مرجع کامل

Entity Framework 4.1

مهندس بهروز راد

انتشارات پندار پارس

سرشناسه : راد، بهروز، ۱۳۶۴ - ، گردآورنده، مترجم
عنوان و نام پدیدآور : مرجع کامل Entity Framework 4.1/ ترجمه و تالیف بهروز راد.
مشخصات نشر : تهران: پندار پارس: مانلی، ۱۳۹۰.
مشخصات ظاهری : ۲۷۲ ص.: مصور،جدول.
شابک : 978-964-2989-76-8 : ۷۸۰۰۰ ریال
وضعیت فهرست نویسی : فیپا
موضوع : مایکروسافت دات نت
موضوع : مایکروسافت دات‌نت فریم‌ورک
موضوع : پایگاه‌های اطلاعاتی -- طراحی
رده بندی کنگره : QA۷۶/۷۶ /م۱۳۹۰۲۳ ۲ر
رده بندی دیویی : ۳/۰۰۵
شماره کتابشناسی ملی : ۲۳۹۱۳۲۰

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره ۱٤، واحد ۱٦ **www.pendarepars.com**

تلفن: ٦٦٥٧٢٣٣٥ - تلفکس: ٦٦٩٢٦٥٧٨ همراه: ۰۹۱۲۲٤٥۲۳٤۸ **info@pendarepars.com**

نام کتاب : مرجع کامل Entity Framework 4.1
ناشر : انتشارات پندار پارس ناشر همکار: مانلی
ترجمه و تالیف : مهندس بهروز راد
چاپ اول : تابستان ۹۰
شمارگان : ۱۰۰۰ نسخه
طرح جلد : محمد اسماعیلی هدی
لیتوگرافی، چاپ، صحافی : ترام‌سنج، صالحان، نوین برتر
قیمت : ۷۸۰۰ تومان به همراه CD شابک : ۸-۷٦-۲۹۸۹-۹٦٤-۹۷۸

فهرست

پیش‌گفتار

اگر به میزان کدی که باید برای انجام اعمال مختلف بر روی داده‌های پایگاه داده نوشت دقت کنید، خواهید دید که بسیاری از اعمال و کدها تکراری هستند و این موجب اتلاف در هزینه و وقت توسعه-گران خواهد شد. علاوه بر آن، نبود یک روش یکپارچه موجب می‌شود تا گسترش، ترمیم، بهبود و نگهداری برنامه‌ها با مشکل مواجه شود.

از گذشته روش‌های مختلفی برای کار با داده‌ها معرفی شدند که هر یک سعی در تسهیل کار توسعه-گران داشتند اما در این بین همیشه جای خالی یک مدل مفهومی که توسعه‌گران به جای اینکه به طور مستقیم با پایگاه داده کار کنند با آن سر و کار داشته باشند احساس می‌شد. با افزایش قابلیت‌های زبان‌های برنامه‌نویسی و مشخص شدن ارزش بیش از پیش برنامه‌نویسی شیءگرا، ایجاد روشی برای یکپارچه کردن و ساده‌سازی روش‌های دسترسی به داده، همراه با بهره بردن از مفاهیم شیءگرایی در آن مورد توجه قرار گرفت. اولین تلاش شرکت مایکروسافت در این زمینه، معرفی LINQ to SQL و سپس Entity Framework بود. Entity Framework به طور جدی به عنوان بستر اصلی مایکروسافت به منظور دسترسی به داده‌ها مورد توجه است و با وجود آن نیاز به استفاده از روش‌های قدیمی همچون استفاده‌ی مستقیم از ADO.NET بسیار کمرنگ شده است.

در این کتاب سعی شده است تا توسعه‌گران نرم‌افزار با تمامی اَبعاد و قابلیت‌های Entity Framework آشنا شوند.

از آنجا که هیچ نوشته‌ای خالی از اِشکال و بحث نیست، خوانندگان گرامی می‌توانند با پست الکترونیکی behrouz.rad@gmail.com با اینجانب تماس داشته باشند.

بهروز راد

فصل ۱

آشنایی با Entity Framework

در ماه جولای سال ۲۰۰۸ میلادی، اولین نسخه از Entity Framework به عنوان بخشی از Visual Studio 2008 Service Pack 1 و .NET Framework 3.5 Service Pack 1 منتشر شد. در آن موقع، از عرضه‌ی LINQ و LINQ to SQL که مورد توجه بسیاری قرار گرفته بودند مدتی می‌گذشت.

مایکروسافت با معرفی Entity Framework و LINQ to SQL نشان داد که توجه ویژه‌ای به افزایش بهره‌وری برنامه‌نویسان دارد و این توجه را با فراهم کردن بستری برای مدیریت داده‌ها به صورت شیء فراهم کرده است. در این حالت می‌توان به‌جای برنامه‌نویسی مستقیم برای جداول موجود در پایگاه داده، با برنامه‌نویسی بر روی یک مدل مفهومی که از روی ارتباط میان جداول موجود در یک پایگاه داده استخراج شده است به نتیجه‌ی دلخواه رسید.

Relational Data اصطلاحی است که به موجودیت‌هایی اطلاق می‌شود که با یکدیگر در ارتباط هستند. در دنیای واقعی، بیشتر بدین گونه است. به عنوان مثال، "خبر" یک موجودیت است که در یک گروه مانند "گروه ورزشی" طبقه‌بندی می‌گردد. پس دو موجودیت با عناوین "خبر" و "گروه خبر" داریم که با یکدیگر در ارتباط هستند.

هر چند مایکروسافت تمامی تلاش خود را در جهت جلب توجه برنامه‌نویسان به سمت Entity Framework به کار بست، اما این تلاش در ابتدا به دلیل محبوبیت LINQ to SQL و عدم درک صحیح از Entity Framework موفقیت زیادی به‌دست نیاورد.

این کتاب، دو هدف را دنبال می‌کند:

۱. پاسخ به این پرسش که Entity Framework چیست و چرا مایکروسافت انرژی و منابع بسیار زیادی را برای توسعه، پیشرفت و مقبولیت آن صرف می‌کند.

۲. آشنایی با تمامی قابلیت‌های Entity Framework 4.0 & 4.1 که نسخه‌ی 4.0 آن همراه با Visual Studio 2010 عرضه شد.

این کتاب برای دو دسته از برنامه‌نویسان ارائه شده است:

۱. افرادی که پیش‌تر با Entity Framework کار نکرده‌اند.

۲. افرادی که با Entity Framework کار کرده‌اند و دوست دارند در مورد قابلیت‌های جدید آن در نسخه‌ی 4.0 و 4.1 بدانند.

هدف از ارائه‌ی Entity Framework، معطوف‌سازی بیشتر توجه برنامه‌نویس به قسمت‌های دیگر برنامه به جز لایه‌ی دسترسی به داده (Data Access) بوده است. البته این بدان معنا نیست که Entity Framework جایگزینی برای ADO.NET است؛ بلکه ارتقایی برای آن است و هدف این است که برنامه‌نویس به جای آنکه با داده‌های خام کار کند، با یک مدل مفهومی (Conseptual Model) و مفاهیم آشنایی همچون Class و Property سر و کار داشته باشد.

نیاز به Entity Framework

برای آنکه متوجه شویم که Entity Framework واقعاً چیست و چرا مهم است، نیاز است تا کمی به عقب بازگشته و برای دسترسی به داده‌ها، نگاهی به برخی از تکنولوژی‌های موجود اندازیم.

مایکروسافت در سال‌های گذشته، زمان و تلاش زیادی را برای توسعه‌ی ADO.NET صرف کرد. پیش از ADO.NET از تکنولوژی RDO و پیش از آن نیز از DAO استفاده می‌شد. با معرفی ADO.NET، توسعه‌گران احساس کردند که مایکروسافت در نهایت در تکنولوژی‌های دسترسی به داده، به ثباتی رسیده است.

با افزایش قابلیت‌های ADO.NET که در هر نسخه از .NET. شاهد آن بودیم، ADO.NET یک تکنولوژی رو به جلو برای دسترسی به داده‌ها محسوب می‌شد. کلاس‌هایی مانند DataReader و DataSet تا چندین سال، انتخاب اصلی توسعه‌گران برای دسترسی به داده‌ها بودند.

با تمامی تلاش‌هایی که برای افزایش قابلیت‌های ADO.NET صورت می‌گرفت، شکافی همچنان بین برنامه و پایگاه داده وجود داشت. اگر تغییراتی در پایگاه داده ایجاد می‌شد، زمان زیادی باید صرف تطبیق کدهای برنامه با تغییرات جدید می‌شد.

به تکه کدی که در ادامه می‌بینید دقت کنید. این کد بر روی پایگاه داده‌ی AdventureWorks کار می‌کند. این پایگاه داده را می‌توان از آدرس http://msftdbprodsamples.codeplex.com دانلود کرد.

پس از مشاهده‌ی کد، به دو سوال ذیل پاسخ دهید:

۱. آیا این کد کامپایل می‌شود؟

۲. اگر کامپایل شود، آیا پیغام "yes, we have rows" را خواهید دید؟

```
try
{
string connectionString = Class1.GetConnectionString();
using (SqlConnection conn = new SqlConnection(connectionString))
    {
conn.Open();
using (SqlCommand cmd = new SqlCommand())
        {
            cmd.Connection = conn;
            cmd.CommandText = "SELECT FirstName, MidName, LastName
FROM Person.Contact WHERE ContactID = @ContactID";
            SqlParameter param = new SqlParameter("@ContactID",
SqlDbType.Int, 50, "ContactID");
            param.Value = 8;
cmd.Parameters.Add(param);
            SqlDataReader rdr = cmd.ExecuteReader();
while (rdr.Read())
            {
Console.WriteLine(String.Format("{0}, {1}",
rdr[0], rdr[1]));
            }
        }
    }
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
```

اگر کدهای فوق را در روال Click یک دکمه بنویسید، پروژه بدون خطا کامپایل و فرم نمایش داده می‌شود اما اگر بر روی دکمه کلیک کنید، پیغامی مبنی بر اینکه فیلدی با عنوان MidName وجود ندارد مشاهده خواهید کرد. در حقیقت نام فیلد، MiddleName است اما در برنامه اشتباه وارد شده است. مشکل اینجا است که چنین خطایی را تا زمانی که کد مربوط به فراخوانی آن اجرا نشود مشاهده نخواهید کرد. به نظر شما بهتر نیست که چنین خطاهایی در زمان کامپایل پروژه گزارش شوند و نه در زمان اجرای آن؟ قطعاً پاسخ شما مثبت خواهد بود، به‌ویژه در زمانی که پروژه مُدام

در حال توسعه و بزرگ‌تر شدن است. برنامه‌نویسان، زمان زیادی را صرف مشکلاتی که از جانب پایگاه داده وجود دارد می‌کنند. صرف چنین زمان و هزینه‌ای نیاز نیست و یک برنامه‌نویس باید تمرکز خود را معطوف به منطق برنامه کند و نباید نگران تغییری در جداول، اجزای پایگاه داده و ارتباط بین جداول باشد.

در کد قبل، خطایی که دریافت شد مثالی از ایجاد تغییرات در جدول، بدون اطلاع برنامه‌نویس بود. زمانی که آن جدول ایجاد شده بود، نام MidName برای فیلد مورد نظر منطقی به نظر می‌رسیده است اما پس از مدتی فردی که مسئول پایگاه داده است تصمیم می‌گیرد تا برای زیباتر شدن نام فیلدها، نام فیلد را به MiddleName تغییر دهد. تصور کنید که این تغییر بدون اطلاع برنامه‌نویس باشد. چه اتفاقی رخ خواهد داد؟

نیازی که در اینجا احساس می‌شود، یکپارچگی پایگاه داده، برنامه و داده‌ها است؛ و این دقیقاً چیزی است که Entity Framework برای ما به ارمغان می‌آورد. Entity Framework با ارائه‌ی یک مدل مفهومی که با پایگاه داده و برنامه کار می‌کند، شکاف میان داده‌ها و برنامه را که به برنامه‌نویس در حالتی که با کلاس‌هایی همچون DataReader و DataSet کار می‌کند تحمیل می‌شود از بین می‌برد.

این تلاش پیش‌تر صورت گرفته است!

تکنیک مدل کردن موجودیت‌ها پیش‌تر وجود داشته است اما نه در مقیاسی که Entity Framework از آن پشتیبانی می‌کند. در لیست ذیل برخی از روش‌های مدل کردن برنامه‌ها و عمده تمرکز آنها را ملاحظه می‌کنید:

۱. **ERM (Entity Relationship Model)**: با پایگاه‌های داده استفاده می‌شود و روشی برای نمایش ارتباط‌های منطقی میان موجودیت‌های پایگاه داده به منظور ایجاد یک پایگاه داده است.
۲. **UML (Unified Modeling Language)**: یک زبان استاندارد مدل کردن است که برای توصیف موجودیت‌ها و ارتباط میان آنها استفاده می‌شود.
۳. **ORM (Object-Relational Mapping)**: روشی است برای تطبیق و تناظر میان اجزای پایگاه داده با زبان‌های برنامه‌نویسی شیءگرا.
٤. **DFD (Data Flow Diagram)**: یک نمودار گرافیکی از نحوه‌ی جریان داده‌ها بین پروسس‌ها و سیستم‌هاست.

مشکل اینجاست که هر یک از این روش‌های مدل کردن، در حوزه‌ی خود محدود است. به عنوان مثال، در ابزارهایی که از روش ERM استفاده می‌کنند، ارتباط منطقی میان موجودیت‌ها به خوبی پیاده‌سازی می‌شود اما قابلیت‌های روش UML را ندارند. یا روش UML در توصیف دقیق اشیا و موجودیت‌ها برتری دارد اما قابلیت‌های روش ERM را ندارد.

و البته نباید فراموش کنیم که ابزارهای خوبی نیز که از روش ERM برای مدل کردن استفاده می‌کنند وجود دارند و قابلیت‌های Entity Framework را ارائه می‌دهند. دو ابزار خوب شامل:

۱. **NHibernate**: از برادر بزرگتر خود یعنی Hibernate و زبان برنامه‌نویسی Java به .NET. منتقل شده است.
۲. **SPRING.Net**: یک فریم‌ورک متن باز است که از برادر بزرگتر خود یعنی Spring از زبان برنامه‌نویسی Java به .NET. منتقل شده است.

هر چند لیست قبل کامل نیست اما لزوم و ضرورت استفاده از ابزارهایی که از روش ERM برای تولید نرم‌افزار استفاده می‌کنند و پیچیدگی‌های کار با پایگاه داده را به منظور افزایش بهره‌وری مخفی می‌کنند نشان می‌دهد.

Entity Framework محدودیت‌های ابزارهای دیگری که برای مدل‌کردن استفاده می‌شود را ندارد. این عدم محدودیت از آنجا ناشی می‌شود که Entity Framework در یک سطح مفهومی که بر مبنای ERM است کار می‌کند و در نتیجه، کاربردی عمیق و غنی را فراهم می‌سازد که حتی بسیاری از ابزارهایی که از روش‌های ERM و UML استفاده می‌کنند فاقد آن هستند.

همان‌گونه که از نام آن نیز برمی‌آید، Entity Framework شما را قادر می‌سازد تا بدون آنکه درگیر DataReader و DataSet شوید با موجودیت‌هایی که نمایانگر اجزای پایگاه داده‌تان است به طور مستقیم کار کنید. برخی از برنامه‌نویسانی که پیش‌تر با Entity Framework کار کرده‌اند یا دانش و مطالعه‌ای در مورد آن داشته‌اند، آن را با ابزارهایی که از روش ORM استفاده می‌کنند مقایسه و در نتیجه Entity Framework را یک ORM می‌دانند. این مقایسه به طور کامل صحیح نیست. اگر چه Entity Framework قابلیت‌های روش ORM را دارد اما قابلیت‌هایی به مراتب بیشتر از یک ORM صِرف را ارائه می‌دهد و این قابلیت‌ها به شکلی متفاوت از ابزارهای چنین خانواده‌ای پیاده‌سازی می‌شوند.

Entity Framework چیست؟

Entity Framework مجموعه‌ای از تکنولوژی‌هایی در ADO.NET است که به از بین بردن شکاف میان توسعه‌ی شیءگرا و پایگاه‌های داده کمک می‌کنند. تلاش‌های مختلفی برای از بین بردن این شکاف صورت گرفته که از متداول‌ترین آنها می‌توان به تناظر بین جداول و کلاس‌ها از یک طرف، و فیلدهای جداول و Propertyهای موجود در کلاس متناظر آن از طرف دیگر اشاره کرد. به عبارت دیگر، به ازای هر جدول، یک کلاس وجود دارد و به ازای هر فیلد جدول نیز یک Property در کلاس مربوط به آن جدول ایجاد می‌گردد.

برای روشن‌تر شدن مسئله اجازه دهید تا مثالی را با دو موجودیت "مشتری" (Customer) و "محصول" (Product) بررسی کنیم. هر محصول توسط یک مشتری می‌تواند خریداری شود، پس در یک سناریوی ساده، یک کلاس با عنوان Product خواهیم داشت که یک Property از نوع کلاس Customer دارد. یک مشتری نیز می‌تواند تعداد نامحدودی محصول خریداری کند؛ پس در کلاس Customer یک Property خواهیم داشت که فهرستی از محصولات خریداری شده توسط مشتری را نگهداری می‌کند. بسیار خوب، حال به چه شکل می‌خواهید ارتباط بین این دو کلاس را با مفاهیم ارتباط بین جداول در یک پایگاه داده تطبیق دهید؟ به عنوان مثال، آن Property که در کلاس Customer لیستی از محصولات خریداری شده توسط مشتری را نگهداری می‌کند، فقط در یک زبان برنامه‌نویسی شیءگرا مفهوم پیدا می‌کند و انتظار وجود یک فیلد متناظر برای آن در جدول Customer در پایگاه داده کاملاً نامربوط است. در اینجا است که "مدل مفهومی" معنا پیدا می‌کند و باعث ایجاد یک سطح انتزاعی از جداول موجود در پایگاه داده می‌شود. این سطح مفهومی به برنامه-نویس اجازه می‌دهد تا برای موجودیت‌ها و ارتباطات بین آنها در مدل مفهومی، کوئری بنویسد و نه به طور مستقیم بر روی پایگاه داده. وظیفه‌ی تبدیل این کوئری‌ها که اغلب با LINQ نوشته می‌شوند به دستوراتی که برای پایگاه داده قابل فهم باشند بر عهده‌ی Entity Framework است. بدین ترتیب، فاصله‌ی بین برنامه‌نویسی شیءگرا و پایگاه داده از بین می‌رود و برنامه‌نویسان عمده تمرکز خود را بر روی توسعه‌ی پروژه خواهند گذاشت و در مورد ساختار پایگاه داده و دسترسی به داده‌ها نگرانی نخواهند داشت.

لزوماً همیشه ارتباط یک به یک میان جداول و کلاس‌ها وجود ندارد. به عبارت دیگر همیشه این طور نیست که به ازای یک جدول یک کلاس وجود داشته باشد. ارتباط میان جداول در پایگاه داده با ارتباط میان کلاس‌ها در یک زبان برنامه‌نویسی شیءگرا کاملاً متفاوت است و در اینجاست که Entity Framework یک راه حل کاملاً متفاوت برای چنین سناریوهایی ارائه می‌دهد. در Entity Framework، اجزای پایگاه داده (جداول، Stored Procedureها، View ها) از طریق یک مدل مفهومی ایجاد می‌شوند. مدل مفهومی مزایای بسیاری دارد؛ به عنوان مثال می‌توان یک جدول را به دو

موجودیت مستقل تبدیل کرد. خلاصه اینکه در مدل مفهومی، انعطاف‌پذیری فوق‌العاده‌ای را در تطبیق اجزای پایگاه داده با معادل مفهومی آنها خواهید داشت.

Entity Framework برای یک پایگاه داده، سه مدل ایجاد می‌کند: مدل مفهومی (Conseptual)، مدل منطقی (Logical)، و مدل فیزیکی (Physical). هر یک از این مدل‌ها به تفصیل در فصل ۳ توضیح داده خواهند شد.

در یک مدل مفهومی، برنامه‌نویس در محیطی شیءگرا که با آن آشناست کار خواهد کرد. Visual Studio.NET که محیطی برای نوشتن برنامه‌های شیءگرای مبتنی بر .NET است، با در اختیار قرار دادن امکانات فوق‌العاده‌ای همچون IntelliSense و بررسی خطاها در زمان کامپایل برنامه، یک محیط غنی را برای برنامه‌نویسان به وجود آورده است. به عنوان مثال اگر کدی را که در چند صفحه قبل دیدید با استفاده از Entity Framework بازنویسی کنید، خطای عدم وجود فیلدی با نام MidName، در زمان کامپایل پروژه دریافت خواهد شد و نه در زمان اجرای آن.

توجه داشته باشید که Entity Framework مزایای بسیار بیشتری از آنچه که تاکنون خواندید دارد که در ادامه‌ی این فصل و کتاب با آنها آشنا خواهید شد.

پایگاه داده و مدل

اکنون شما دو روش دسترسی به داده‌ها در اختیار دارید. یا به طور مستقیم از کلاس‌هایی مانند DataReader و DataSet و یا این‌که از Entity Framework استفاده کنید. یک برنامه‌نویس باید از کدام روش استفاده کند؟ از آنجا که این کتاب در مورد Entity Framework است واضح است که شما را به استفاده از Entity Framework ترغیب خواهد کرد؛ با این وجود، دو قسمت آتی در مورد تفاوت‌های هر دو روش صحبت می‌کند.

روش پایگاه داده محور

برنامه‌نویسانی که با DataReader و DataSet کار می‌کنند به خوبی می‌دانند که بیشترین وقت و کد آنها صرف ارتباط با بانک اطلاعاتی، بازیابی داده‌ها، انجام یک سری اعمال بر روی آنها و تبدیل داده‌های خام به کلاس‌های معادل جداول می‌شود. حداقل کدی که برای ایجاد یک ارتباط معمولی با پایگاه داده نوشته می‌شود به شکل ذیل است:

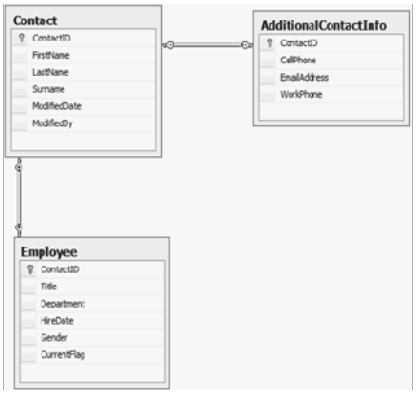
```
using (SqlConnection conn = new SqlConnection(@"Data
Source=(local);Initial Catalog=AdventureWorks;UID=sa;PWD=pwd"))
{
conn.Open();
using (SqlCommand cmd = new SqlCommand())
    {
        cmd.Connection = conn;
        cmd.CommandType = CommandType.Text;
        cmd.CommandText = "SELECT FirstName, MiddleName, LastName
FROM Person.Contact WHERE LastName = @LastName";
cmd.Parameters.Add(new SqlParameter("@LastName",
System.Data.SqlDbType.NVarChar)).Value = "Behrouz";
using (SqlDataReader rdr = cmd.ExecuteReader())
        {
if (rdr.HasRows)
            {
rdr.Read();
                //
            }
        }
    }
}
```

این روش کدنویسی در گذشته استفاده می‌شد که موجب ادغام کدهای برنامه و داده‌ها (ساختار پایگاه داده) می‌گردد. در این حالت، برنامه به طور ضمنی حاوی مدل فیزیکی پایگاه داده است.

روش مدل محور

با وجود Entity Framework، نیازی نیست نگران پایگاه داده باشید. در عوض، خیلی ساده برای تعدادی شیء (موجودیت) که از روی ساختار پایگاه داده ایجاد شده‌اند کوئری می‌نویسید و نتیجه به صورت شیء یا مجموعه‌ای از اشیاء مرتبط به هم برگشت داده می‌شود و بر خلاف روش پایگاه داده محور، برنامه‌نویس مجبور نیست تا با صرف هزینه و نوشتن کدهای بسیار، داده‌های برگشتی را به قالب اشیاء (کلاس‌ها) در آورد؛ این کار به طور خودکار توسط Entity Framework انجام می‌گیرد.

عبارت یا مفهومی که باید با آن آشنا شوید، Entity Data Model یا به اختصار EDM است. EDM، پایه و اساس Entity Framework است و از سه مدلی که پیش‌تر به آنها اشاره شد تشکیل شده است: مدل مفهومی، مدل منطقی، و مدل فیزیکی. می‌توان به EDM به عنوان یک مدل پیشرفته از روش ERM نگاه کرد. EDM، بخشی است که مدل کلی اجزای سیستم را نمایش می‌دهد. Entity Framework کاری بیش از یک ارتباط یک به یک میان اجزای پایگاه داده و مدل شیءگرای معادل آنها ایجاد می‌کند. به عنوان مثال، شکل ۱-۱ که بخشی از پایگاه داده‌ی AdventureWorks است را ملاحظه کنید. در این شکل، ۳ جدول وجود دارد: Contact، Employee و AdditionalContactInfo. در جدول Contact، اطلاعات شخصی افراد نگهداری می‌شود. در جدول Employee اطلاعات کارمندی افراد نگهداری می‌شود. در جدول AdditionalContactInfo نیز اطلاعات ارتباطی دیگری از افراد نگهداری می‌شود. فیلد ContactID نیز به منظور ارتباط بین این سه جدول استفاده می‌شود.



شکل ۱-۱: مدل پایگاه داده

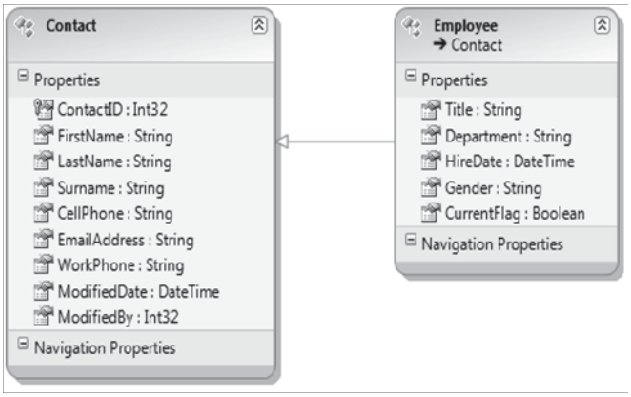
یک برنامه‌نویس ممکن است نیاز به یک کوئری داشته باشد تا اطلاعاتی را از هر سه جدول استخراج کند. به عنوان مثال، ممکن است فرمی وجود داشته باشد که نیاز است تا اطلاعات کارمند از قبیل نام، نام خانوادگی، شغل، تاریخ استخدام و تعدادی اطلاعات دیگر را نمایش دهد. یک کوئری که می‌تواند چنین اطلاعاتی را استخراج کند به شکل ذیل است:

```
SELECT c.FirstName, c.LastName, e.Title, e.HireDate, aci.CellPhone,
aci.EmailAddress
FROM Contact c
INNER JOIN AdditionalContactInfo aci ON c.ContactID = aci.ContactID
INNER JOIN Employee e ON c.ContactID = e.ContactID
```

مدلی که Entity Framework از روی پایگاه داده ایجاد می‌کند با نام Entity Data Model یا به اختصار EDM شناخته می‌شود.

EDMیی که Entity Framework از روی اجزای پایگاه داده ایجاد می‌کند لزوماً همانند مدل آنها در پایگاه داده نیست. EDM، ساختار پایگاه داده را در یک قالب مفهومی ارائه می‌دهد؛ در صورتی که پایگاه داده، اجزای تشکیل دهنده‌ی پایگاه داده را دربر می‌گیرد. اینها دو مدل مجزا از یکدیگر هستند اما هدف Entity Framework، تطبیق و تبدیل اجزای پایگاه داده با معادل شیءگرای آنها است تا در لایه‌ی منطقی برنامه که از آن با نام BLL یاد می‌شود بتوان به راحتی از آنها استفاده نمود.

به عنوان مثال، EDMیی که Entity Framework از روی سه جدول قبل ایجاد می‌کند "می‌تواند" توسط برنامه‌نویس به شکل ۲-۱ تبدیل شود.



شکل ۲-۱: یک مدل تبدیل شده با استفاده از مفهوم ارثبَری

با استفاده از LINQ، بسیار ساده و زیبا می‌توان کوئری قبل را به شکل ذیل نوشت:

```
from c in Contact.TypeOf<Employee> select c;
```

توجه داشته باشید که با وجود Entity Framework، برنامه‌نویسان با یک سطح شیءگرا سر و کار خواهند داشت نه به طور مستقیم با جداول و اجزای پایگاه داده. در فصل ۲ با EDM به طور کامل آشنا خواهید شد.

کار با موجودیت‌ها

یک واژه‌ی کلیدی که هر فردی که از Entity Framework استفاده می‌کند باید با آن آشنا باشد، واژه‌ی"Entity" است. Entity یا همان "موجودیت" در یک پایگاه داده به هر چیزی اطلاق می‌شود که در مورد آن اطلاعاتی نگهداری می‌کنیم و برای آن، جدولی در پایگاه داده بدین منظور در نظر گرفته شده است. به عنوان مثال، "کارمند" یا "ماشین"، هر کدام یک موجودیت هستند. اما در Entity Framework این موجودیت‌ها به معادل شیءگرای خود تبدیل می‌شوند و به هر کلاسی که برای هر موجودیت ایجاد می‌شود یک Entity گفته می‌شود. دو شیئی که در شکل ۲-۱ مشاهده کردید دو Entity هستند. در فصل ٤ با موجودیت‌ها به طور کامل آشنا خواهید شد اما در حال حاضر چون آشنایی با آنها برای کار با Entity Framework و همچنین EDM ضروری است، یک آشنایی مختصر لازم به نظر می‌رسد.

موجودیت‌ها همانند اشیا هستند. به عنوان مثال:

- ✓ موجودیت‌ها یک نوع مشخص دارند.
- ✓ موجودیت‌ها یک یا چند Property دارند و نوع هر Property مشخص است. به عنوان مثال، یک Property با نام Phone که از نوع int است.
- ✓ Property یک Entity می‌تواند ارجاعی به Property یک Entity دیگر داشته باشد. از این طریق بین Entityها ارتباط ایجاد می‌شود.
- ✓ هر Entity یک یا چند ستون دارد که رکوردهای آن را از یکدیگر متمایز می‌کند. در حقیقت هر Entity باید حداقل یک فیلد Identity داشته باشد.

اگر چه موجودیت‌ها از Propertyها تشکیل شده‌اند اما می‌توانند رفتار (Method) مختصری نیز داشته باشند.

موجودیت‌ها با اشیاء نیز کمی متفاوت هستند. به عنوان مثال:

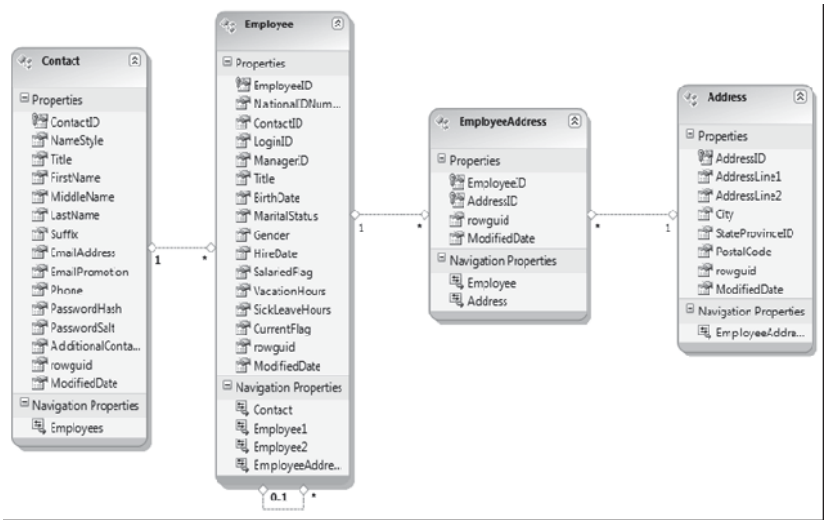
- ✓ موجودیت‌ها در یک مجموعه وجود دارند.
- ✓ هر موجودیت با موجودیت دیگر به نوعی در ارتباط است.
- ✓ موجودیت‌ها کلید اصلی (Primary Key) دارند تا رکوردهای آنها قابلیت متمایز شدن داشته باشند.

تفاوت‌های فوق، همان شباهت موجودیت‌ها و جداول پایگاه داده است.

موجودیت‌ها با جداول پایگاه داده تفاوت‌هایی نیز دارند:

- ✓ یک Property در یک موجودیت می‌تواند از نوع Complex باشد. در ادامه، با Propertyهای Complex آشنا خواهید شد.
- ✓ موجودیت‌ها از ارث‌بری پشتیبانی می‌کنند.
- ✓ برای موجودیت‌ها مفهومی با عنوان ذخیره‌سازی فیزیکی همانند جداول پایگاه داده معنا ندارد. آنها اشیایی هستند که هیچ اطلاعی در مورد نحوه‌ی ذخیره‌سازی خود در پایگاه داده ندارند.

موجودیت‌هایی که توسط EDM ایجاد می‌شوند بسیار انعطاف‌پذیر هستند. خیلی راحت می‌توان آنها را تغییر شکل داد. مثالی در این مورد را در شکل ۲-۱ مشاهده کردید. موجودیت‌ها می‌توانند با یکدیگر ارتباط داشته باشند و این ارتباط به طور خودکار توسط Entity Framework از روی ارتباط‌هایی که طراح پایگاه داده بین جداول ایجاد کرده است استخراج می‌شود. اگر این ارتباط‌ها در پایگاه داده ایجاد نشده بود، می‌توان موجودیت‌ها را به طور دستی و مستقیم از طریق EDM Designer با یکدیگر مرتبط کرد.



شکل ۳-۱: موجودیت‌ها و ارتباط بین آنها

پیش‌تر مشاهده کردید که چقدر ساده توانستیم با استفاده از LINQ برای چنین مدل‌هایی کوئری بنویسیم. در مدل ایجاد شده، همان طور که در شکل ۳-۱ ملاحظه می‌کنید، تمامی ارتباطات بین موجودیت‌ها به طور خودکار توسط Entity Framework ایجاد شده است. EDMDesigner، نحوه‌ی ارتباط میان موجودیت‌ها را به شکل تصویری به راحتی فراهم می‌کند. در شکل ۳-۱، هر موجودیت

از یک یا چند Navigation Property ایجاد شده است. به عنوان مثال، مشخصات یک کارمند که اطلاعات آن در جدول Employee ذخیره می‌شود، در جدول Contact ذخیره می‌گردد. از آنجا که جداول Employee و Contact با یکدیگر ارتباط دارند، پس یک Navigation Property با نام Contact در موجودیت Employee وجود دارد تا بتوان به سادگی به اطلاعات فردی یک کارمند از طریق شیئی که بعداً از کلاس Employee ایجاد می‌شود دسترسی پیدا کرد. Navigation Propertyها به طور خودکار از روی ارتباط میان موجودیت‌ها ایجاد می‌شوند.

قابلیت‌های Entity Framework 4.0

آخرین مبحثی که در این فصل در مورد آن صحبت می‌کنیم، برخی از قابلیت‌های جدید Entity Framework 4.0 است. لیست ذیل مهم‌ترین این قابلیت‌هاست که در نخستین نسخه‌ی Entity Framework که نسخه‌ی 3.5 بود فقدان آنها به طرز ملموسی احساس می‌شد.

- ✓ پشتیبانی از POCO (Plain Old CLR Objects)
- ✓ پشتیبانی کامل از روش Model First
- ✓ بازیابی با تأخیر Navigation Propertyهای یک موجودیت که با عنوان Deffered Loading شناخته می‌شود
- ✓ استفاده از متدها در کوئری‌های LINQ to Entities. البته متدهایی که قابلیت تبدیل به معادل دستور پایگاه داده‌ی آنها وجود داشته باشد
- ✓ سرویس جمع و مفردسازی موجودیت‌ها. به عنوان مثال، تبدیل خودکار Category به Categories
- ✓ پشتیبانی از نوع Complex
- ✓ سفارشی‌سازی کدهای .NET تولیدی توسط Entity Framework
- ✓ افزایش قابلیت‌های پنجره‌ی Model Browser

در ادامه به طور خلاصه با هر یک از این قابلیت‌ها آشنا خواهید شد. جزئیات بیشتر در مورد این قابلیت‌ها در ادامه‌ی این کتاب آورده شده است.

پشتیبانی از POCO

فرض کنید کلاس‌های شما که نمایانگر موجودیت‌هایتان هستند را خودتان ایجاد کردید و می‌خواهید از آنها به عنوان معادل شیءگرای جداول پایگاه داده استفاده کنید. Entity Framework این قابلیت را به شما می‌دهد تا بتوانید از این کلاس‌ها استفاده کنید. به این قابلیت، POCO می‌گویند.

پشتیبانی از روش Model First

با استفاده از این روش می‌توان بدون وجود پایگاه داده، ابتدا مدل مورد نظر را با ابزارهایی که EDM Designer در اختیار قرار می‌دهد ایجاد نمود و سپس پایگاه داده را از روی مدل ایجاد شده درست کرد.

پشتیبانی از Deffered Loading

Deffered Loading که از آن با عنوان Lazy Loading نیز یاد می‌شود، تکنیکی است که در هنگام دسترسی به Navigation Propertyهای یک موجودیت به طور خودکار موجب می‌شود تا برای دسترسی به زیر موجودیت مربوطه، یک کوئری ایجاد شود و اطلاعات مورد نظر بازیابی گردد. به عنوان مثال، اگر نیاز به بازیابی اطلاعات تماس یک کارمند نباشد، چه لزومی دارد تا جدول Contact نیز در نتیجه‌ی کوئری شرکت کند؟ آیا بهتر نیست حضور جدول Contact تنها در زمانی باشد که به اطلاعات تماس کارمند نیاز است؟ البته بازیابی با تأخیر، معایبی نیز دارد، از جمله ایجاد Connection و اجرای کوئری مجدد بر روی پایگاه داده. باید شرایط را سنجید و روش صحیح را برگزید.

استفاده از متدها در کوئری‌های LINQ to Entities

پشتیبانی از متدها در نخستین نسخه از Entity Framework محدود بود. در یک پایگاه داده، منظور از متد، یک Stored Procedure یا یک User Defined Function است.

دو کلاس جدید با نام‌های SqlFunctions و EntityFunctions در Entity Framework 4.0 برای افزایش قابلیت پشتیبانی از متدها اضافه شد. همچنین یک Attribute با نام EdmFunctionAttribute نیز معرفی شد تا زمانی که به یک متد اعمال شود نمایانگر این باشد که آن متد واسطه‌ای برای

فراخوانی Stored Procedure یا User Defined Functionیی است که در پایگاه داده تعریف شده است. جزئیات بیشتر در این زمینه را در فصل ۵ خواهید دید.

سرویس جمع و مفردسازی اسامی

طراحان پایگاه داده معمولاً از اسامی جمع برای نام‌گذاری جداول پایگاه داده استفاده می‌کنند. برای نمونه، موجودیتی مانند Category را به صورت Categories به نام جدول اطلاق می‌کنند. اما از آنجا که حالت تناظری بین جداول و اشیاء معادل آنها توسط Entity Framework ایجاد می‌شود و برنامه-نویس در هر لحظه با "یک شیء" از آن موجودیت سر و کار دارد، وجود موجودیتی با نام Categories جالب نخواهد بود. در نسخه‌ی پیشین Entity Framrework، نام یک موجودیت، هم به طور مستقل و هم در حالتی که به عنوان Navigation Property در یک موجودیت دیگر وجود داشت، همان نامی بود که طراح پایگاه داده به جدول داده بود اما در Entity Framework 4.0، سرویس جمع و مفردسازی اسامی معرفی شد و این نام‌گذاری به طور خودکار در مدل مفهومی به موجودیت‌ها اعمال می‌شود.

پشتیبانی از نوع Complex

پشتیبانی کامل از نوع Complex یکی از قابلیت‌های بسیار خوب Entity Framework 4.0 است. نوع Complex، یک موجودیت سفارشی ایجاد شده توسط برنامه‌نویس است که شامل Propertyهایی مانند Propertyهای موجودیت‌های دیگر است. از نوع Complex بیشتر برای دسته‌بندی خصوصیات مشترک بین موجودیت‌ها استفاده می‌شود. به عنوان مثال می‌توان یک نوع Complex سفارشی با نام Address با دو Property با نام‌های Street1 و Street2 ایجاد و این نوع را به صورت یک Property به چند موجودیت اضافه کرد. با نوع Complex به طور کامل در فصل ۳ آشنا خواهید شد.

سفارشی سازی کدهای .NET. تولیدی

کدهای دات‌نتی که برای کار با داده‌ها به آنها نیاز است به طور خودکار توسط Entity Framework تولید می‌شوند. در Entity Framework 4.0 می‌توان خروجی نهایی را مدیریت کرد و بنا به نیاز تغییر داد.

افزایش قابلیت‌های پنجره‌ی Model Browser

چندین قابلیت جدید به پنجره‌ی Model Browser اضافه شده است که کار با آن را راحت‌تر کرده است. این قابلیت‌ها شامل موارد زیر است:

- ✓ امکان آپدیت کردن مدل پس از اعمال تغییرات به پایگاه داده
- ✓ حذف اشیاء از مدل
- ✓ جستجو برای عبارتی خاص در مدل مفهومی و مدل ذخیره‌سازی
- ✓ تعیین نوع Property در محیط طراحی مدل

اینها مهم‌ترین قابلیت‌ها بودند. در این مورد به تفصیل در فصل ۲ خواهید خواند.

پایگاه‌های داده‌ی پشتیبانی شده

نکته‌ی جالبی که در مورد Entity Framework وجود دارد این است که آن در ذات خود در مورد پایگاه داده‌ای که با آن ارتباط برقرار می‌کند هیچ اطلاعی ندارد.

Entity Framework به طور پیش‌فرض با دو Provider ارتباط دارد. Provider، واسطی است که انجام وظیفه‌ای خاص را بر عهده می‌گیرد.

۱. EntityClient که به وسیله‌ی برنامه‌های مبتنی بر Entity Framework برای دسترسی به داده‌هایی که ساختار آنها در EDM تعریف شده است استفاده می‌شود. این Provider از SqlClient که Providerیی برای دسترسی به داده‌های پایگاه داده‌ی SQL Server است برای SQL Server استفاده می‌کند.
۲. SQL Provider که در .NET. وجود دارد و برای ارتباط با پایگاه داده‌ی SQL Server استفاده می‌شود.

Entity Framework مستقل از پایگاه داده است و بر اساس مدل ADO.NET Data Provider کار می‌کند. بنابراین می‌توان با ایجاد یک Provider سفارشی برای یک پایگاه داده‌ی خاص بر مبنای ADO.NET Data Provider، به انواع پایگاه‌های داده دسترسی داشت. به عنوان مثال، از طریق Providerهایی که توسط شرکت‌های مختلف ایجاد شده‌اند می‌توان با پایگاه‌های داده‌ی ذیل از طریق Entity Framework ارتباط برقرار کرد:

- `Oracle`
- `MySql`

- PostgreSQL
- SQL Anywhere
- DB2
- Informix
- U2
- Ingres
- Progress
- Firebird
- Synergy
- Virtuoso

فهرست فوق را می‌توان معیاری برای میزان محبوبیت Entity Framwork نیز بر شمرد.

برای مشاهده‌ی فهرست کامل این شرکت‌ها می‌توان به پیوند ذیل مراجعه نمود:

http://msdn.microsoft.com/en-us/data/dd363565.aspx

نکته‌ی بسیار زیبایی که در اینجا وجود دارد، تبدیل کوئری‌ها به زبان مربوط به پایگاه داده به طور خودکار توسط Provider مربوطه است. شما تنها باید Connection String را برای Provider تعیین کنید؛ بقیه‌ی کار بر عهده‌ی Provider است. به جای اینکه نیاز باشد تا زبان SQL مختص آن پایگاه داده را فرا گیرید، با LINQ to Entities یا Entity SQL کدهای مورد نیاز برای اعمال CRUD بر روی پایگاه داده را می‌نویسید و وظیفه‌ی تبدیل آنها به SQL معادل پایگاه داده‌ی مربوطه بر عهده‌ی Provider است. در این حالت، دیگر نیازی نیست نگران نوع پایگاه داده‌ی مربوطه باشید. دستورات شما بر روی هر نوع پایگاه داده‌ای کار خواهند کرد! از این بهتر چه انتظاری دارید؟

CRUD از چهار کلمه‌ی Create، Retrieve، Update، Delete گرفته شده است و به معنای اعمال چهارگانه‌ای (ایجاد، بازیابی، آپدیت، حذف) است که بر روی رکوردها در پایگاه داده انجام می‌شود.

در این کتاب از SQL Server 2008 به عنوان پایگاه داده و SqlClient به عنوان Provider آن استفاده شده است. با این Provider می‌توانید با نسخه‌های مختلف SQL Server (۲۰۰۰، ۲۰۰۵، ۲۰۰۸) ارتباط برقرار کنید. حتی از SQL Server Compact Edition نیز پشتیبانی می‌شود.