6PrecompiledViews.xml";
    InteractiveViews.SetViewCacheFactory(this, new FileViewCacheFactory(precompiledViewsFilePath));
    _isPreGeneratedViewCacheSet = true;
}
```

توی کلاس `context` گذاشتم بعد از اجرای یک فایل `Xml` در مسیر

C:\Users\Hadi\AppData\Local\Temp\Temporary ASP.NET  
Files\root\2781dacc\5d62fdaf\assembly\d13\142eef19\00077ffc\_731ed001

میسازه، البته من بصورت دستی این تابع رو یک بار اجرا کردم و بعد غیرفعالش کردم.  
بعد این تابع رو در `application_start` نوشتم:

```
InteractiveViews
```

```
.SetViewCacheFactory(ctx, new FileViewCacheFactory(new  
FileInfo(Assembly.GetExecutingAssembly().Location).DirectoryName + @"\EF6PrecompiledViews.xml"));
```

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۳/۱۰/۰۲ ۱۰:۳۹

- این فایل باید به ازای هربار تغییر در مدل‌ها دوباره ساخته شود. بنابراین تولید دوباره آن نباید غیرفعال شود.  
- Assembly.GetExecutingAssembly().Location برای برنامه‌های دسکتاپ مفید است (یعنی مسیر فایل اجرایی). در برنامه‌های وب، به این مسیر عموماً دسترسی ندارید. به همین جهت بهتر است از HttpRuntime.AppDomainAppPath استفاده کنید.

نویسنده: مهدی بهزاد  
تاریخ: ۱۳۹۴/۰۲/۱۴ ۲۱:۲۲

سلام. آیا برای بهبود سرعت در EF Designer database Model در WPF هست؟  
در این روش برای بار اول وقتی با دیتابیس برخورد میکنه حدود 5 ثانیه طول میکشه از دفعات بعد سرعت عالی میشه ، لطفا راهی بهم پیشنهاد بدین

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۴/۰۲/۱۵ ۱:۳۰

- نگارش‌های جدید EF بر اساس روش Code first کار می‌کنند (حتی اگر به ظاهر DB First باشند). بنابراین روش تکمیلی مطرح شده [در نظرات](#) ، با تمام نگارش‌های EF کار می‌کند.  
- همچنین علت زمانبر بودن را می‌توانید با [DNT Profiler](#) بررسی کنید که چه مراحل در پشت صحنه انجام می‌شوند. این مراحل، بار اول در بانک اطلاعاتی پردازش و plan آن‌ها کش خواهند شد. به همین جهت برای بار دوم سریعتر است.

نویسنده: مهدی مقدسی  
تاریخ: ۱۳۹۴/۰۴/۲۰ ۱۸:۴۶

بخشید منظورتون از ورژن چند به بعد که مثل هم رفتار میشه؟  
سوال دوم این هست که ما وقتی از روش DatabaseFrist استفاده کنم در Entity 6.1.3 تمامی کلاس‌ها از جمله DbContext با کوچکترین تغییر دوباره ایجاد میشه و تابعی که قراره داخل اون قرار بدماز بین می‌ره. چجوری باید اونو رفع کرد؟

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۴/۰۴/۲۰ ۱۹:۴۴

«... تابعی که قراره داخل اون قرار بدم ...»  
نظرات را مطالعه کردید یکبار؟ مطلب نوشته شده با یک افزونه‌ی جدید جایگزین شده‌است. این مورد هم نیازی به دستکاری Context شما ندارد و هیچ کدی را نیازی نیست، داخل آن قرار دهید.