

گام به گام

Entity Framework 4.0
And ASP.NET Web Forms

Tom Dykstra

ابوذر زارعی

خلاصه: در این کتاب، اصول استفاده از روش Database First در Entity Framework را به منظور نمایش داده ها در یک برنامه ASP.NET Web Forms خواهید آموخت.

طبقه بندی: گام به گام

تهیه شده بر اساس: ASP.NET 4.0, ASP.NET Web Forms, Entity Framework 4.0, Visual Studio 2010

مؤلف: Tom Dykstra

مترجم: مهندس ابوذر زارعی

تاریخ انتشار نسخه اصلی: June 2012

تاریخ انتشار نسخه ترجمه شده: Oct 2013

ایمیل پشتیبانی: a.zarei.se@gmail.com

دانلود پروژه آماده به همراه دیتابیس مورد نیاز:

<http://code.msdn.microsoft.com/ASPNET-Web-Forms-97f8ee9a>

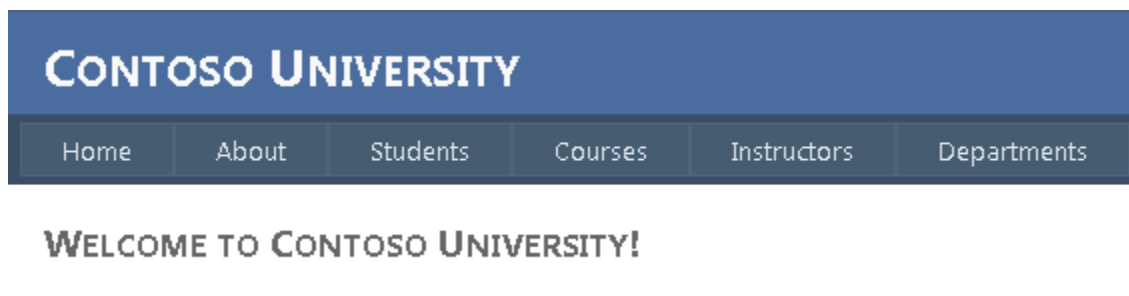
۱	مقدمه
۳	ایجاد Web Application
۶	ساخت پایگاه داده
۱۰	ایجاد Entity Framework Data Model
۱۵	بررسی Entity Framework Data Model
۲۵	کنترل EntityDataSource
۲۶	افزودن کنترل EntityDataSource و تنظیمات آن
۳۲	تنظیمات قوانین (Rule) دیتابیس برای عملیات حذف
۳۶	استفاده از کنترل GridView به منظور خواندن و بروزرسانی موجودیت ها
۴۱	باز بینی کد ساخت یافته کنترل EntityDataSource به منظور بهبود کارایی
۴۳	نمایش داده ها از طریق خصوصیت Navigation
۴۵	استفاده از کنترل DetailsView برای درج موجودیت ها
۴۷	نمایش داده ها در Drop-Down List
۵۱	فیلتر کردن، مرتب سازی و گروه بندی داده ها
۵۳	استفاده از خصوصیت "Where" در EntityDataSource برای فیلتر کردن داده ها
۵۴	استفاده از خصوصیت "OrderBy" کنترل EntityDataSource برای مرتب کردن داده ها
۵۵	استفاده از Control Parameter برای تنظیم کردن خصوصیت "Where"
۶۰	استفاده از خصوصیت "GroupBy" کنترل EntityDataSource برای گروه بندی داده ها
۶۲	استفاده از کنترل QueryExtender به منظور فیلتر کردن و مرتب کردن
۶۶	استفاده از عملگر "Like" برای مرتب سازی داده ها
۶۸	کار با داده های وابسته
۶۹	نمایش و بروزرسانی موجودیت های وابسته در کنترل
۷۴	نمایش موجودیت های وابسته در کنترل
۷۹	استفاده از رخداد "Selected" کنترل EntityDataSource برای نمایش داده های وابسته

۸۳.....	ادامه کار با داده های وابسته.....
۸۵.....	اضافه کردن یک موجودیت همراه با رابطه ای به یک موجودیت موجود.....
۸۸.....	کار با ارتباط های چند به چند.....
۹۴.....	پایاده سازی وراثت Table-per-Hierarchy.....
۹۴.....	وراثت Table-per-Hierarchy در مقابل Table-per-Type.....
۹۶.....	اضافه کردن موجودیت های Student و Instructor.....
۱۰۱.....	نگاشت موجودیت های Student و Instructor به جدول Person.....
۱۰۳.....	استفاده از موجودیت های Student و Instructor.....
۱۱۰.....	استفاده از رویه های ذخیره شده.....
۱۱۱.....	ایجاد رویه های ذخیره شده در دیتابیس.....
۱۱۴.....	اضافه کردن رویه ذخیره شده به مدل داده.....
۱۱۵.....	نگاشت رویه های ذخیره شده.....
۱۱۹.....	استفاده از رویه های ذخیره شده ی درج، بروزرسانی و حذف.....
۱۲۱.....	استفاده از رویه های ذخیره شده انتخاب.....
۱۲۲.....	استفاده از Dynamic Data Functionality برای شکل دهی و اعتبار سنجی داده ها.....
۱۲۳.....	استفاده از کنترل های DynamicField و DynamicControl.....
۱۲۸.....	اضافه کردن فراداده به مدل داده.....
۱۳۲.....	کنترل ObjectDataSource.....
۱۳۳.....	کلاس های منطق تجاری و مخزن.....
۱۳۵.....	بروزرسانی دیتابیس و مدل داده.....
۱۳۶.....	اضافه کردن رابطه به دیتابیس.....
۱۳۸.....	اضافه کردن دید به دیتابیس.....
۱۳۹.....	به روزرسانی مدل داده.....
۱۴۴.....	استفاده از کلاس مخزن و کنترل ObjectDataSource.....
۱۴۸.....	اضافه کردن عملیات های درج و حذف.....

۱۴۹.....	متد Attach
۱۵۰.....	متد SaveChanges
۱۵۰.....	برگرداندن نام اساتید برای انتخاب در زمان درج
۱۵۱.....	ایجاد یک صفحه برای درج دانشکده ها
۱۵۵.....	اضافه کردن عملیات بروزرسانی
۱۶۰.....	اضافه کردن لایه منطق تجاری و تست واحد
۱۶۰.....	ایجاد واسط مخزن
۱۶۲.....	ایجاد کلاس منطق تجاری
۱۶۶.....	ایجاد یک پروژه تست واحد و پیاده سازی مخزن
۱۶۸.....	ایجاد تست های واحد
۱۷۲.....	اضافه کردن منطق تجاری برای موفقیت در تست
۱۷۶.....	اداره کردن استثناهای ObjectDataSource
۱۸۰.....	مرتب سازی و فیلتر کردن
۱۸۱.....	اضافه کردن قابلیت مرتب سازی ستون های GridView
۱۸۴.....	اضافه کردن جعبه متنی جستجو
۱۸۶.....	افزودن ستون Details برای هر Grid Row
۱۸۹.....	اداره کردن همزمانی
۱۹۰.....	برخوردهای همزمانی
۱۹۱.....	همزمانی بدبینانه
۱۹۱.....	همزمانی خوشبینانه
۱۹۳.....	تشخیص برخوردهای همزمانی
۱۹۴.....	اداره کردن همزمانی خوشبینانه بدون خصوصیت tracking
۱۹۵.....	فعال سازی Tracking همزمانی در مدل داده
۱۹۶.....	اداره کردن استثناهای همزمانی در لایه دسترسی به داده (DAL)
۱۹۶.....	اداره کردن استثناهای همزمانی در لایه ارائه

۱۹۹.....	Departments	در صفحه خوشبینانه
۲۰۱.....	tracking	از خصوصیت استفاده با خوشبینانه
۲۰۱.....	OfficeAssignment	به مدل داده شده ذخیره رویه
۲۰۴.....	DAL	به OfficeAssignment متدهای افزودن
۲۰۶.....	BLL	به OfficeAssignment متدهای افزودن
۲۰۸.....	OfficeAssignments	وب ایجاد صفحه
۲۱۰.....	OfficeAssignments	در صفحه خوشبینانه
۲۱۱.....	EntityDataSource	با کنترل اداره کردن همزمانی
۲۱۶.....		بالا بردن کارایی
۲۱۷.....		بارگذاری موثر داده های مرتبط
۲۲۰.....		مدیریت وضعیت دید
۲۲۲.....	NoTracking	از گزینه استفاده
۲۲۳.....	LINQ	قبل از کامپایل
۲۲۶.....		ارسالی به دیتابیس
۲۳۵.....		پیش تولید کردن دیدها
۲۴۱.....	Entity Framework 4	وجود دارد
۲۴۱.....		وابستگی های کلید خارجی
۲۴۳.....	SQL	تعریف شده توسط کاربر
۲۴۶.....	Model-First	توسعه
۲۵۶.....	POCO	از پشتیبانی
۲۵۷.....	Code-first	توسعه
۲۵۸.....		اطلاعات بیشتر

برنامه ای که شما با توجه به مطالب آموزشی این کتاب خواهید ساخت، نمونه ساده ای از یک سایت دانشگاه است.



کاربرها که عبارت اند از دانشجویان و اساتید امکان مشاهده یا ویرایش اطلاعات مجاز را خواهند داشت. تعدادی عکس از بخش هایی که شما در آینده ایجاد خواهید کرد را مشاهده می کنید:

STUDENT LIST

	ID	Name	EnrollmentDate
Edit Delete	3	Peggy Justice	9/1/2001
Edit Delete	6	Yan Li	9/1/2002
Edit Delete	7	Laura Norman	9/1/2003

ADD NEW STUDENTS

First Name	<input type="text"/>
Last Name	<input type="text"/>
Enrollment Date	<input type="text"/>
Insert Cancel	

COURSES BY DEPARTMENT

Select a Department

CourseID	Title	Credits
1050	Chemistry	4
1061	Physics	4

COURSES BY NAME

Enter a course name

Department	CourseID	Title	Credits
Economics	4041	Macroeconomics	3
Economics	4022	Microeconomics	3

INSTRUCTORS

	ID	Name	Hire Date	Office Assignment
Edit Select	1	Abercrombie, Kim	3/11/1995	17 Smith
Edit Select	4	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	5	Harui, Roger	7/1/1998	37 Williams
Edit Select	18	Zheng, Roger	2/12/2004	143 Smith
Edit Select	25	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	27	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	31	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	32	Xu, Kristen	7/23/2001	203 Williams
Edit Select	34	Van Houten, Roger	12/7/2000	213 Smith

COURSE DETAILS

ID	2030
Title	Poetry
Credits	2
Department	English
Location	
URL	http://www.fine art

STUDENT GRADES

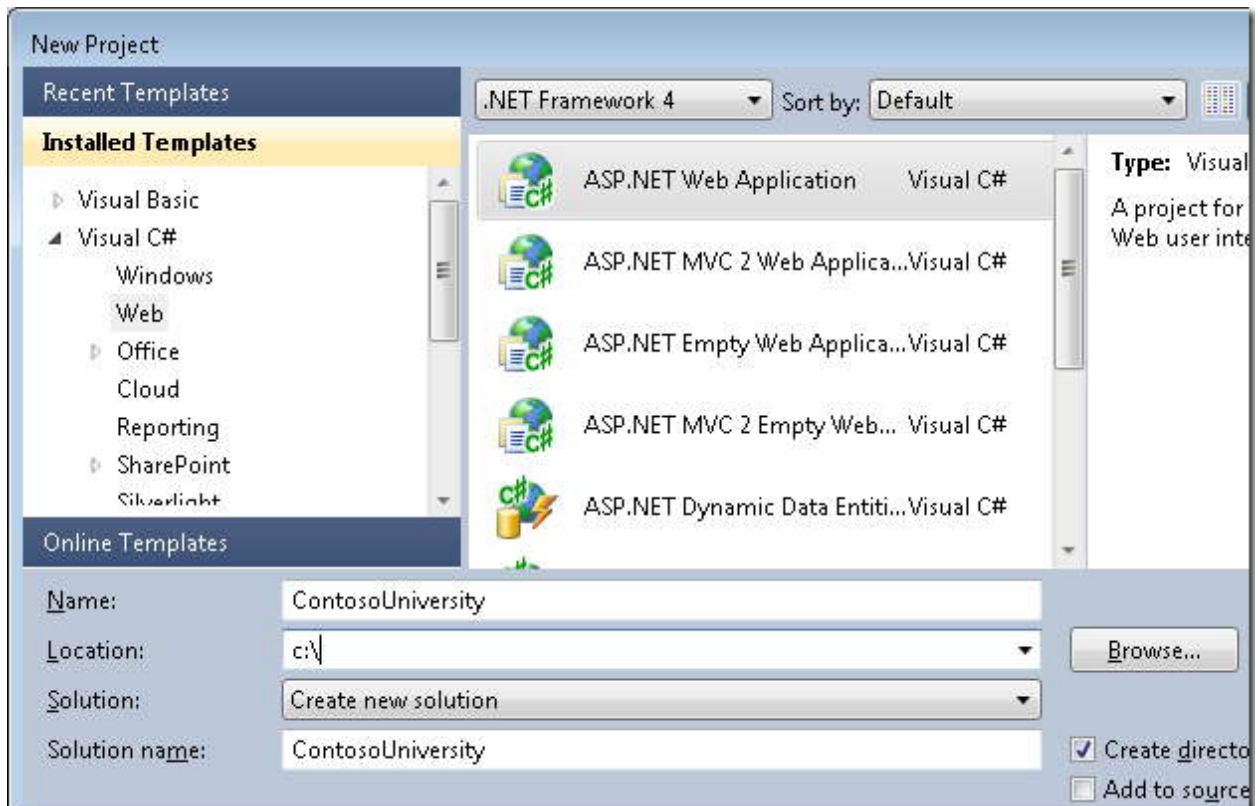
ID	Name	Grade
2	Barzdukas, Gytis	3.50
3	Justice, Peggy	4.00

COURSES TAUGHT

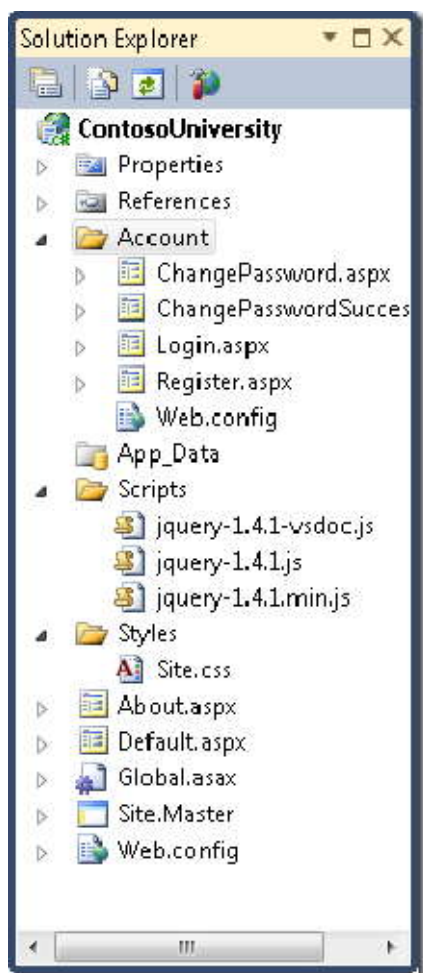
	ID	Title	Department
Select	2030	Poetry	English

ایجاد Web Application

برای شروع، برنامه Visual Studio 2010 را باز کرده و یک ASP.NET Web Application جدید با استفاده از قالب **ASP.NET Web Application** ایجاد کنید:



این قالب یک پروژه Web Application حاوی StyleSheet و MasterPage، ایجاد می کند.



Site.Master را باز کرده و عبارت "My ASP.NET Application" را به "Contoso University" تغییر دهید.

```
<h1>  
    Contoso University  
</h1>
```

کنترل **Menu** با نام **NavigationMenu** را یافته و کد زیر را به جای آن جایگزین کنید. این کد **Menu item** های مورد نیاز صفحاتی که در آینده خواهید ساخت را اضافه میکند.

```

<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu" EnableViewState="false"
IncludeStyleBlock="false"
    Orientation="Horizontal">
    <Items>
    <asp:MenuItem NavigateUrl="/Default.aspx" Text="Home"/>
    <asp:MenuItem NavigateUrl="/About.aspx" Text="About"/>
    <asp:MenuItem NavigateUrl="/Students.aspx" Text="Students">
    <asp:MenuItem NavigateUrl="/StudentsAdd.aspx" Text="Add Students"/>
    </asp:MenuItem>
    <asp:MenuItem NavigateUrl="/Courses.aspx" Text="Courses">
    <asp:MenuItem NavigateUrl="/CoursesAdd.aspx" Text="Add Courses"/>
    </asp:MenuItem>
    <asp:MenuItem NavigateUrl="/Instructors.aspx" Text="Instructors">
    <asp:MenuItem NavigateUrl="/InstructorsCourses.aspx" Text="Course
Assignments"/>
    <asp:MenuItem NavigateUrl="/OfficeAssignments.aspx" Text="Office Assignments"/>
    </asp:MenuItem>
    <asp:MenuItem NavigateUrl="/Departments.aspx" Text="Departments">
    <asp:MenuItem NavigateUrl="/DepartmentsAdd.aspx" Text="Add Departments"/>
    </asp:MenuItem>
    </Items>
</asp:Menu>

```

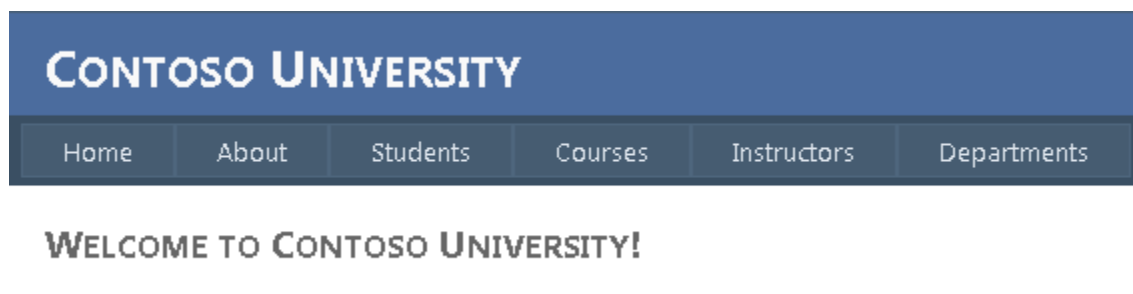
صفحه Default.aspx را باز کرده و کد مربوط به کنترل Content با نام **BodyContent** را به صورت زیر تغییر دهید:

```

<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
<h2>
    Welcome to Contoso University!
</h2>
</asp:Content>

```

حال شما یک صفحه خانگی با لینک هایی به صفحاتی دارید که در آینده خواهید ساخت:



ساخت پایگاه داده

در این بخش شما از Entity Framework data model designer (محیط طراحی مدل داده) برای ایجاد خودکار مدل داده از روی دیتابیس موجود، استفاده خواهید کرد. (که اغلب این روش را "ابتدا پایگاه داده" یا *database-first* می نامند). روش دیگری که در این کتاب به آن پرداخته نشده، به این صورت است که ابتدا به صورت دستی مدل داده را ایجاد می کنید، آنگاه توسط یک designer می توانید اسکریپت هایی تولید کنید که با آن دیتابیس را بسازید. (روش "ابتدا مدل" یا *model-first*)

برای استفاده از روش *database-first* در این بخش، گام بعدی اضافه کردن یک پایگاه داده به سایت است. راحت ترین راه دانلود پروژه از سایتی است که کتاب را از آن دانلود کرده اید، یا دانلود از [سایت مایکروسافت](#) می باشد. پس از دانلود نوبت به اضافه کردن دیتابیس موجود در App_Data پروژه دانلود شده با نام School.mdf (توسط **Add Existing Item**) به پروژه جاری است. راه دیگر دنبال کردن دستورالعمل موجود در [Creating the School Sample Database](#) است. اگر دیتابیس را توسط اسکریپت ایجاد کردید آن را از مسیر زیر به پوشه App_Data برنامه کپی کنید.

```
%PROGRAMFILES%\Microsoft SQL Server\MSSQL10.SQLEXPRESS\MSSQL\DATA
```

توجه: این مسیر با توجه مکان تعیین شده توسط کاربر موقع نصب sql server 2008 express متفاوت است.

(منظور از %PROGRAMFILES% پوشه برنامه های نصب شده در سیستم است که میتواند به طور نمونه به این صورت باشد:

c:\programfiles

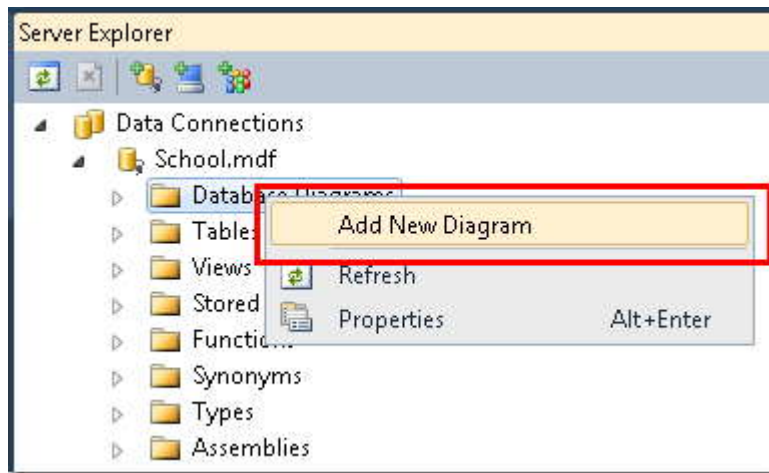
اگر توسط اسکریپت، دیتابیس را ایجاد کرده اید، گام های زیر را برای تولید شمای دیتابیس بردارید:

در قسمت **Server Explorer**، گزینه **Data Connections** و سپس *School.mdf* را گسترش داده، روی **Database Diagrams**

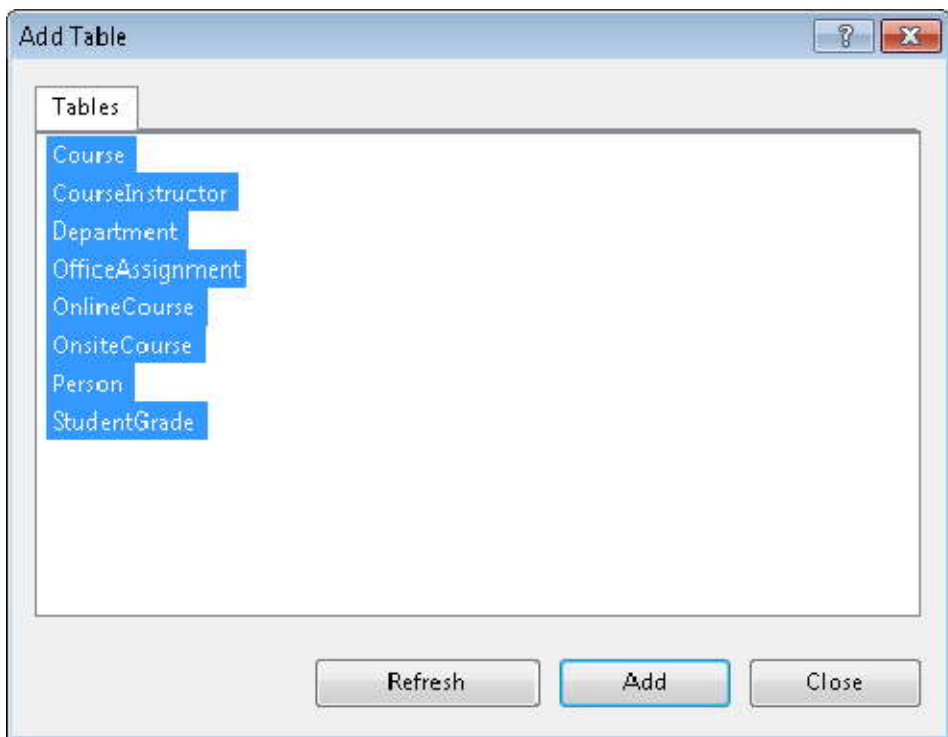
کلیک راست کنید، سپس **Add New Diagram** را انتخاب کنید.

(در این جا ممکن است به خاطر نداشتن اجازه دسترسی برنامه پیغامی برای تولید دوباره دیاگرام نمایش دهد، در این صورت روی

Yes کلیک کنید.)



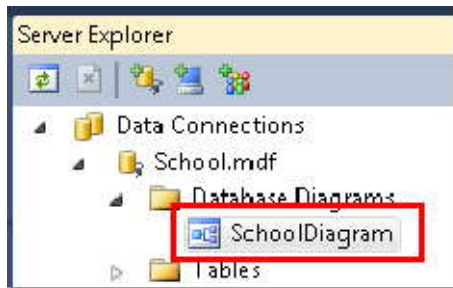
همه جدول ها را انتخاب و سپس روی **Add** کلیک کنید.



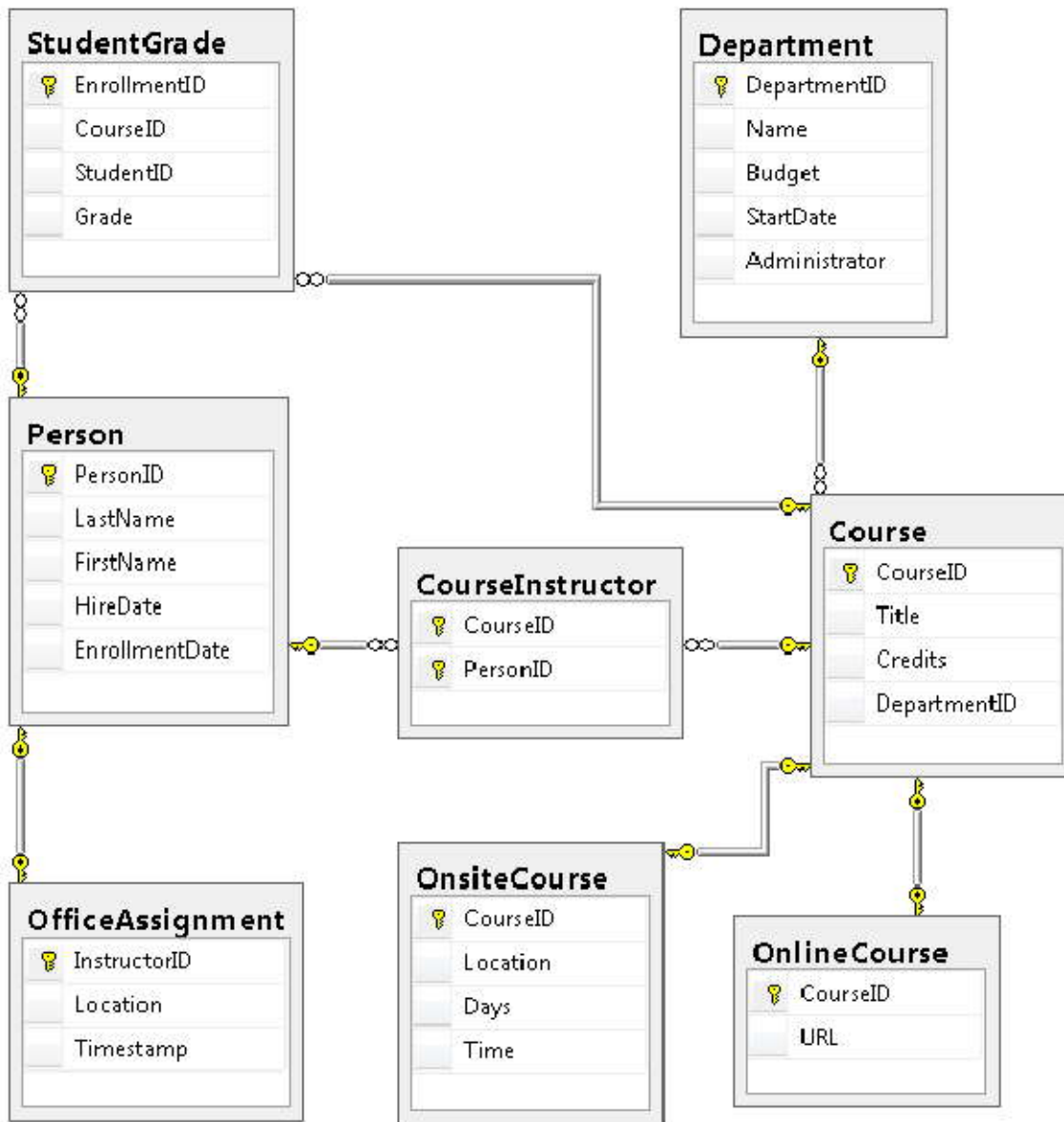
دیاگرام دیتابیس توسط Sql server ساخته می شود، که در آن جدول ها، ستون ها و روابط بین جدول ها نشان داده می شوند. شما می توانید برای سامان دهی ظاهر دیاگرام، جدول ها را در هر جای صفحه که می خواهید جابجا کنید.

دیاگرام را با نام SchoolDiagram ذخیره کرده و ببندید.

اگر فایل School.mdf را دانلود و به پروژه اضافه کرده اید، می توانید دیاگرام دیتابیس را در SchoolDiagram از زیر شاخه های Database Diagrams واقع در Server Explorer مشاهده کنید.



دیاگرام، چیزی شبیه شکل زیر خواهد بود. (ممکن است در دیاگرام شما مکان جدول ها، متفاوت با آنچه که در شکل نمایش داده شده باشد)

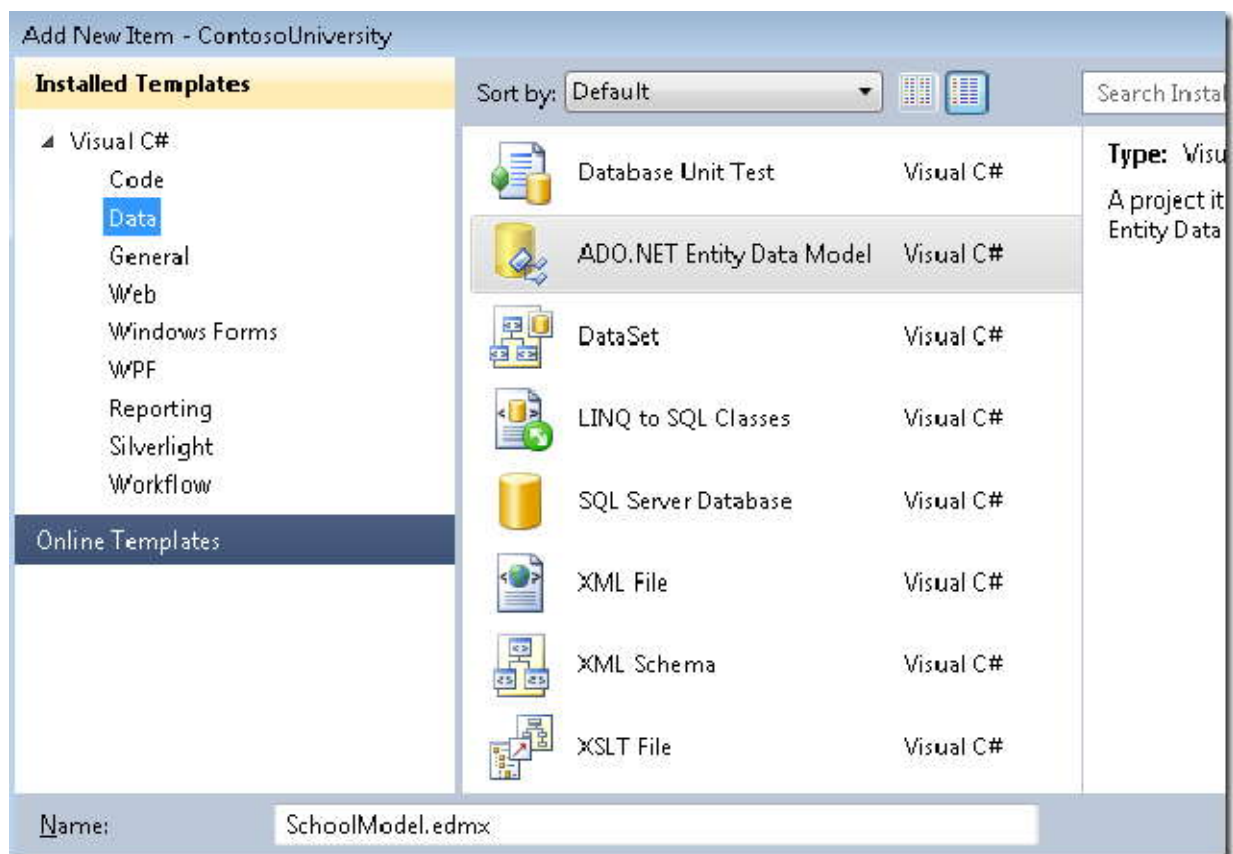


ایجاد Entity Framework Data Model

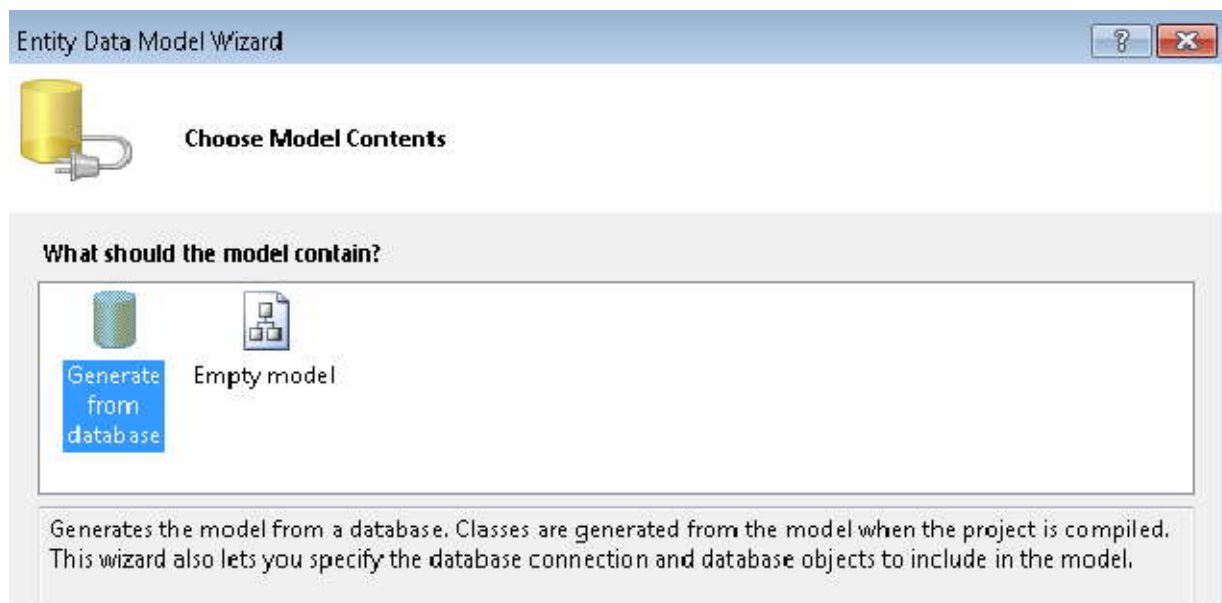
اکنون شما می توانید Entity Framework Data Model را از روی این دیتابیس ایجاد نمائید. می توان این مدل داده را در پوشه اصلی برنامه ایجاد کرد، اما در این بخش آن را درون پوشه ای با نام DAL (مخفف Data Access Layer) قرار دهید.

در قسمت **Solution Explorer** برای پروژه جاری یک پوشه با نام **DAL** اضافه نمائید. (دقت کنید، این پوشه جزو زیر شاخه های پروژه باشد نه Solution).

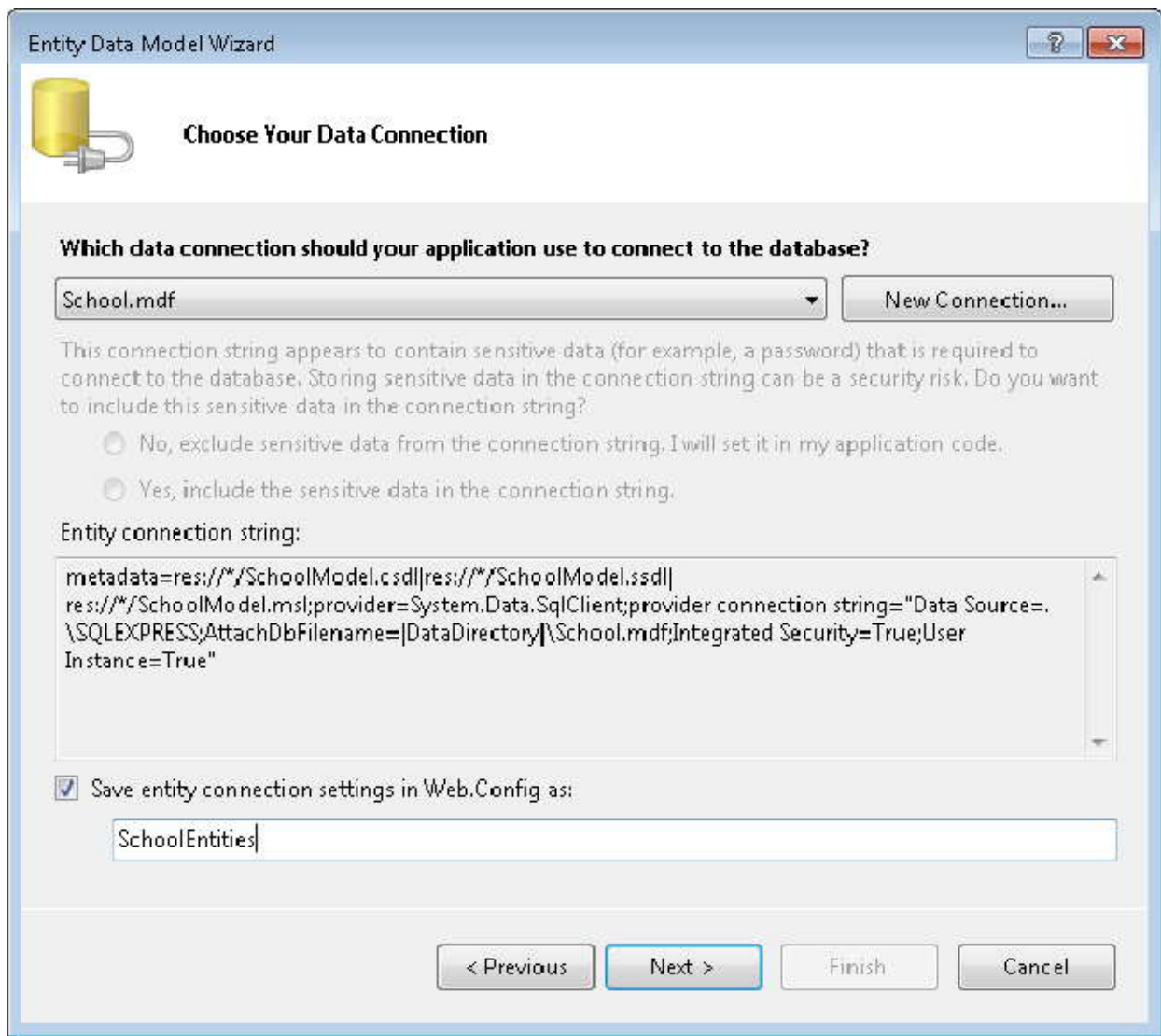
روی DAL کلیک راست کرده، **Add** و سپس **New Item** را انتخاب کنید. در قسمت **Installed Templates** گزینه **Data** و سپس الگوی **ADO.NET Entity Data Model** را انتخاب کرده و نام آن را **SchoolModel.edmx** قرار داده و روی **Add** کلیک کنید.



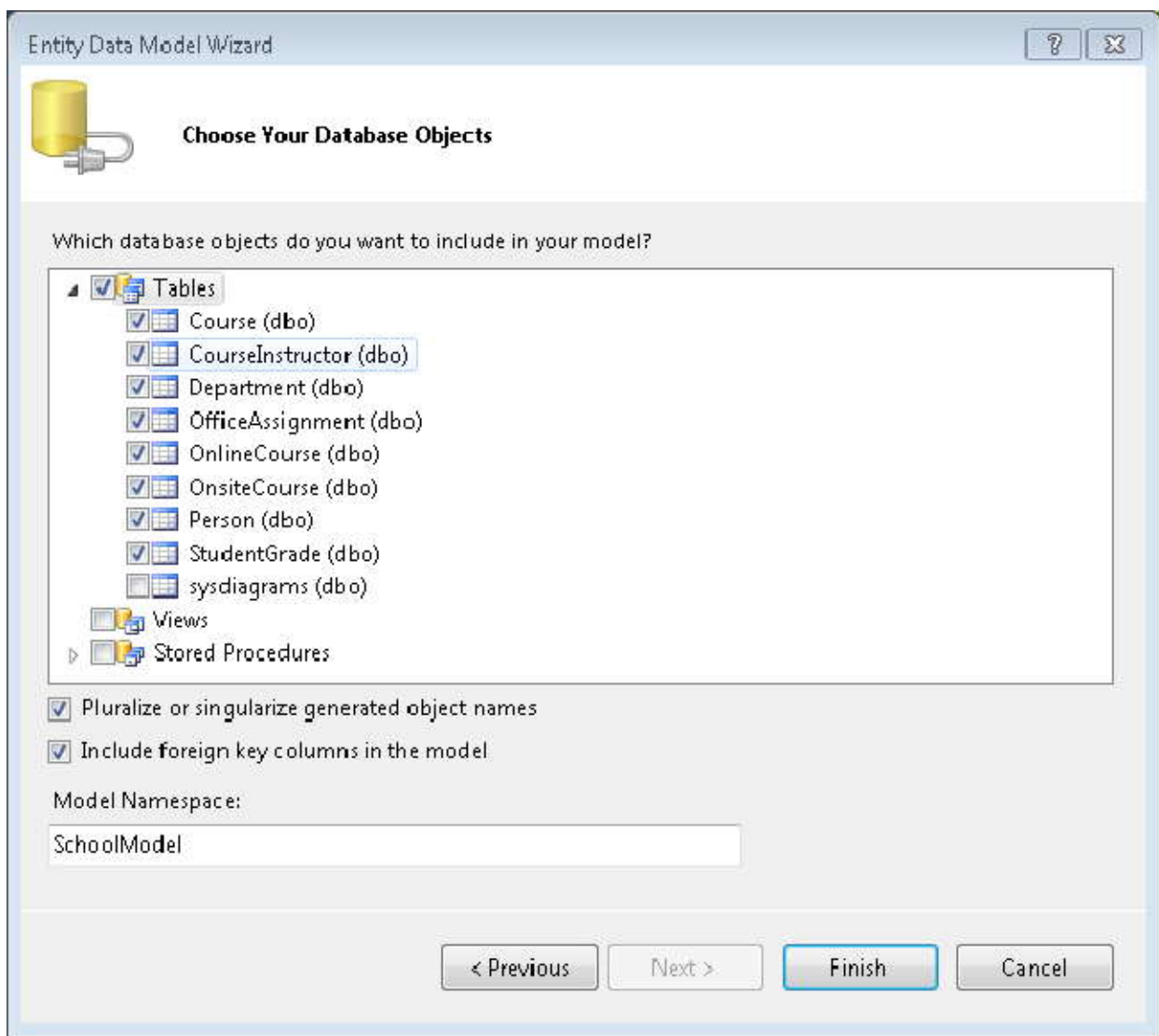
با این کار، Entity Data Model Wizard اجرا می شود. در گام نخست این فرآیند، گزینه **Generate from database** به طور پیش فرض انتخاب شده، پس روی **Next** کلیک کنید.



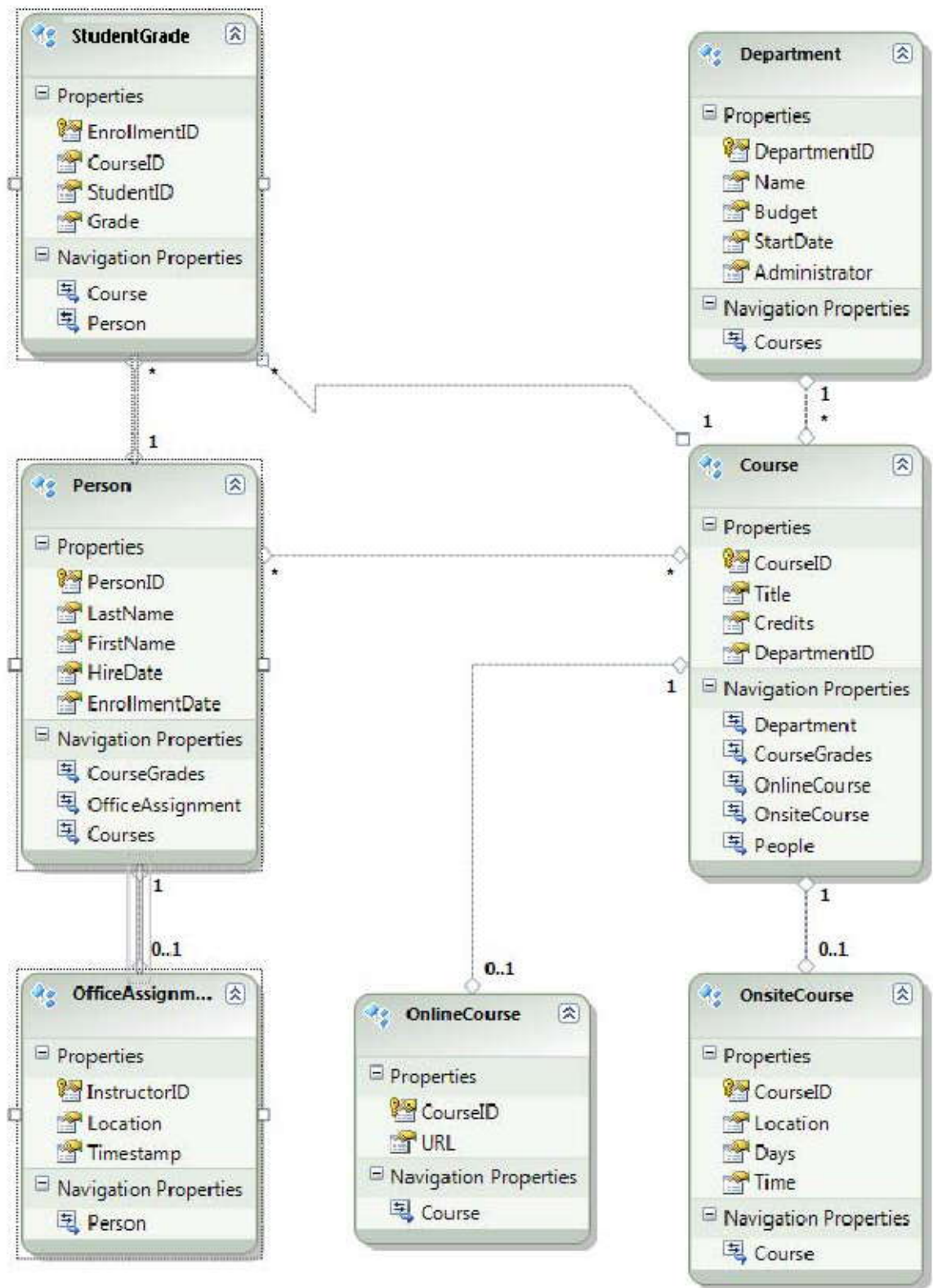
در مرحله **Choose Your Data Connection** مقادیر پیش فرض را رها کرده و روی **Next** کلیک کنید. دیتابیس School به طور پیش فرض انتخاب شده، و رشته اتصال به پایگاه داده یا Connection String در فایل Web.config تحت عنوان SchoolEntities ذخیره می شوند.



در مرحله **Choose Your Database Objects**، همه جدول ها به جز sysdiagrams (که برای دیاگرامی که اخیرا شما ایجاد کرده اید ایجاد شده است) را انتخاب کرده و روی دکمه **Finish** کلیک کنید.

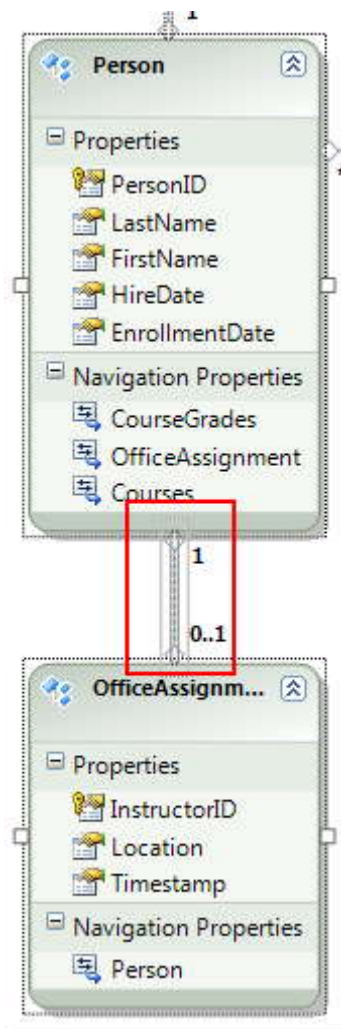


پس از پایان ساخت مدل داده، ویژوال استودیو یک نمای گرافیکی از اشیاء Entity Framework که مشابه جدول های دیتابیس شما هستند، نمایش می دهد.(همانند دیاگرام دیتابیس، مکان هر کدام از اشیاء ممکن است با آنچه که در تصویر زیر دیده می شود، متفاوت باشد. در صورت تمایل می توانید اشیاء را جابجا کنید تا همانند تصویر زیر شود)



بررسی Entity Framework Data Model

همانطور که مشاهده میکنید، Entity Framework diagram خیلی شبیه دیتاگرام دیتابیس است، با دو تفاوت. یک تفاوت، اضافه کردن سمبل (symbol) در پایان هر وابستگی، که نشان گر نوع وابستگی است: (ارتباط جدول ها¹ در Entity Framework Data Model، وابستگی موجودیت² نامیده می شود)

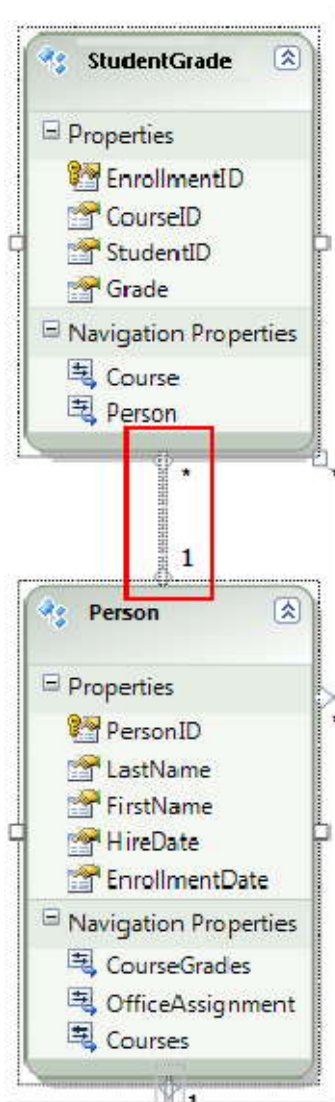


وابستگی یک به صفر، به صورت "1" و "0..1" نشان داده می شود:

در این مورد، موجودیت Person ممکن است با موجودیت OfficeAssignment وابستگی داشته باشد یا نداشته باشد. اما یک موجودیت OfficeAssignment در صورت وجود باید فقط با یک موجودیت Person وابستگی داشته باشد. به عبارت دیگر یک استاد ممکن است دارای یک دفتر کار باشد یا نباشد. اما هر دفتر کار فقط می تواند به یک آموزگار تعلق داشته باشد.

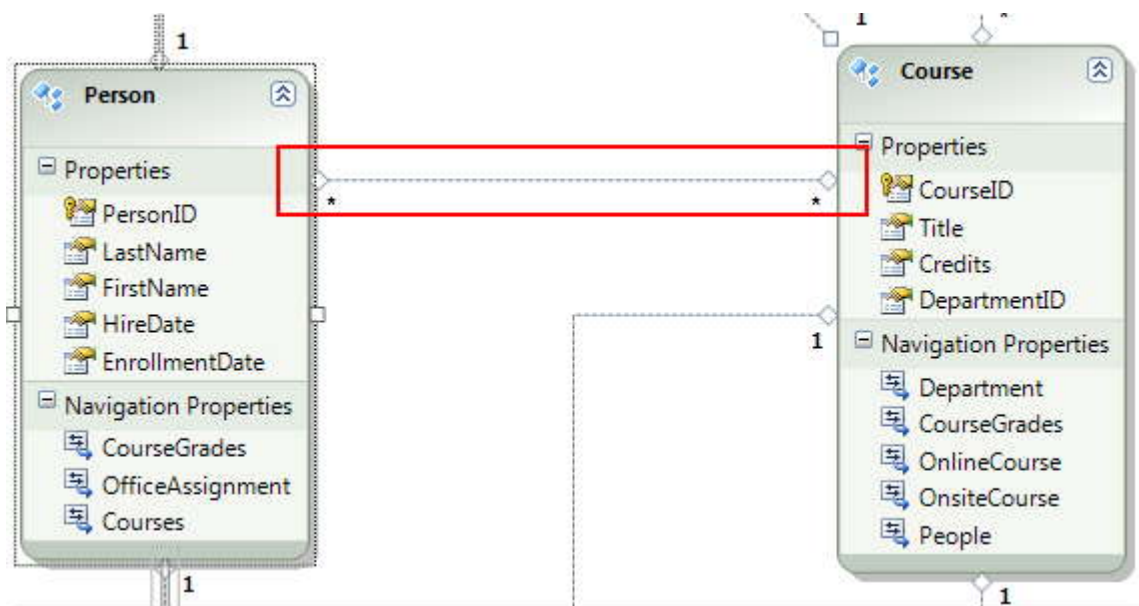
¹ table relationships
² entity associations

وابستگی یک به چند به صورت "1" و "*" نمایش داده می شود:



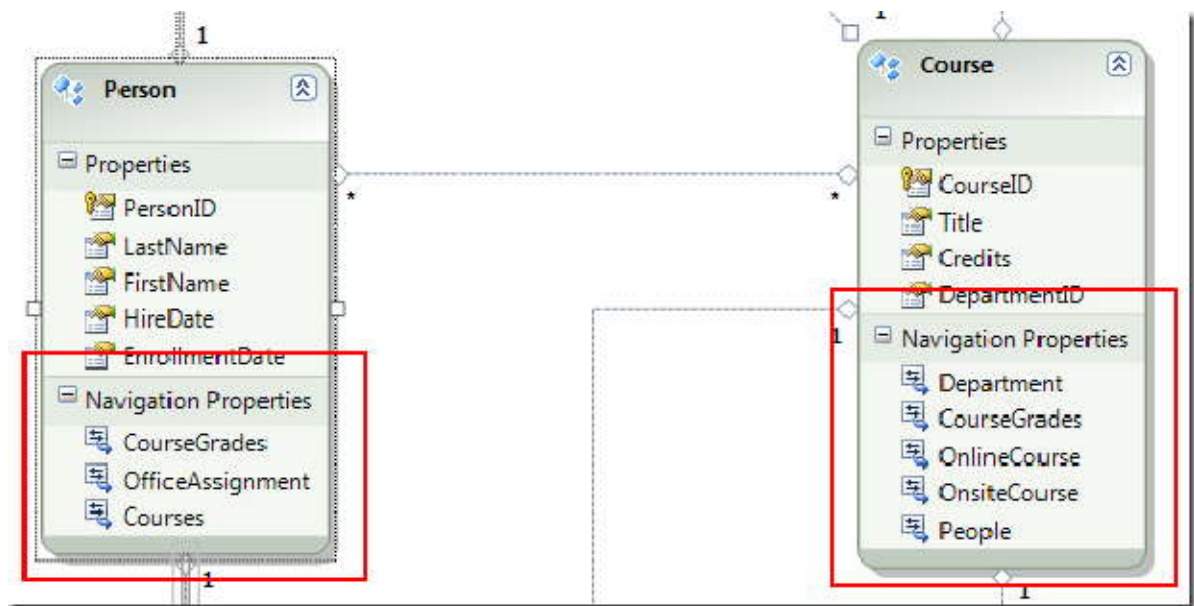
در این مورد موجودیت Person ممکن است با موجودیت StudentGrade وابستگی داشته باشد یا نداشته باشد. موجودیت StudentGrade در صورت وجود، باید فقط با یک موجودیت Person وابستگی داشته باشد. StudentGrade در واقع دوره های ثبت نام شده (به عبارتی درس های برداشته شده توسط دانشجو) در دیتابیس را ارائه می دهد. اگر یک دانشجو در دوره ای ثبت نام کرده باشد و برای هنوز او نمره ای وارد نشده باشد، مقدار خصوصیت Grade برای او null خواهد بود. به عبارت دیگر ممکن است یک دانشجو در یک هیچ یک از دوره ها هنوز ثبت نام نکرده باشد، یا در یک دوره ثبت نام کرده باشد، یا در چندین دوره ثبت نام کرده باشد. هر نمره در یک دوره فقط متعلق به یک دانشجو است.

وابستگی چند به چند به صورت "*" و "*" نشان داده می شود.

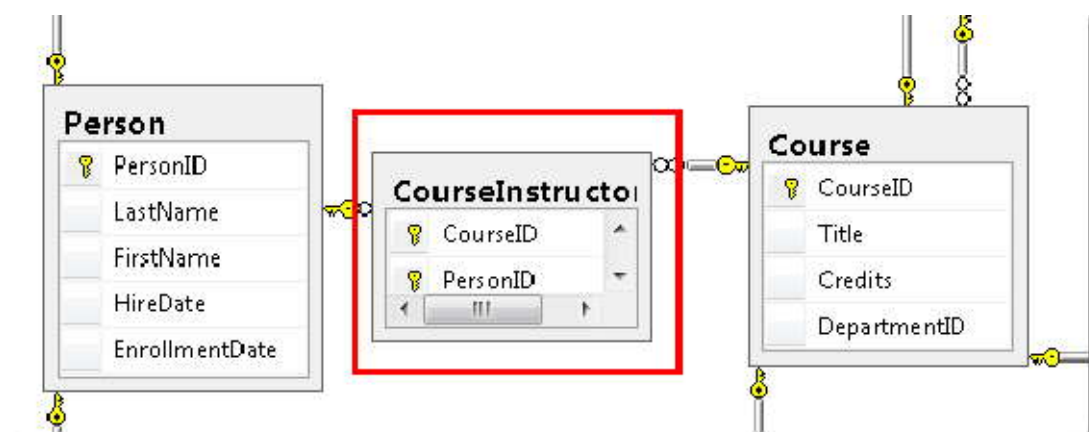


در این مورد موجودیت Person ممکن است با موجودیت های Course وابستگی داشته باشد یا نداشته باشد. و به عکس موجودیت Course ممکن است با موجودیت های Person وابستگی داشته باشد یا نداشته باشد. به عبارت دیگر یک آموزگار ممکن است چند دوره آموزشی را تدریس کند، و یک دوره آموزشی ممکن است توسط چند آموزگار تدریس شود. (در این دیتابیس این ارتباط تنها متعلق به آموزگار است و دانشجو را به Course ارتباط نمی دهد. Student توسط جدول StudentGrade به Course مرتبط است.)

دیگر تفاوت بین دیاگرام دیتابیس و مدل داده در اضافه شدن بخش **Navigation Properties** به هر موجودیت است. یک Navigation Propertie برای هر موجودیت به موجودیت های وابسته اشاره دارد. به طور مثال خصوصیت Course (یا Courses در صورتی که در قسمت خصوصیت Entity Set Name موجودیت Course مقدار Courses وارد شده باشد. نکته ای که باید مورد توجه قرار گیرد این هست که برای دستیابی به مجموعه موجودیت ها از نام مجموعه یا Entity Set Name استفاده می شود. به طور پیش فرض این نام با نام موجودیت مورد نظر یکسان است.) در موجودیت Person یک مجموعه از موجودیت های Course که با Person مرتبط هستند را شامل می شود.

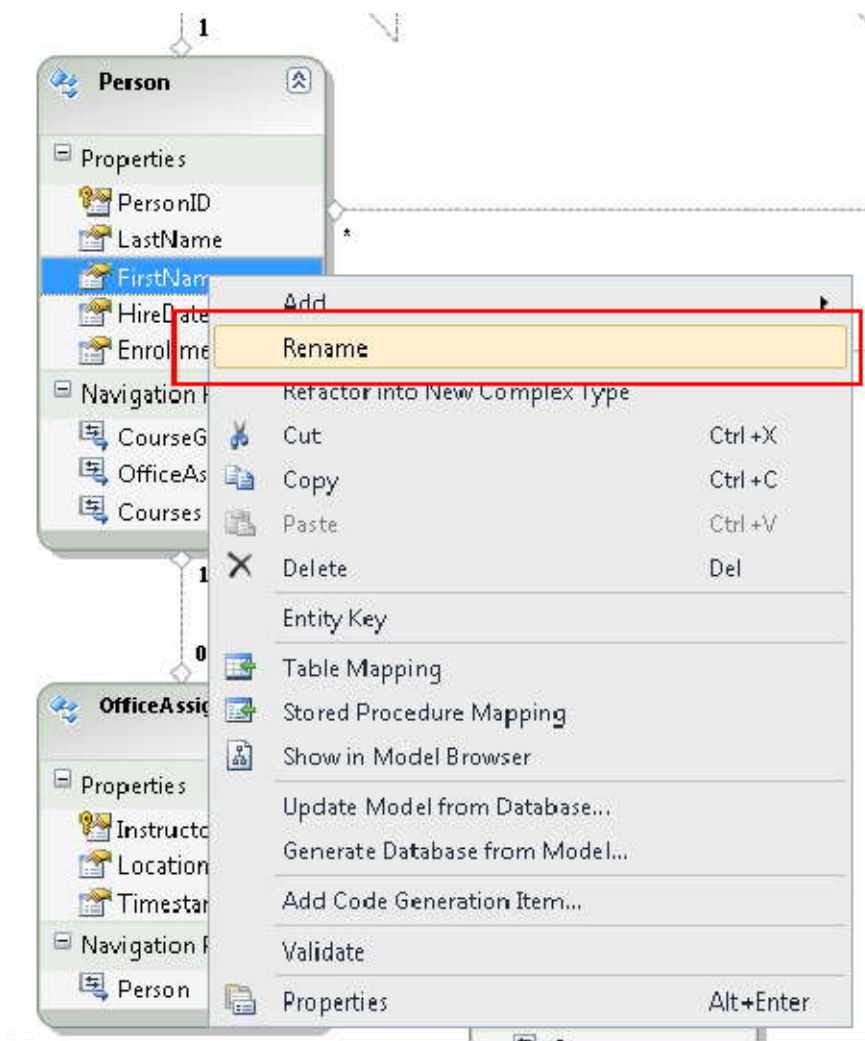


هنوز تفاوت دیگری بین دیاگرام دیتابیس و مدل داده، وجود دارد. و آن نبود جدول ارتباط **CourseInstructor** است که در دیتابیس برای ارتباط دادن جدول های **Person** و **Course** در ارتباط چند به چند استفاده شده. **navigation properties** شما را قادر خواهد کرد تا ارتباط موجودیت های **Course** را از موجودیت های **Person** و همچنین ارتباط موجودیت های **Person** را از موجودیت **Course** به دست آورید. بنابراین نیازی به ارائه جدول ارتباط در مدل داده نیست.

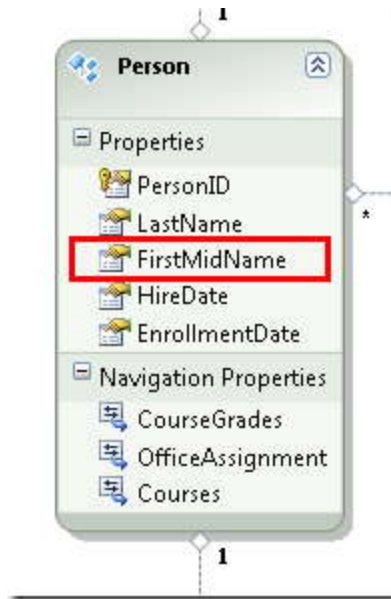


توجه داشته باشید در این بحث، ستون **FirstName** متعلق به جدول **Person** در واقع دو قسمت **first name** و **middle name** را شامل می شود. در صورتی که بخواهید نام این ستون را با نامی متناسب تغییر دهید، اما مدیر دیتابیس (DBA) تمایلی به تغییر دیتابیس نداشته باشد، شما می توانید در مدل داده نام آن را تغییر دهید، در حالی که دیتابیس همان دیتابیس قبلی و بدون اعمال تغییرات است.

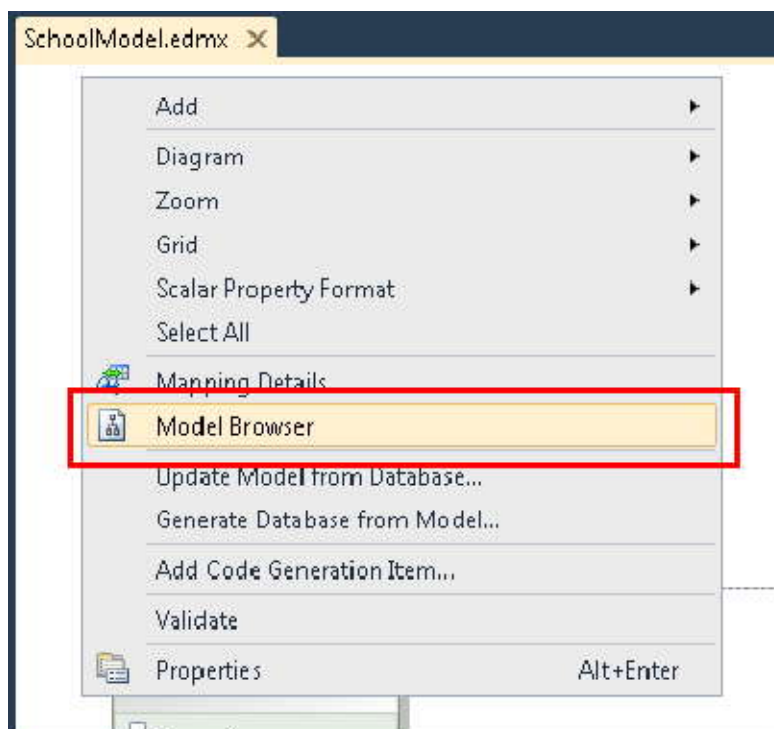
به این منظور، در محیط طراحی مدل داده، در موجودیت **Person** روی **FirstName** راست کلیک کرده و گزینه **Rename** را انتخاب کنید.



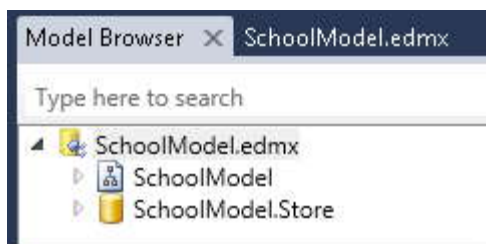
"FirstMidName" را به عنوان نام جدید تایپ نمائید. با انجام این کار، راه دستیابی به ستون مورد نظر، در قسمت کد، تغییر خواهد کرد، البته بدون اعمال تغییر در دیتابیس.



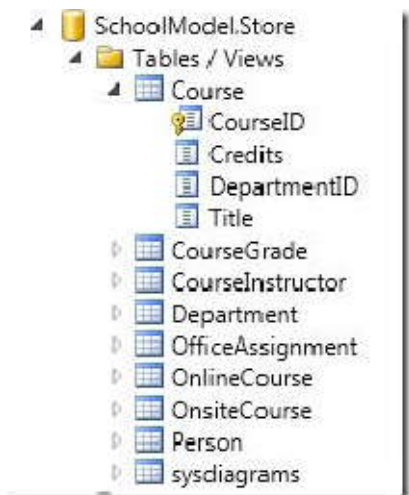
مرورگر مدل (model browser) راهی برای مشاهده ساختار دیتابیس و ساختار مدل داده و نگاشت بین این دو را فراهم می کند. برای مشاهده این موضوع در قسمت خالی از محیط طراحی مدل داده (entity designer) کلیک راست کرده و سپس روی **Model Browser** کلیک کنید.



Model Browser یک دید درختی (tree view) را نشان می دهد. (Model Browser ممکن است در قسمت **Explorer** Solution قرار گیرد). گره **SchoolModel** ساختار مدل داده، و گره **SchoolModel.Store** ساختار دیتابیس را ارائه می دهند.



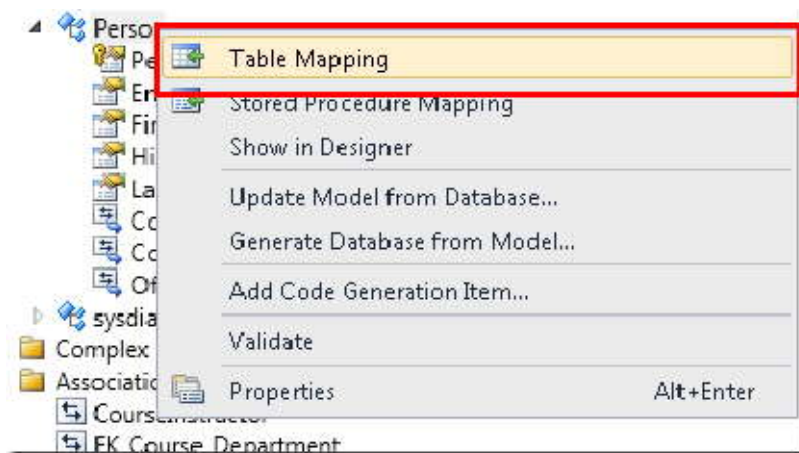
برای دیدن جدول ها **SchoolModel.Store** و سپس **Tables / Views** را گسترش دهید. برای نمونه **Course** را بسط گسترش دهید تا ستون های این جدول را مشاهده نمایید.



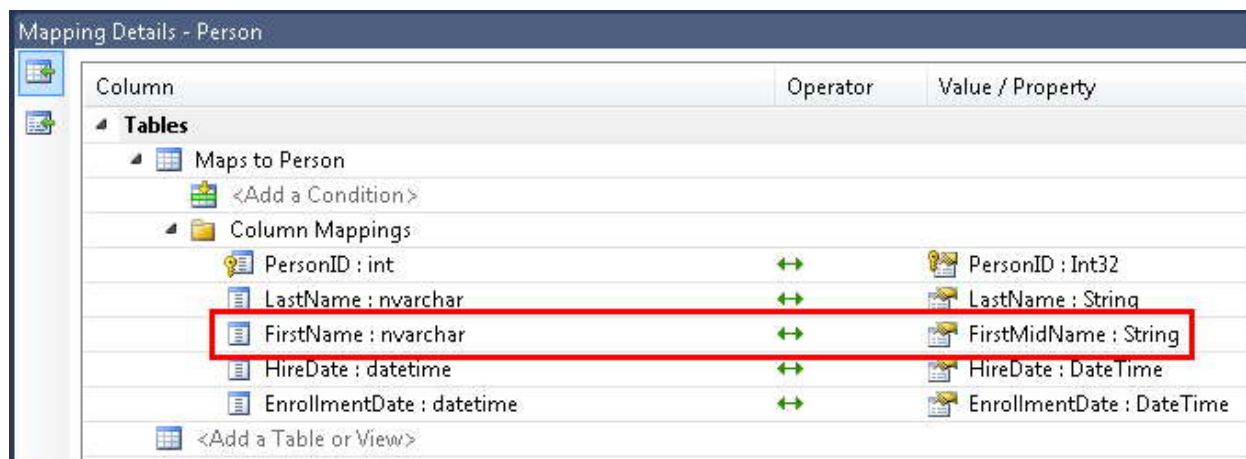
اکنون **SchoolModel** و سپس **Entity Types** و بعد گره **Course** را گسترش دهید تا موجودیت‌ها و خصوصیت‌های (properties) درون آن‌ها را مشاهده نمایید.



در **Entity designer** و **Model Browser** می‌توانید مشاهده کنید که چگونه **Entity Framework** اشیاء دو مدل را با هم مرتبط می‌کند. روی موجودیت **Person** کلیک راست و سپس **Table Mapping** را انتخاب کنید.



با این کار، پنجره **Mapping Details** باز می شود. توجه کنید، این پنجره به شما امکان مشاهده نگاشت بین ستون `FirstName` از دیتابیس و `FirstMidName` که در مدل داده آن را تغییر نام دادیم، را می دهد.



Entity Framework برای ذخیره اطلاعات در مورد دیتابیس، مدل داده و نگاشت بین آنها، از XML استفاده می کند. فایل `SchoolModel.edmx` در واقع یک فایل XML حاوی این اطلاعات است. طراح (designer) این اطلاعات را در یک قالب گرافیکی نمایش می دهد، اما این امکان برای شما فراهم است که فایل XML را با راست کلیک روی فایل `.edmx` در **Explorer** و کلیک روی **Open With** و سپس انتخاب **XML (Text) Editor** مشاهده نمایید. (designer و XML editor دو راه برای باز کردن و کار کردن با یک فایل هستند، بنابراین شما نمی توانید یک فایل باز در designer داشته باشید و همزمان آن فایل را در XML editor باز کنید).

اکنون شما یک وب سایت، یک دیتابیس و یک مدل داده ساخته اید. در مرحله بعد شما با استفاده از مدل داده (data model) و کنترل ASP.NET EntityDataSource، با داده ها کار خواهید کرد.

کنترل EntityDataSource

در این بخش با کنترل EntityDataSource که ASP.NET به منظور راحت کردن کار با Entity Framework data model فراهم کرده، کار خواهید کرد. در آینده شما یک کنترل GridView به منظور نمایش و ویرایش اطلاعات دانشجویها و یک کنترل DetailsView برای اضافه کردن دانشجویان جدید و یک کنترل DropDownList برای انتخاب کردن دانشکده¹ (که بعدا شما از آن برای نمایش Course های اختصاص داده شده، استفاده خواهید کرد) ایجاد خواهید کرد.

STUDENT LIST

	Name	EnrollmentDate	Number of Courses
Edit Delete	Abercrombie, Kim		0
Edit Delete	Barzdukas, Gytis	9/1/2005	2
Edit Delete	Justice, Peggy	9/1/2001	2

ADD NEW STUDENTS

FirstMidName	John
LastName	Smith
EnrollmentDate	1/1/2011
Insert Cancel	

COURSES BY DEPARTMENT

Select a department:

- Engineering
- English
- Economics
- Mathematics

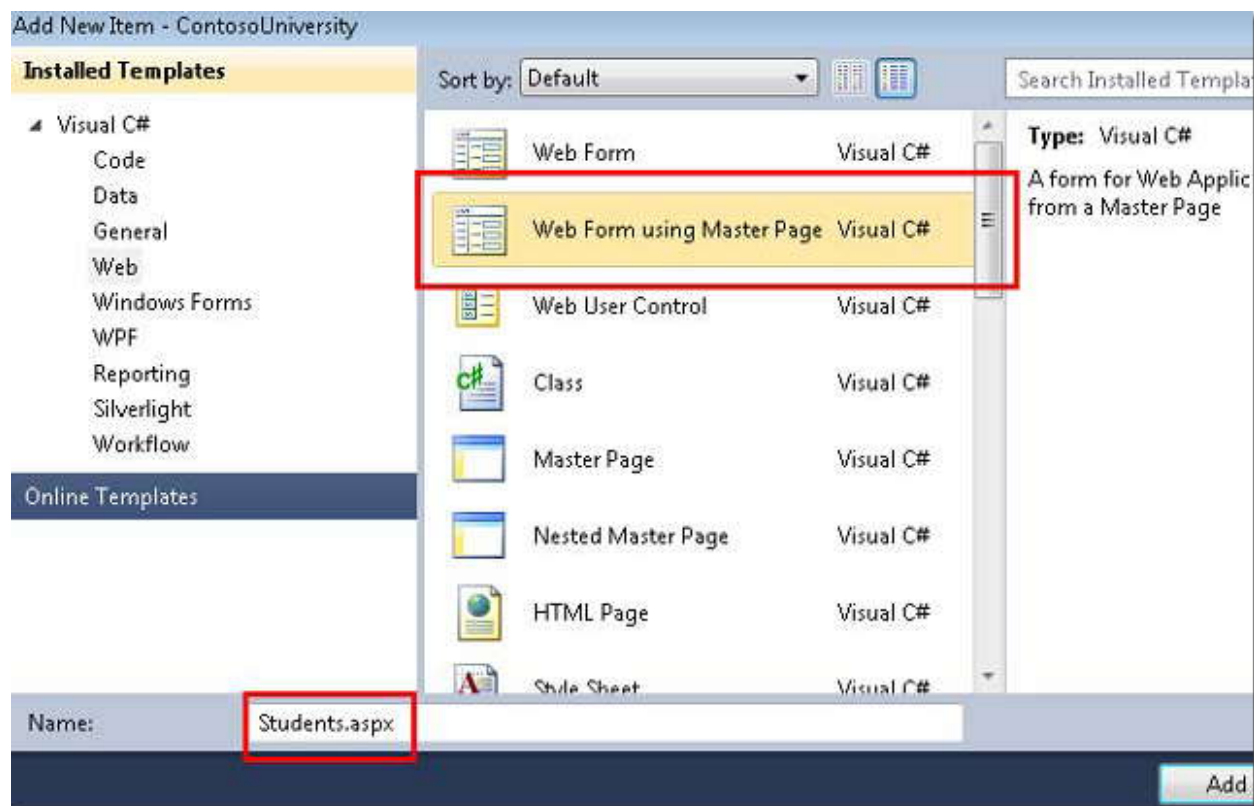
توجه فرمائید، در این برنامه input validation را به صفحه هایی که دیتابیس را Update می کنند، اضافه نخواهید کرد، و بعضی از error handling اینقدر قوی نیستند که ضرورت باشد در یک برنامه کاربردی استفاده شوند. در اینجا روی Entity Framework تمرکز می کنیم، و مطالب را طولانی نمی کنیم. برای مشاهده اطلاعات بیشتر در زمینه اضافه کردن این ویژگی ها

¹ Department

به برنامه می توانید به [Validating User Input in ASP.NET Web Pages](#) و [Error Handling in ASP.NET Pages and Application](#) مراجعه نمایید.

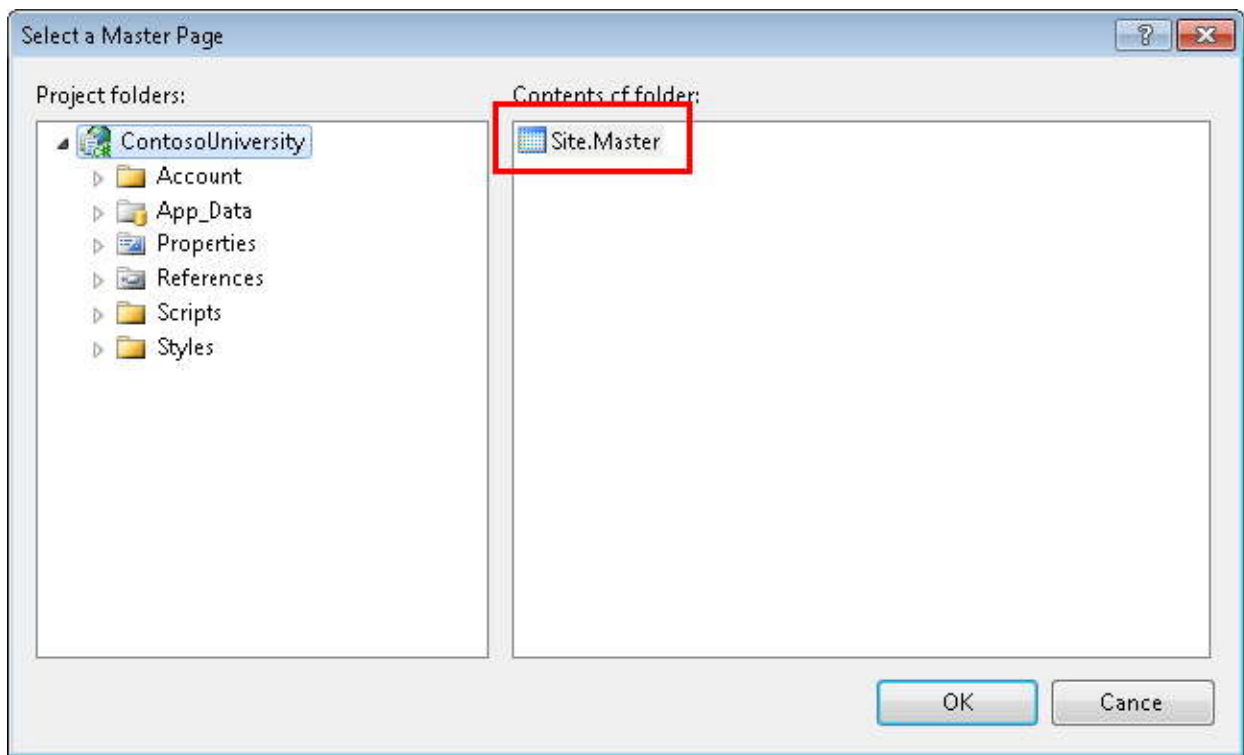
افزودن کنترل EntityDataSource و تنظیمات آن

با تنظیمات کنترل EntityDataSource به منظور خواندن موجودیت های Person از مجموعه موجودیت های People شروع می کنیم. پروژه ای که در بخش یک ایجاد کرده اید را باز کنید. با استفاده از قالب **Web Form using Master Page** یک صفحه وب جدید ایجاد کنید، و نام آن را Students.aspx قرار دهید.



Site.Master را به عنوان master page انتخاب کنید.

همه صفحاتی که در این پروژه ایجاد می کنید از این master page استفاده خواهد کرد.



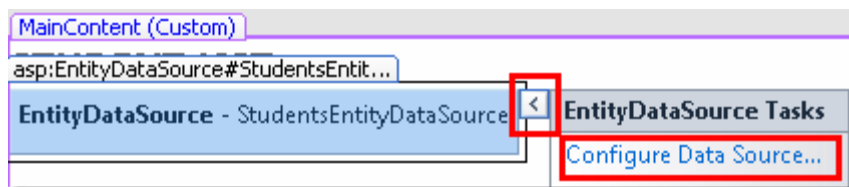
در قسمت **Source** این صفحه ، یک عنوان h2 به کنترل Content با نام Content2 ، همانند آنچه در زیر مشاهده می کنید، اضافه کنید:

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h2>Student List</h2>
</asp:Content>
```

از قسمت **Data** در **Toolbox** کنترل EntityDataSource را کشیده و زیر عنوان انداخته، و مقدار ID را به StudentsEntityDataSource تغییر دهید.

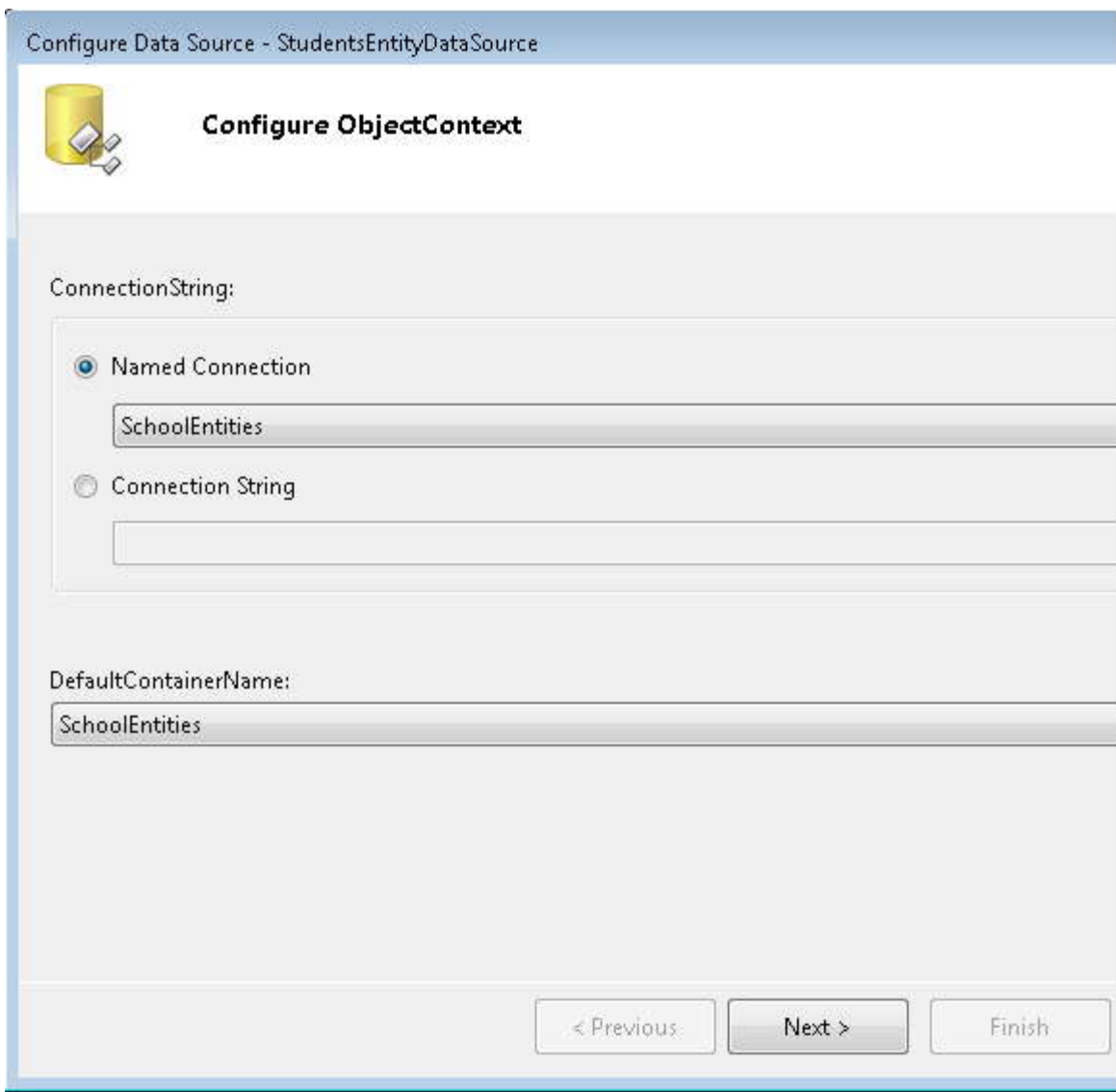
```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h2>Student List</h2>
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server">
</asp:EntityDataSource>
</asp:Content>
```

به محیط **Design** وارد شده، روی مربع حاوی فلش (smart tag) سمت راست کنترل **data source** کلیک کنید، سپس روی **Configure Data Source** کلیک کنید تا پنجره **Configure Data Source wizard** باز شود.

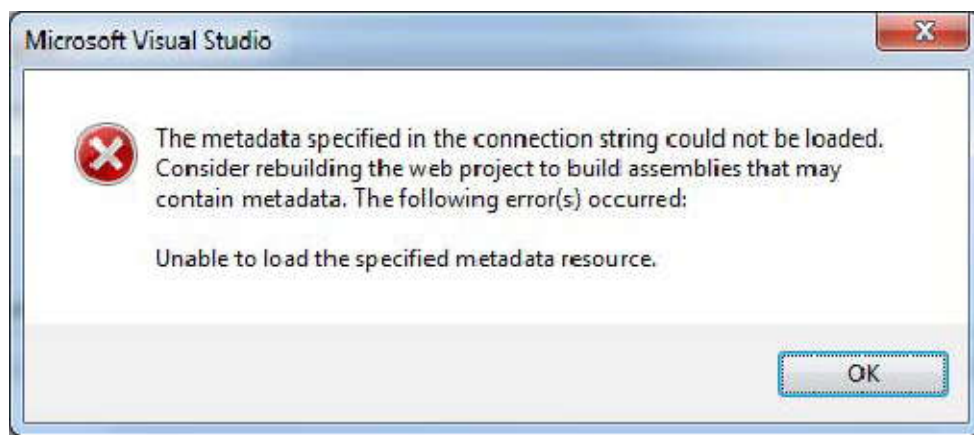


در مرحله **ConfigureObjectContext**، برای قسمت **Named Connection** مقدار **SchoolEntities** را انتخاب کنید، و برای قسمت **DefaultContainerName** نیز **SchoolEntities** انتخاب کنید، سپس روی **Next** کلیک کنید.

(در صورتی که بعد از انتخاب قسمت **named connection** با خطا مواجه شدید، و گزینه ای برای انتخاب در قسمت **DefaultContainerName** نبود، یک بار پروژه را **Debug** کنید، سپس دوباره تا این مرحله پیش بیایید)



توجه فرمائید، اگر در این مرحله با پیغام زیر مواجه شدید، قبل از ادامه باید یک بار پروژه را بسازید (build).



در مرحله **Configure Data Selection**، برای **EntitySetName** مقدار **Person** را انتخاب کنید. در قسمت **Select** از انتخاب گزینه **Select All** مطمئن شوید. سپس گزینه های **Update** و **Delete** را انتخاب کرده، بعد از آن روی **Finish** کلیک کنید.



Configure Data Selection

EntitySetName:

People

EntityTypeFilter:

(None)

Select:

- Select All (Entity Value)
- PersonID
- LastName
- FirstMidName
- HireDate
- EnrollmentDate

- Enable automatic inserts
- Enable automatic updates
- Enable automatic deletes

< Previous

Next >

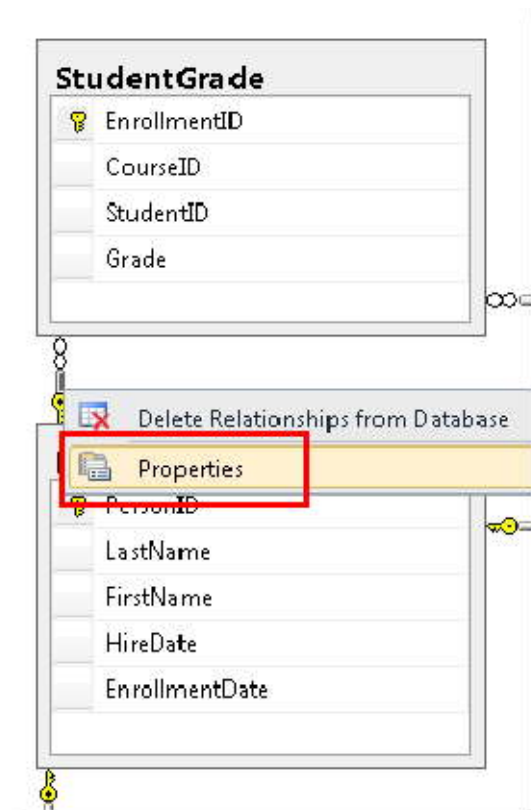
Finish

تنظیمات قوانین (Rule) دیتابیس برای عملیات حذف

صفحه ای را ایجاد می کنیم که به کاربران امکان حذف دانشجویان از جدول Person را می دهد، و دارای سه ارتباط با جدول های دیگر است. (Course, StudentGrade, OfficeAssignment). در حالت پیش فرض، دیتابیس اجازه حذف سطرهای جدول Person را نمی دهد، چون سطرهایی در جدول دیگری با آن ارتباط دارند. برای حل این موضوع می توانید به صورت دستی سطرهایی که با هم ارتباط دارند را حذف کنید، یا دیتابیس را طوری تنظیم کنید که هنگام حذف یک سطر Person به طور خودکار سطرهای مرتبط در جدول های دیگر حذف شوند. در این بخش برای رکوردهای دانشجویان، دیتابیس را طوری تنظیم میکنیم تا سطرهای مرتبط را به طور خودکار حذف کند. از آنجایی که دانشجویان تنها می توانند با سطرهای جدول StudentGrade ارتباط داشته باشند، کافی است تنها یک ارتباط از سه ارتباط جدول Person را تنظیم کنیم.

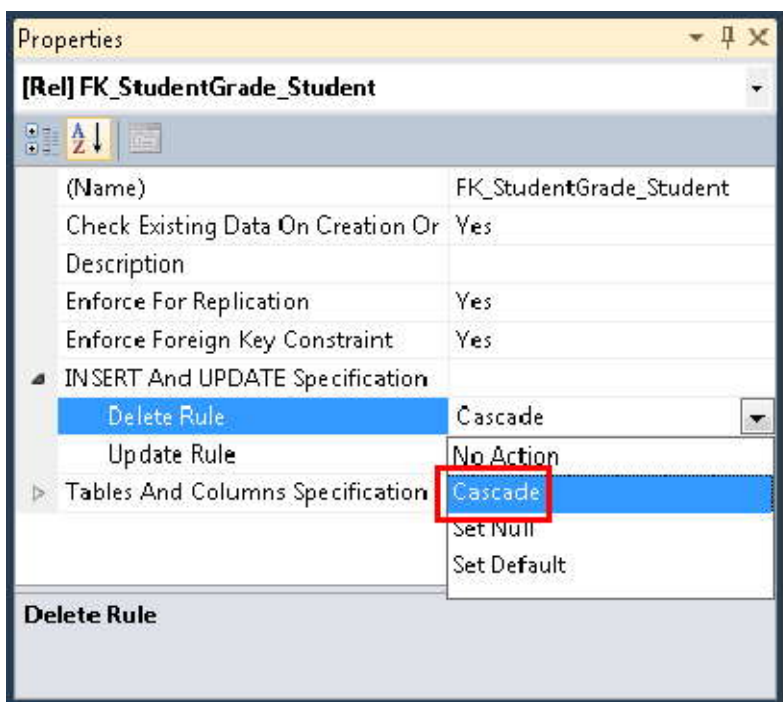
اگر از فایل *School.mdf* دانلود شده استفاده می کنید. نیاز به انجام این تنظیمات نیست، چون قبلا در آن اعمال شده.

در قسمت **Server Explorer**، دیگرام دیتابسی که قبلا ایجاد کرده اید را باز کنید. روی ارتباط¹ (خط بین دو جدول) بین Person و StudentGrade راست کلیک کرده و سپس **Properties** را انتخاب کنید.



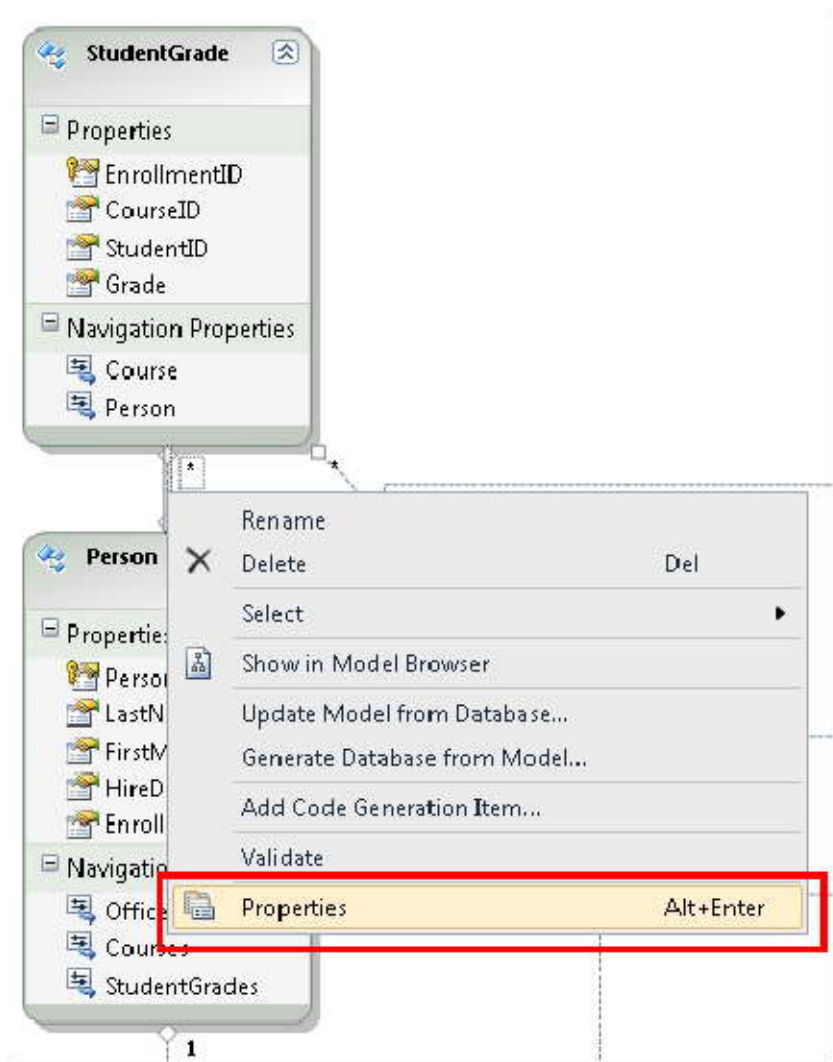
¹ Relationship

در پنجره **Properties** گزینه **INSERT and UPDATE Specification** را گسترش داده و برای خصوصیت **DeleteRule** مقدار **Cascade** را انتخاب کنید.

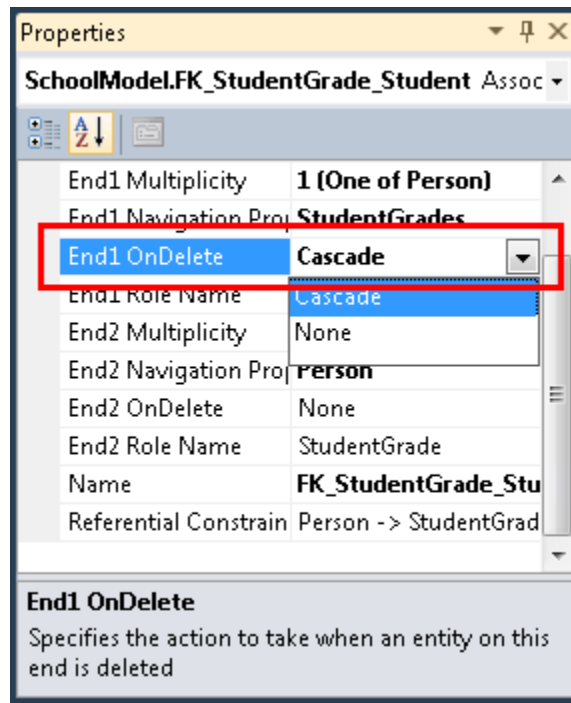


دیباگرام را ذخیره کرده و ببندید. اگر از شما سوال شد که آیا می خواهید دیتابیس را به روز رسانی کنید، روی **Yes** کلیک کنید. برای اطمینان از اینکه مدل، هماهنگی و سازگاری موجودیت هایی که در حافظه هستند را با موجودیت های درون دیتابیس حفظ می کند، شما باید قوانین همگام سازی را در مدل داده، تنظیم کنید.

را **SchoolModel.edmx** باز کنید، روی خط میان **Person** و **StudentGrade** راست کلیک کرده، و سپس **Properties** را انتخاب کنید.



در پنجره **Properties** مقدار **End1 OnDelete** را برابر **Cascade** قرار دهید.



فایل *SchoolModel.edmx* را ذخیره کرده و ببندید. سپس پروژه را دوباره بساز (rebuild).

در کل، وقتی دیتابیس تغییر میکند، شما انتخاب های مختلفی برای همگام سازی مدل دارید:

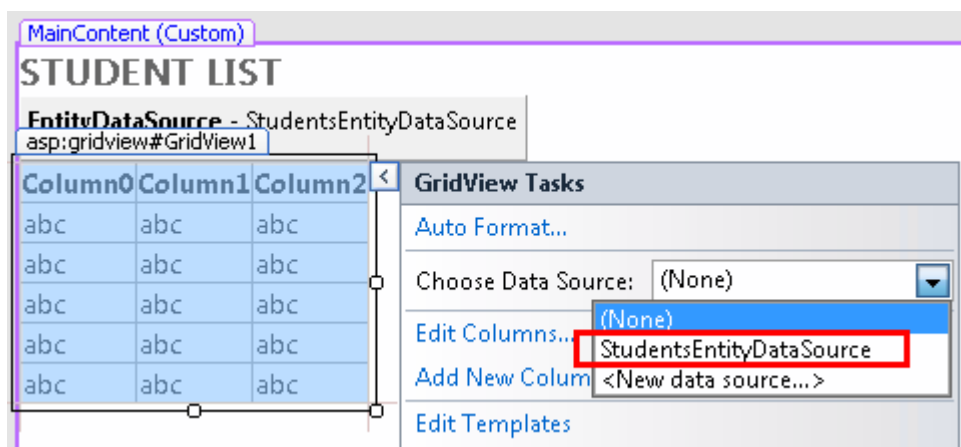
- برای تغییرات معین (مانند اضافه کردن^۱ یا نوسازی^۲ جدول ها، دیدها^۳، رویه های ذخیره شده^۴)، در محیط طراحی راست کلیک کرده و **Update Model from Database** را انتخاب کنید، تا designer تغییرات را به طور خودکار اعمال کند.
- ساخت دوباره مدل داده^۵
- انجام عملیات بروزرسانی^۶ به صورت دستی.

¹ inserting
² Refreshing
³ view
⁴ Stored procedure
⁵ Data model
⁶ Update

استفاده از کنترل GridView به منظور خواندن و بروزرسانی موجودیت ها

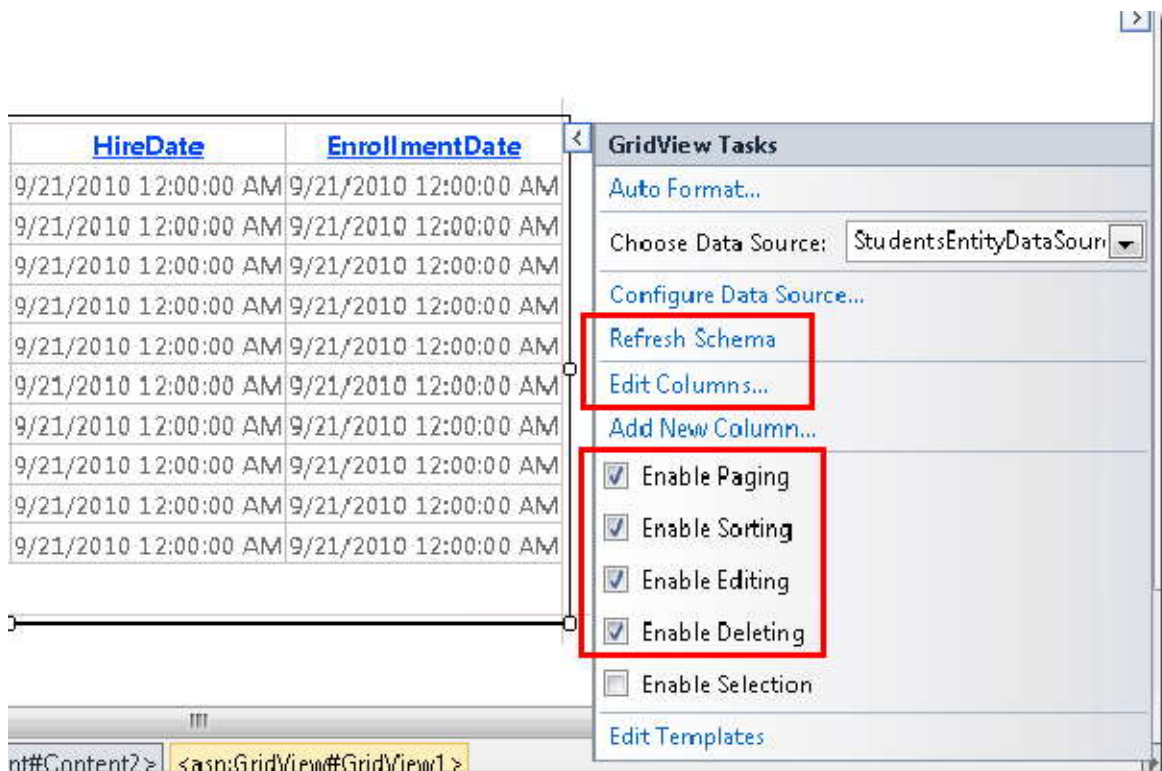
در این بخش شما یک کنترل GridView به منظور نمایش، بروزرسانی و حذف دانشجوها اضافه خواهید کرد.

در *Students.aspx* را باز کنید و به محیط **Design** وارد شوید. از بخش **Data** در **Toolbox** کنترل GridView را کشیده و در قسمت راست EntityDataSource قرار دهید. نام آن (ID) را به StudentsGridView تغییر داده و روی مربع سمت راست آن کلیک کرده و **StudentsEntityDataSource** را به عنوان Data Source انتخاب کنید.

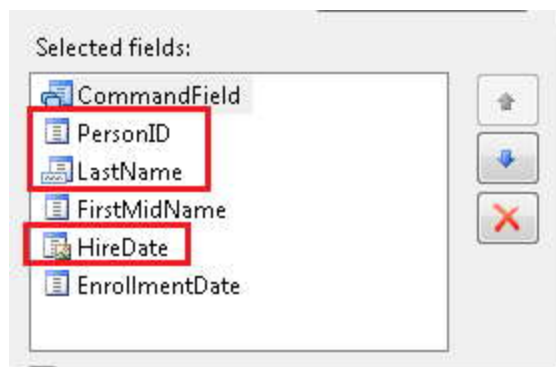


روی **Refresh Schema** کلیک کنید (اگر از شما تأیید خواست، روی **Yes** کلیک کنید)، سپس روی **Enable Paging**، **Enable**، **Enable Deleting** و **Enable Editing** کلیک کنید.

روی **Edit Columns** کلیک کنید.

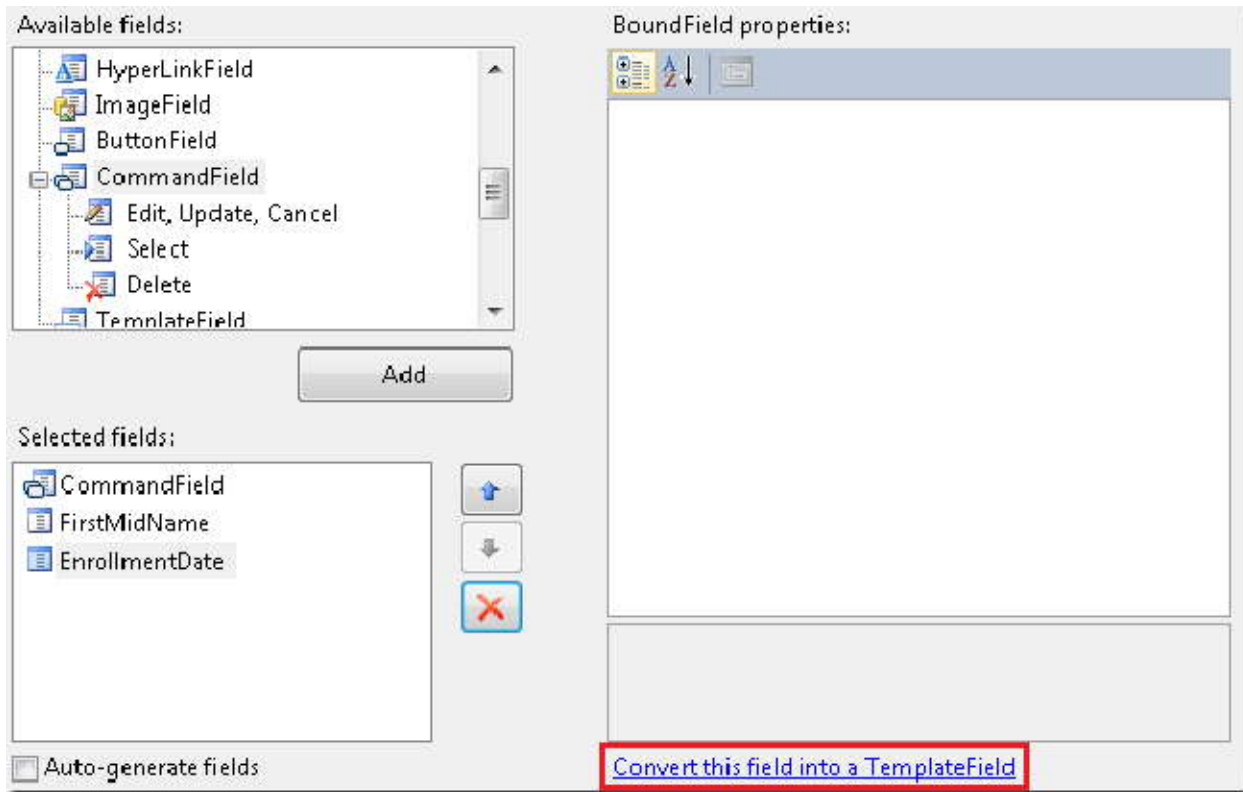


در قسمت **Selected fields**، ستون های **PersonID**، **LastName** و **HireDate** را حذف کنید. شما نباید کلید رکورد را به کاربران نمایش دهید، hiredate به دانشجو ربطی ندارد. (شما هر دو بخش نام را درون یک فیلد قرار خواهید داد، پس تنها به یکی از فیلدهای نام را نیاز دارید).



فیلد **FirstMidName** را انتخاب کرده و روی **Convert this field into a TemplateField** کلیک کنید.

همین کار را برای EnrollmentDate انجام دهید.



روی **OK** کلیک کرده و به محیط **Source** وارد شوید. تغییرات باقیمانده اگر در قسمت کد باشد راحتتر خواهد بود.

کد کنترل **GridView** اکنون شبیه مثال زیر است:

```
<asp:GridView ID="StudentsGridView" runat="server" AllowPaging="True"
    AllowSorting="True" AutoGenerateColumns="False" DataKeyNames="PersonID"
    <asp:TextBox ID="TextBox1" runat="server" Text='<%= Bind("FirstMidName")
%>'></asp:TextBox>
        </EditItemTemplate>
    </ItemTemplate>
    <asp:Label ID="Label1" runat="server" Text='<%=
Bind("FirstMidName") %>'></asp:Label>
        </ItemTemplate>
    </asp:TemplateField>
```

```

        <asp:TemplateField HeaderText="EnrollmentDate"
SortExpression="EnrollmentDate">
            <EditItemTemplate>
                <asp:TextBox ID="TextBox2" runat="server" Text='<##
Bind("EnrollmentDate") %>'></asp:TextBox>
            </EditItemTemplate>
            <ItemTemplate>
                <asp:Label ID="Label2" runat="server" Text='<##
Bind("EnrollmentDate") %>'></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>

```

اولین ستون بعد از فیلد command یک فیلد الگو (template field) است که در حال حاضر نام اول را نشان می دهد. برای فیلد الگوی مذکور، کد ساختیافته (markup) را به صورت زیر تغییر دهید:

```

<asp:TemplateField HeaderText="Name" SortExpression="LastName">
<EditItemTemplate>
<asp:TextBox ID="LastNameTextBox" runat="server" Text='<##
Bind("LastName")%>'></asp:TextBox>
<asp:TextBox ID="FirstNameTextBox" runat="server" Text='<## Bind("FirstMidName")
%>'></asp:TextBox>
</EditItemTemplate>
<ItemTemplate>
<asp:Label ID="LastNameLabel" runat="server" Text='<##
Eval("LastName")%>'></asp:Label>,
    <asp:Label ID="FirstNameLabel" runat="server" Text='<##
Eval("FirstMidName")%>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>

```

در حالت نمایش، دو برجسب^۱ first name و last name را نمایش داده می شوند. در حالت ویرایش، دو جعبه متنی^۲ قرار داده شده تا بتوانید، دو فیلد مذکور را ویرایش کنید. هنگامی که همراه کنترل Label در حالت نمایش اطلاعات، از Bind و Eval استفاده می کنید، دقیقا مانند زمانی است که با کنترل ASP.NET data source مستقیما به دیتابیس وصل می شوید. تنها تفاوت این است که به جای ستون های دیتابیس، خصوصیت های یک موجودیت را تعیین می کنید.

^۱ Label

^۲ Text box

ستون آخر یک فیلد الگو است که تاریخ ثبت نام را نمایش می دهد. کد ساختیافته این فیلد را به صورت زیر تغییر دهید:

```
<asp:TemplateField HeaderText="Enrollment Date" SortExpression="EnrollmentDate">
  <EditItemTemplate>
    <asp:TextBox ID="EnrollmentDateTextBox" runat="server"
Text='<%#Bind("EnrollmentDate", "{0:d}") %>'></asp:TextBox>
  </EditItemTemplate>
  <ItemTemplate>
    <asp:Label ID="EnrollmentDateLabel" runat="server"
Text='<%# Eval("EnrollmentDate", "{0:d}") %>'></asp:Label>
  </ItemTemplate>
</asp:TemplateField>
```

در هر دو حالت نمایش و ویرایش، رشته قالب `"{0,d}"`¹ باعث نمایش تاریخ به صورت "short date" خواهد شد. (ممکن است کامپیوتر شما به گونه ای تنظیم شده باشد که تاریخ را متفاوت با آنچه در عکس های این کتاب نمایش داده شده، نشان دهد).

توجه کنید، هر کدام از فیلدهای الگو در اینجا به صورت پیش فرض از `Bind` استفاده می کنند، اما شما در `ItemTemplate` به `Bind` را به عبارت `Eval` تغییر داده اید. عبارت `Bind` باعث می شود داده ها برای خصوصیت های² کنترل `GridView`، (در صورت نیاز شما به آنها) در کد(کد برنامه، برای مثال کدی که به زبان سی شارپ نوشته می شود و در سمت سرور اجرا می شود)، قابل دسترس باشند. در این مثال شما به دسترسی داده ها در قسمت کد نیازی ندارید، بنابراین می توانید از `Eval` که کارآمد تر است استفاده نمائید.

برای اطلاعات بیشتر به [Getting your data out of the data controls](#) مراجعه نمائید.

¹ Format string

² Propertie

باز بینی کد ساختیافته کنترل EntityDataSource به منظور بهبود کارایی

در قسمت کد ساختیافته کنترل EntityDataSource. صفات ConnectionString و DefaultContainerName را حذف کرده و با صفت ContextTypeName="ContosoUniversity.DAL.SchoolEntities" جایگزین نمائید. هر زمان یک EntityDataSource ایجاد می کنید باید این تغییر را اعمال کنید، مگر اینکه نیاز به استفاده از connection ای داشته باشید که متفاوت با connection موجود در کلاس object context است.

مزیت های استفاده از صفت ContextTypeName:

- کارایی بهتر. هنگامی که کنترل EntityDataSource مدل داده (data model) را با استفاده از ConnectionString و DefaultContainerName مقداردهی اولیه می کند، در هر درخواست¹، کارهای اضافه ای را برای بارگذاری متاداده ها² انجام می دهد. این کارها در صورت استفاده از صفت ContextTypeName انجام نمی شوند.
- Lazy loading (بارگذاری تنبل)، به طور پیش فرض در Entity Framework 4.0 در کلاس های object context تولید شده، این ویژگی روشن هست. این بدین معنی است که در صورت نیاز، خصوصیت navigation همراه با داده های مرتبط³ درست بارگذاری شده است. Lazy loading را در آینده شرح خواهیم داد.
- هر گونه خصوصیت سفارشی که شما در مورد کلاس شی Context (در این مثال کلاس SchoolEntities) اعمال کرده اید، برای کنترل هایی که از کنترل EntityDataSource استفاده می کنند، قابل دستیابی خواهد بود. سفارشی سازی کلاس شی Context موضوع پیشرفته ای است که در این کتاب به آن پرداخته نشده است.

برای اطلاعات بیشتر به [Extending Entity Framework Generated Types](#) مراجعه نمائید.

کد ساختیافته باید شبیه مثال زیر باشد (ترتیب خصوصیت ها ممکن است متفاوت باشد):

```
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="People"
EnableDelete="True" EnableUpdate="True">
</asp:EntityDataSource>
```

¹ Request

² metadata

³ related data

صفت `EnableFlattening` به ویژگی اشاره میکند که در نسخه های گذشته Entity Framework نیاز است چون ستون های کلید خارجی به عنوان خصیصه موجودیت نمایش داده نمی شوند. اما در نسخه کنونی استفاده از وابستگی های کلید خارجی امکان پذیر است. خصوصیت های کلید خارجی برای همه وابستگی ها به جز وابستگی چند به چند¹ نمایش داده می شوند. اگر موجودیت های شما دارای کلید خارجی هستند و دارای [Complex types](#) (انواع پیچیده) نیستند می توانید مقدار این صفت را `False` باقی گذارید.

برای اطلاعات بیشتر در مورد به [Flattening Objects](#) مراجعه نمائید.

صفحه را اجرا کنید، لیستی از دانشجویان و کارمندان را مشاهده می کنید (در بخش بعد فقط دانشجویها را فیلتر می کنیم). نام و نام خانوادگی با هم نمایش داده شده اند.

STUDENT LIST		
	Name	EnrollmentDate
Edit Delete	Abercrombie, Kim	
Edit Delete	Barzdukas, Gytis	9/1/2005
Edit Delete	Justice, Peggy	9/1/2001
Edit Delete	Fakhouri, Fadi	
Edit Delete	Harui, Roger	
Edit Delete	Li, Yan	9/1/2002
Edit Delete	Norman, Laura	9/1/2003
Edit Delete	Olivotto, Nino	9/1/2005
Edit Delete	Tang, Wayne	9/1/2005
Edit Delete	Alonso, Meredith	9/1/2002
1 2 3 4		

به منظور مرتب سازی²، روی نام ستون کلیک کنید.

روی **Edit** در هر سطر کلیک کنید. جعبه های متنی که شما می توانید `first name` و `last name` را در آنها تغییر دهید نمایش داده می شوند.

¹ Many to many

² Sort

STUDENT LIST

	Name	EnrollmentDate
Edit Delete	Abercrombie, Kim	
Update Cancel	Barzdukas, Gytis	9/1/2005
Edit Delete	Justice, Peggy	9/1/2001
Edit Delete	Fakhouri, Fadi	
Edit Delete	Harui, Roger	
Edit Delete	Li, Yan	9/1/2002
Edit Delete	Norman, Laura	9/1/2003
Edit Delete	Olivotto, Nino	9/1/2005
Edit Delete	Tang, Wayne	9/1/2005
Edit Delete	Alonso, Meredith	9/1/2002

1 2 3 4

دکمه Delete نیز کار می کند. روی دکمه Delete در سطری که دارای تاریخ ثبت نام (enrollment date) است کلیک کنید، آن سطر ناپدید می شود. (سطرهایی که تاریخ ثبت نام ندارند مربوط به آموزگاران هستند و ممکن است در هنگام حذف، خطای عدم حفظ جامعیت ارجاعی¹ را دریافت کنید. در بخش بعد شما لیست را طوری فیلتر خواهید کرد تا تنها دانشجویان را شامل شود).

نمایش داده ها از طریق خصوصیت Navigation

فرض می کنیم شما می خواهید بدانید یک دانشجو در چند دوره آموزشی ثبت نام کرده است. Entity Framework این اطلاعات را از طریق خصوصیت navigation (ناوبری) StudentGrades مربوط به موجودیت Person فراهم می کند. چون دیتابیس اجازه ثبت نام دانشجو در یک دوره را بدون ثبت نمره نمی دهد. برای توضیح این مطلب، می توانید فرض کنید در جدول StudentGrade سطری دارید که با یک سطر متناظر ثبت شده در جدول Course وابستگی دارد. (خصوصیت navigation مربوط به جدول Course تنها برای آموزگاران است).

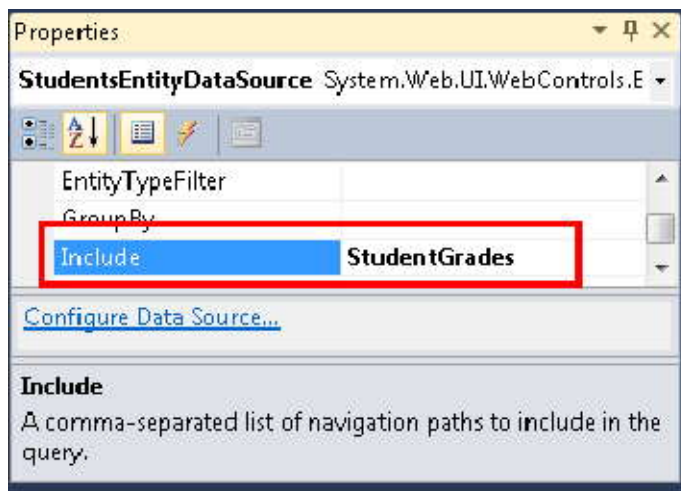
هنگامی که از صفت ContextTypeName متعلق به کنترل EntityDataSource استفاده می کنید، Entity Framework هنگامی که شما به خصوصیت navigation دسترسی پیدا می کنید، به طور خودکار اطلاعات را برای این خصوصیت برمی

¹ referential integrity

گرداند. این حالت *Lazy loading* نامیده می شود. گرچه، ممکن است ناکارا باشد، چون این کار باعث می شود، هر بار که اطلاعات بیشتری نیاز است یک اتصال با دیتابیس شکل بگیرد. اگر شما برای هر موجودیتی که توسط کنترل `EntityDataSource` برگردانده می شود نیاز به دستیابی به داده هایی از خصوصیت `navigation` دارید. کارا تر آن است که داده های مرتبط¹ را همراه با خود موجودیت، از طریق یک ارتباط با دیتابیس برگردانید. این حالت *eager loading* نامیده می شود. این کار را با تعیین حالت `eager loading` برای خصوصیت `navigation`، از طریق تنظیم خصوصیت `Include` کنترل `EntityDataSource` انجام دهید.

در `Students.aspx` شما باید تعداد دوره ها را برای هر دانشجو نشان دهید، بنابراین `eager loading` بهترین انتخاب است. اگر همه دانشجویان را لیست میکنید اما تعداد دوره ها را برای اندکی از آنها نمایش می دهید(ممکن بود نیاز به کد نویسی باشد)، `Lazy loading` ممکن است انتخاب بهتری باشد.

در `Students.aspx` را باز کنید، به محیط طراحی² وارد شوید، `StudentsEntityDataSource` را انتخاب کنید و در پنجره `Properties` مقدار خصوصیت `Include` را برابر `StudentGrade(s)` (در صورتی که `Entity Set Name` این موجودیت `StudentGrades` باشد) قرار دهید (اگر می خواهید چند خصوصیت `Navigation` داشته باشید، می توانید نام آنها را با کاما از هم جدا کنید به طور مثال: `StudentGrade, Courses`)



¹ related data

² design

به قسمت **Source** این صفحه بروید، در کنترل **StudentsGridView** بعد از آخرین عنصر **asp:TemplateField** یک **template field** مانند مثال زیر را اضافه نمائید:

```
<asp:TemplateField HeaderText="Number of Courses">
<ItemTemplate>
<asp:Label ID="Label1" runat="server" Text='<%# Eval("StudentGrade.Count")
%>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
```

در عبارت **Eval**، می توانید به خصوصیت **Navigation** مربوط به **StudentGrade** رجوع کنید. چون این خصوصیت شامل یک مجموعه است، و دارای یک خصوصیت **Count** است که می توان تعداد دوره هایی که دانشجو در آنها ثبت نام کرده را نشان داد. در بخش بعد شما چگونگی نمایش داده ها از خصوصیت **Navigation** ای که به جای مجموعه، شامل یک موجودیت است را خواهید دید. (توجه کنید شما نمی توانید از عنصر **BoundField** برای نمایش داده های خصوصیت **Navigation** استفاده کنید)

صفحه را اجرا کنید، تعداد دوره هایی که دانشجو در آن ثبت نام کرده است را مشاهده خواهید کرد.

STUDENT LIST			
	Name	EnrollmentDate	Number of Courses
Edit Delete	Abercrombie, Kim		0
Edit Delete	Barzdukas, Gytis	9/1/2005	2
Edit Delete	Justice, Peggy	9/1/2001	2

استفاده از کنترل **DetailsView** برای درج¹ موجودیت ها

گام بعد ایجاد یک صفحه حاوی کنترل **DetailsView** است که به شما امکان اضافه کردن دانشجوی جدید را می دهد. مرورگر را ببندید و یک صفحه وب جدید که از **Site.Master** به عنوان **master page** استفاده می کند ایجاد کنید. نام آن را **StudentsAdd.aspx** قرار دهید و به قسمت **Source** وارد شوید.

¹ Insert

کد ساختیافته زیر را با کد کنترل Content با نام Content2 جایگزین کنید:

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h2>Add New Students</h2>
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EnableInsert="True" EntitySetName="Person">
</asp:EntityDataSource>
<asp:DetailsView ID="StudentsDetailsView" runat="server"
DataSourceID="StudentsEntityDataSource" AutoGenerateRows="False"
DefaultMode="Insert">
<Fields>
<asp:BoundField DataField="FirstMidName" HeaderText="First Name"
SortExpression="FirstMidName"/>
<asp:BoundField DataField="LastName" HeaderText="Last Name"
SortExpression="LastName"/>
<asp:BoundField DataField="EnrollmentDate" HeaderText="Enrollment Date"
SortExpression="EnrollmentDate"/>
<asp:CommandField ShowInsertButton="True"/>
</Fields>
</asp:DetailsView>
</asp:Content>
```

این کد ساختیافته یک کنترل EntityDataSource ایجاد می کند که مانند کنترلی است که در صفحه *Students.aspx* ایجاد کردید، به جز اینکه مورد اخیر، قابلیت درج دارد. زمانی که درون کنترل **GridView**، فیلدهای مقید¹ مربوط به کنترل **DetailsView** کد نویسی می شوند، همانند زمانی خواهد بود که data control مستقیماً به دیتابیس وصل شود اما با این تفاوت که به خصوصیت های یک موجودیت رجوع می کنند. در این مثال، کنترل **DetailsView** تنها برای درج سطرها استفاده شده، بنابراین شما می توانید حالت پیش فرض را **Insert** قرار دهید.

صفحه را اجرا کرده و یک دانشجوی جدید اضافه نمائید.



ADD NEW STUDENTS	
FirstMidName	John
LastName	Smith
EnrollmentDate	1/1/2011
Insert Cancel	

¹ Bound field

زمانی که دانشجوی جدید اضافه می کنید، چیزی به شما نشان داده نخواهد شد. اما وقتی `Students.aspx` را اجرا کنید، اطلاعات دانشجوی جدید را مشاهده خواهید کرد.

نمایش داده ها در Drop-Down List

در گام های بعد، شما یک کنترل `DropDownList` را با استفاده از یک کنترل `EntityDataSource` به یک `entity set` مفید^۱ می سازید. در این بخش شما کار زیادی با این لیست نخواهید داشت. گرچه در بخش های آینده از این لیست برای انتخاب یک دانشکده^۲ به منظور نمایش دوره های وابسته به آن استفاده خواهید کرد.

یک صفحه وب جدید با نام `Courses.aspx` ایجاد کنید. در قسمت `source` یک عنوان، به کنترل `Content` که با نام `Content2` مشخص شده است، اضافه کنید:

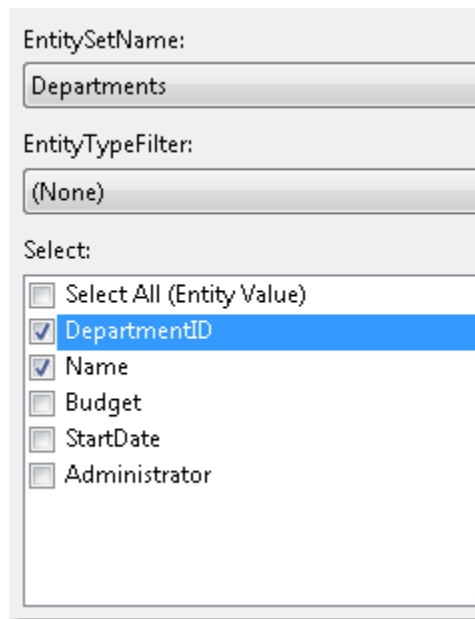
```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h2>Courses by Department</h2>
</asp:Content>
```

در قسمت طراحی^۳ یک کنترل `EntityDataSource` همانند قبل به صفحه اضافه کنید و نام آن را `DepartmentsEntityDataSource` قرار دهید. `Department(s)` را به عنوان مقدار `EntityType` انتخاب کنید. تنها خصوصیت های `DepartmentID` و `Name` را انتخاب کنید.

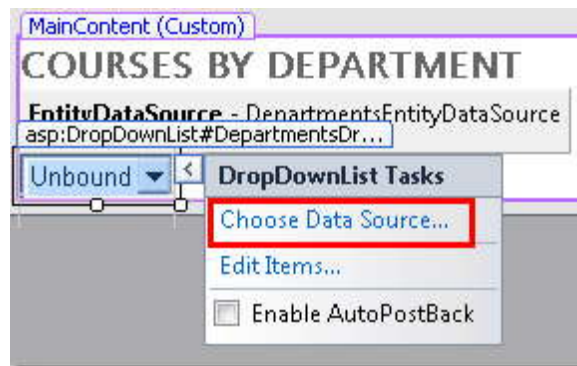
¹ Databind

² department

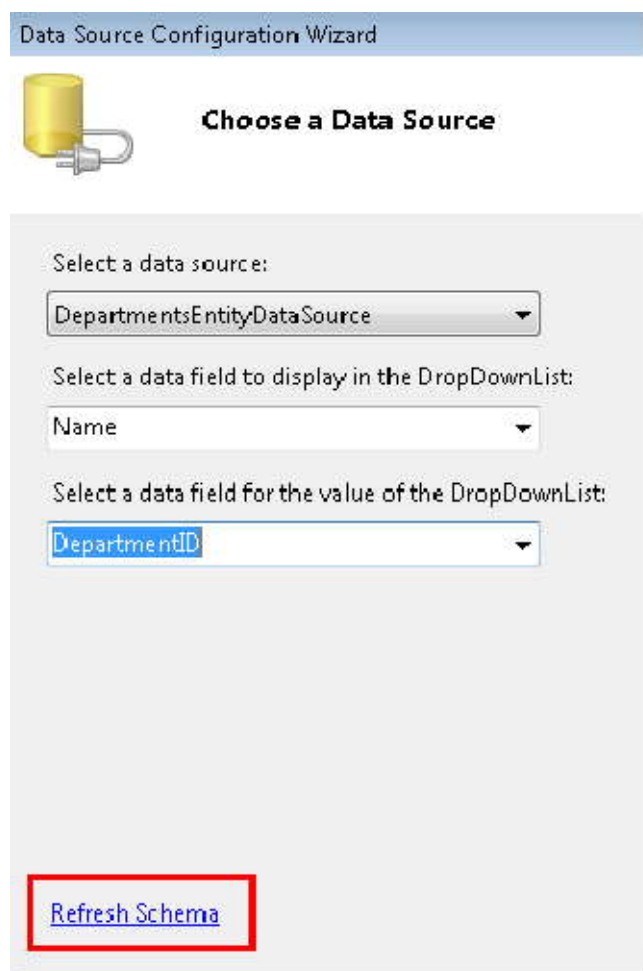
³ Design



از قسمت **Standard** در **Toolbox** یک کنترل **DropDownList** را کشیده و در صفحه قرار دهید، نام آن را **DepartmentsDropDownList** گذارده، روی مربع کناری آن کلیک کرده و **Choose Data Source** را انتخاب کنید تا **DataSource Configuration Wizard** باز شود.



در مرحله **Choose a Data Source**، به عنوان **DataSource** مقدار **DepartmentsEntityDataSource** را انتخاب کنید و روی **Refresh Schema** کلیک کنید، فیلد **Name** را برای **data field to display** و فیلد **DepartmentID** را برای **data field for the value** انتخاب کنید. روی **OK** کلیک کنید.



روشی که شما با استفاده از Entity Framework داده ها را به کنترل خاصی مقید کردید، مانند روشی است که در مورد کنترل های ASP.NET DataSource انجام می شود، با این تفاوت که شما در Entity Framework، موجودیت ها¹ و خصیصه های موجودیت ها² را مشخص می کنید.

¹ Entities

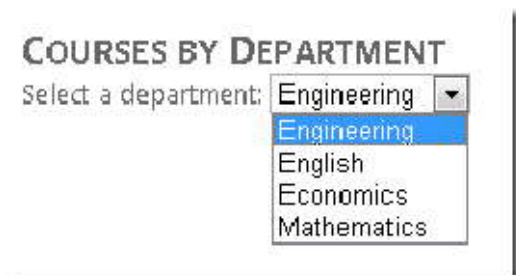
² Entity Properties

به قسمت **Source** وارد شده و "Select a department:" را دقیقاً قبل از کنترل **DropDownList** اضافه کنید.

```
Select a department:
<asp:DropDownList ID="DepartmentsDropDownList" runat="server"
    DataSourceID="DepartmentsEntityDataSource" DataTextField="Name"
    DataValueField="DepartmentID">
</asp:DropDownList>
```

به عنوان یادآوری، کد markup مربوط به کنترل **EntityDataSource** را تغییر داده و صفت های **ConnectionString** و **DefaultContainerName** را با صفت **ContextTypeName="ContosoUniversity.DAL.SchoolEntities"** جایگزین نمائید. معمولاً بهتر است، که قبل از هر تغییری در کد کنترل **EntityDataSource**، کنترل **data-bound** را که به یک کنترل **data source** پیوند داده شده ایجاد کنید، زیرا بعد از اعمال تغییرات در کد، عمل **designer Refresh Schema** را در مورد کنترل **data-bound** انجام نخواهد داد.

صفحه را اجرا و یک دانشکده را از لیست کشویی انتخاب کنید.



معرفی کنترل **EntityDataSource** در اینجا به اتمام میرسد. کار با این کنترل به طور کلی تفاوتی با کنترل های دیگر **ASP.NET** **data source** ندارد، به جز اینکه شما به جای ارجاع به ستون ها و جدول ها به موجودیت ها و خصیصه های موجودیت ها ارجاع می کنید. تنها استثنا زمانی است که می خواهید به خصوصیت **Navigation** دسترسی پیدا کنید. در بخش بعد مشاهده خواهید کرد که نحوه استفاده از کنترل **EntityDataSource** هنگام مرتب نمودن، گروه بندی و فیلتر کردن داده ها ممکن است با نحوه استفاده از کنترل های **data source** دیگر متفاوت باشد.

فیلتر کردن، مرتب سازی و گروه بندی داده ها

در قسمت های قبل شما از کنترل `EntityDataSource` برای نمایش و ویرایش داده ها استفاده کردید. در این بخش شما عملیات فیلتر کردن^۱، مرتب کردن^۲ و گروه بندی^۳ را انجام خواهید داد. برای انجام این کارها، آنچه شما در تنظیمات خصوصیت های^۴ کنترل `EntityDataSource` انجام می دهید، با آنچه در مورد دیگر کنترل های `Data Source` انجام می دهید متفاوت است. به هر حال، شما امکان استفاده از کنترل `QueryExtender` به منظور کم کردن این تفاوت ها را خواهید داشت.

در اینجا صفحه `Students.aspx` را به منظور انجام عملیات فیلتر برای دانشجویان، و مرتب سازی و جستجو بر اساس `name`، تغییر خواهیم داد. همچنین صفحه `Courses.aspx` را به منظور نمایش دوره ها برای دانشکده های انتخابی و جستجوی دوره ها بر اساس `Name`، تغییر خواهیم داد. در پایان، آمار دانشجویان را در صفحه `About.aspx` اضافه خواهیم کرد.

STUDENT LIST			
	Name	Enrollment Date	Number of Courses
Edit Delete	Barzdukas, Gytis	9/1/2005	2
Edit Delete	Justice, Peggy	9/1/2001	2
Edit Delete	Li, Yan	9/1/2002	2

¹ Filtering

² Ordering

³ Grouping

⁴ Properties

COURSES BY DEPARTMENT

Select a Department

ID	Title	Credits
2021	Composition	3
2030	Poetry	2
2042	Literature	4

COURSES BY NAME

Enter a course name

Department	ID	Title	Credits
Economics	4041	Macroeconomics	3
Economics	4022	Microeconomics	3
Economics	4063	new course	5

STUDENT BODY STATISTICS

Date of Enrollment	Students
9/1/2000	2
9/1/2001	5
9/1/2002	3
1/30/2003	1
9/1/2003	3
9/1/2004	5
9/1/2005	6
1/1/2011	1

FIND STUDENTS BY NAME

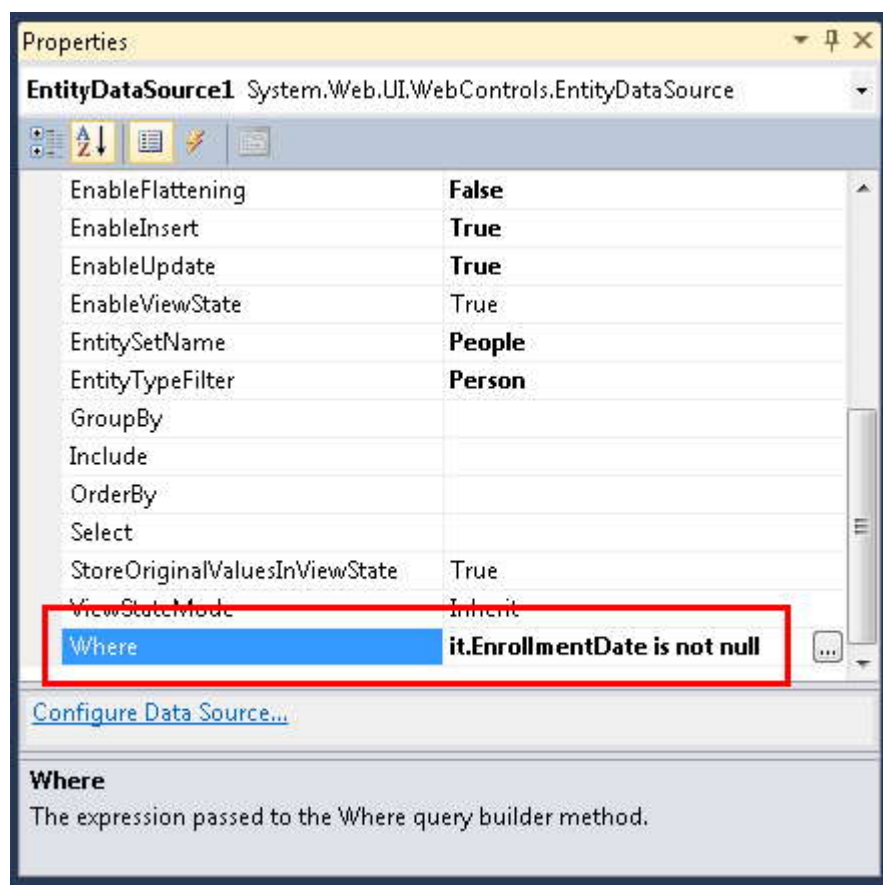
Enter any part of the name

Name	EnrollmentDate
Barzdukas, Gytis	9/1/2005
Justice, Peggy	9/1/2001
Tang, Wayne	9/1/2005

استفاده از خصوصیت "Where" در EntityDataSource برای فیلتر کردن داده ها

صفحه *Students.aspx* را که قبلا ایجاد کرده اید، باز کنید. همان طور که در حال حاضر تنظیم شده، کنترل *GridView* در این صفحه همه نام های مجموعه موجودیت های *Person* را نمایش می دهد. اما شما می خواهید که فقط دانشجویان را نشان دهد، به این منظور می توانید آنها را با انتخاب موجودیت های *Person* ای که تاریخ ثبت نام آنها هیچ مقدار¹ نیست، این کار را انجام دهید.

به محیط **Design** وارد شوید و کنترل *EntityDataSource* را انتخاب کنید. در پنجره **Properties**، مقدار خصوصیت *Where* را *it.EnrollmentDate is not null* قرار دهید.



¹ Null

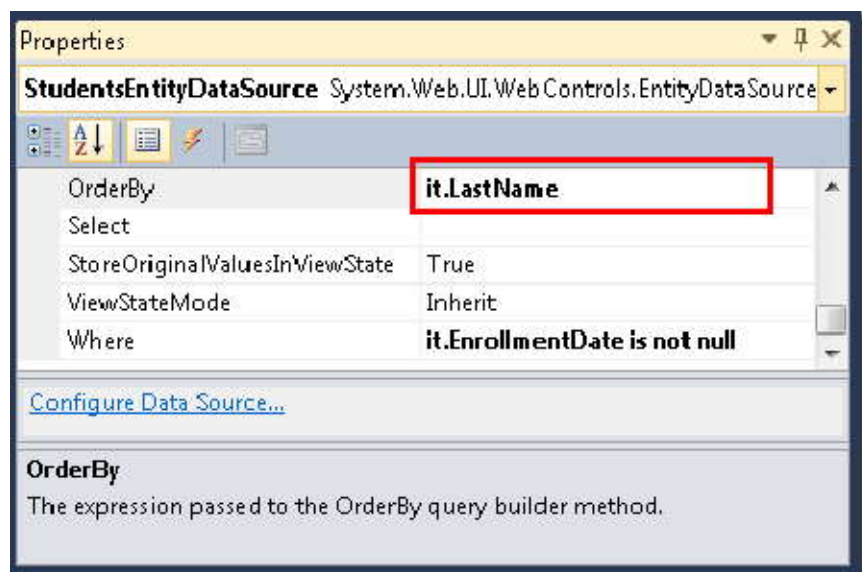
ساختار زبانی که شما در مورد خصوصیت Where کنترل EntityDataSource استفاده می کنید، Entity SQL است. Entity SQL شبیه Transact-SQL است، اما به جای کار با اشیاء دیتابیس، برای کار با موجودیت ها، سفارشی شده است. در عبارت it.EnrollmentDate is not null، کلمه it به موجودیتی که توسط کوئری برگردانده شده اشاره می کند. بنابراین، it.EnrollmentDate به خصوصیت EnrollmentDate موجودیت Person که کنترل EntityDataSource برمی گرداند، اشاره می کند.

صفحه را اجرا کنید. اکنون لیست دانشجویان تنها شامل دانشجویان است.(سطری که تاریخ ثبت نام ندارد نشان داده نشده است).

STUDENT LIST			
	Name	EnrollmentDate	Number of Courses
Edit Delete	Barzdukas, Gytis	9/1/2005	2
Edit Delete	Justice, Peggy	9/1/2001	2
Edit Delete	Li, Yan	9/1/2002	2

استفاده از خصوصیت "OrderBy" کنترل EntityDataSource برای مرتب کردن داده ها

شما همچنین می خواهید در لحظه اولی که لیست نمایش داده می شود، بر اساس نام مرتب شده باشد. در صفحه *Students.aspx* که اکنون در محیط Design باز است، و کنترل EntityDataSource انتخاب شده است، در پنجره Properties خصوصیت OrderBy را برابر it.LastName قرار دهید.



صفحه را اجرا کنید، اکنون لیست دانشجویان بر اساس نام خانوادگی مرتب شده است.

STUDENT LIST

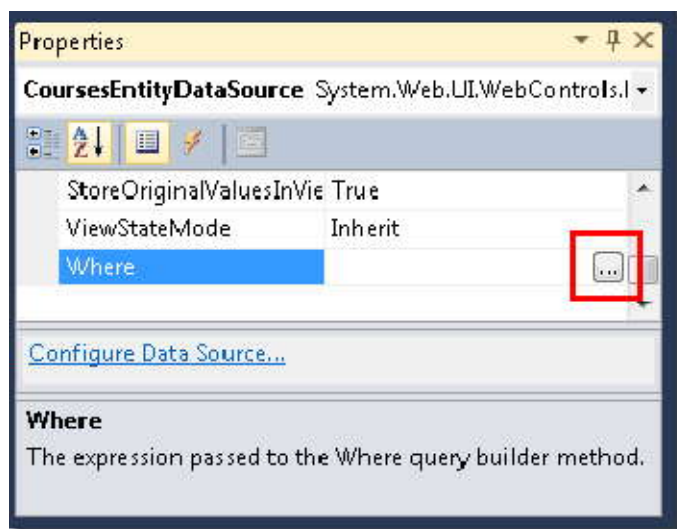
	<u>Name</u>	<u>EnrollmentDate</u>	Number of Courses
Edit Delete	Alexander, Carson	9/1/2005	3
Edit Delete	Alonso, Meredith	9/1/2002	1
Edit Delete	Anand, Arturo	9/1/2003	2

استفاده از Control Parameter برای تنظیم کردن خصوصیت "Where"

همان طور که در مورد کنترل های data source دیگر، می توانید مقدار پارامترها را به خصوصیت **Where** بفرستید. در صفحه *Courses.aspx* که قبلا ایجاد کرده اید، شما می توانید این روش را برای نمایش دوره ها استفاده کنید. دوره هایی که به دانشکده ها وابستگی دارند و یک کاربر آنها را از لیست انتخاب می کند.

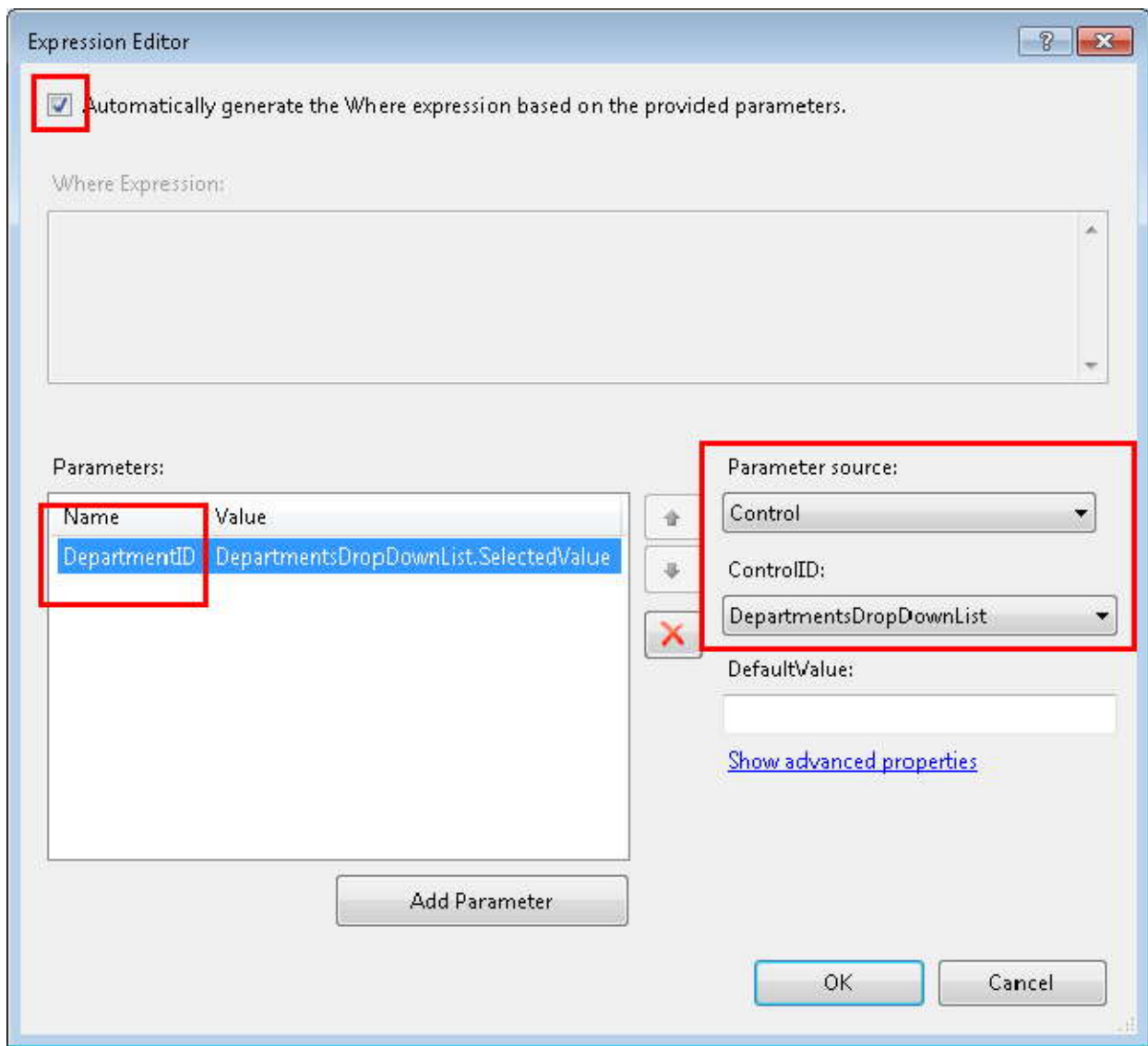
Courses.aspx را باز کرده و به قسمت **Design** وارد شوید. یک کنترل `EntityDataSource` دیگر به نام `CoursesEntityDataSource` اضافه کنید. آن را به مدل `SchoolEntities` وصل¹ کنید، و `Course(s)` را برای مقدار `EntitySetName` انتخاب کنید.

در پنجره **Properties** روی دکمه روبروی خصوصیت **Where** کلیک کنید (مطمئن شوید قبل از استفاده از پنجره **Properties**، کنترل `CoursesEntityDataSource` انتخاب شده است).

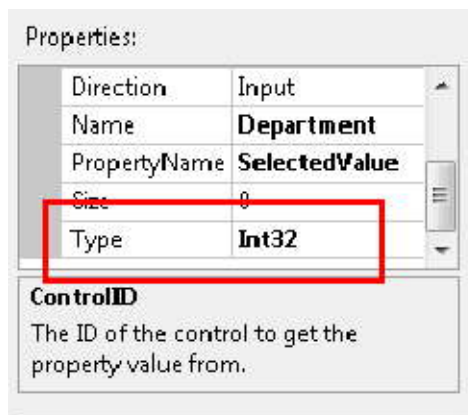


کادر محاوره ای **Expression Editor** نمایش داده شده. در این کادر، **Automatically generate the Where expression based on the provided parameters** را انتخاب کنید. سپس روی **Add Parameter** کلیک کنید. نام پارامتر جدید را `DepartmentID` قرار دهید، **Control** را برای مقدار `Parameter source`، و `DepartmentsDropDownList` را برای مقدار `ControlID` انتخاب کنید.

¹ Connect



روی **Show advanced properties** کلیک کنید، سپس در پنجره **Properties** کادر محاوره ای **Expression Editor**، خصوصیت **Type** را به **Int32** تغییر دهید.



بعد از انجام این کار، روی OK کلیک کنید.

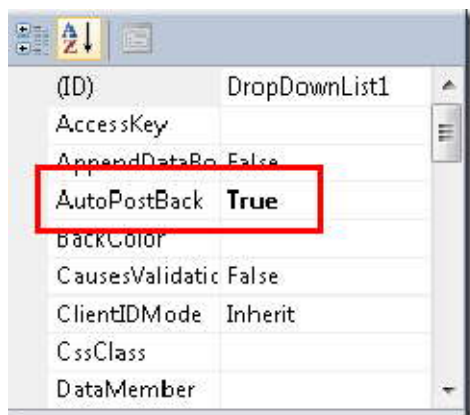
زیر قسمت drop-down list، یک کنترل GridView را به صفحه اضافه کرده و نام آن را CoursesGridView قرار دهید. آن را به کنترل CoursesEntityDataSource وصل¹ کنید، روی Refresh Schema کلیک کنید، روی Edit Columns کلیک کنید، و ستون DepartmentID را حذف کنید.

کد ساختیافته کنترل GridView مانند مثال زیر خواهد شد.

```
<asp:GridView ID="CoursesGridView" runat="server" AutoGenerateColumns="False"
  DataKeyNames="CourseID" DataSourceID="CoursesEntityDataSource">
  <Columns>
    <asp:BoundField DataField="CourseID" HeaderText="CourseID" ReadOnly="True"
      SortExpression="CourseID" />
    <asp:BoundField DataField="Title" HeaderText="Title"
      SortExpression="Title" />
    <asp:BoundField DataField="Credits" HeaderText="Credits"
      SortExpression="Credits" />
  </Columns>
</asp:GridView>
```

شما نیاز دارید، زمانی که کاربر دانشکده انتخابی را تغییر می دهد، دوره های وابسته به طور خودکار تغییر کنند. برای اینکه این اتفاق بیفتد drop-down list (لیست کشویی) را انتخاب کنید و در پنجره Properties مقدار True را برای خصوصیت AutoPostBack انتخاب کنید.

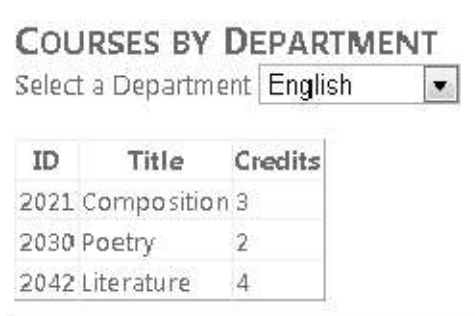
¹ Connect



اکنون کار شما با محیط طراحی¹ تمام شده است. به محیط **Source** وارد شوید و صفت های **ConnectionString** و **DefaultContainer** را با **CoursesEntityDataSource** کنترل به کنترل **ContextTypeName="ContosoUniversity.DAL.SchoolEntities"** جایگزین کنید. بعد از این کار کد ساخت یافته کنترل مورد نظر مانند مثال زیر خواهد بود:

```
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
    AutoGenerateWhereClause="True"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="Course" Where="">
    <WhereParameters>
        <asp:ControlParameter ControlID="DepartmentsDropDownList"
            Name="DepartmentID"
            PropertyName="SelectedValue" Type="Int32" />
    </WhereParameters>
</asp:EntityDataSource>
```

صفحه را اجرا کرده و از **drop-down list** برای انتخاب دانشکده دیگر استفاده کنید. تنها، دوره هایی که با دانشکده های انتخابی آورده می شوند، در کنترل **GridView** نمایش داده می شوند.



¹ designer

استفاده از خصوصیت "GroupBy" کنترل EntityDataSource برای گروه‌بندی داده‌ها

فرض بر این است که دانشگاه کانتوسو می‌خواهد آماری را از دانشجویان، در صفحه About نشان دهد. به طور خاص، می‌خواهد تفکیکی از تعداد دانشجویان را بر اساس تاریخ ثبت نام نشان دهد.

صفحه *About.aspx* را باز کنید، به قسمت کد وارد شوید، محتوی کنترل BodyContent را با "Student Body Statistics" که بین دو تک h2 قرار گرفته جایگزین نمایید.

```
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
<h2>Student Body Statistics</h2>
</asp:Content>
```

بعد از عنوان صفحه، یک کنترل EntityDataSource با نام StudentStatisticsEntityDataSource اضافه نمایید. آن را به SchoolEntities وصل کنید، مجموعه موجودیت Person را انتخاب، و کادر Select را آنگونه که هست رها کنید. خصوصیت های زیر را در پنجره Properties تنظیم کنید:

- برای اینکه تنها دانشجویان فیلتر شوند، مقدار خصوصیت Where را به صورت `it.EnrollmentDate is not null` قرار دهید.
- برای گروه‌بندی نتیجه بر اساس تاریخ ثبت نام، مقدار خصوصیت GroupBy را به صورت `it.EnrollmentDate` قرار دهید.
- برای انتخاب تاریخ ثبت نام و تعداد دانشجوها، مقدار خصوصیت Select را به صورت `it.EnrollmentDate, Count(it.EnrollmentDate) AS NumberOfStudents` قرار دهید.
- برای مرتب‌سازی نتایج بر اساس تاریخ ثبت نام، مقدار خصوصیت OrderBy را به صورت `it.EnrollmentDate` قرار دهید.

در قسمت **Source**، خصوصیت های `ConnectionString` و `DefaultContainer` را با خصوصیت `ContextTypeName` جایگزین نمائید. کد ساختیافته کنترل `EntityDataSource` اکنون باید شبیه مثال زیر باشد.

```
<asp:EntityDataSource ID="StudentStatisticsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="Person" GroupBy="it.EnrollmentDate"
    OrderBy="it.EnrollmentDate"
    Select="it.EnrollmentDate, Count(it.EnrollmentDate) AS NumberOfStudents"
    Where="it.EnrollmentDate is not null">
</asp:EntityDataSource>
```

ساختار نحوی خصوصیت های `Select`، `GroupBy` و `Where` شبیه زبان `Transact-SQL` است به جز کلمه کلیدی `it` که مشخص کننده موجودیت جاری است.

کد زیر را برای افزودن کنترل `GridView` به منظور نمایش داده ها اضافه کنید:

```
<asp:GridView ID="StudentStatisticsGridView" runat="server"
    AutoGenerateColumns="False" DataSourceID="StudentStatisticsEntityDataSource">
<Columns>
<asp:BoundField DataField="EnrollmentDate" DataFormatString="{0:d}"
    HeaderText="Date of Enrollment"
    ReadOnly="True" SortExpression="EnrollmentDate"/>
<asp:BoundField DataField="NumberOfStudents" HeaderText="Students"
    ReadOnly="True" SortExpression="NumberOfStudents"/>
</Columns>
</asp:GridView>
```

صفحه را اجرا کنید، لیستی که تعداد دانشجویان را بر اساس تاریخ ثبت نام نشان می دهد مشاهده خواهید کرد.

STUDENTBODY STATISTICS	
Date of Enrollment	Students
9/1/2000	2
9/1/2001	5
9/1/2002	3
1/30/2003	1
9/1/2003	3
9/1/2004	5
9/1/2005	6
1/1/2011	1

استفاده از کنترل QueryExtender به منظور فیلتر کردن و مرتب کردن

کنترل **QueryExtender** امکانی را برای فیلتر کردن و مرتب کردن در قسمت markup در اختیار شما قرار می دهد. ساختار نحوی آن بستگی به سیستم مدیریت دیتابسی (DBMS)¹ که شما استفاده می کنید دارد. همچنین این ساختار نحوی مستقل از Entity Framework است. اما یک استثنا وجود دارد که آن ساختاری نحوی است که شما در مورد خصوصیت های Navigation استفاده می کنید، و این ساختار منحصر به Entity Framework است.

در این بخش از آموزش، شما از یک کنترل QueryExtender برای فیلتر و مرتب کردن داده ها استفاده خواهید کرد، که یکی از فیلد های OrderBy آن خصوصیت Navigation خواهد بود.

(اگر ترجیح می دهید به جای کد نویسی از کوئری هایی که به طور خودکار توسط کنترل EntityDataSource تولید می شود استفاده کنید، می توانید با اداره کردن رخداد QueryCreated این کار را انجام دهید. همچنین کنترل QueryExtender، به این صورت، کوئری های کنترل EntityDataSource را توسعه می دهد.)

در صفحه *Courses.aspx* کد ساختیافته زیر را بعد از کدی که قبلا اضافه کردید، اضافه کنید. هدف از این کار، ایجاد عنوان، یک جعبه متنی برای وارد کردن عبارت جستجو، یک دکمه جستجو، و یک کنترل EntityDataSource است که به مجموعه موجودیت های Course(s) مقید شده است،

```
<h2>Courses by Name</h2>
Enter a course name
<asp:TextBox ID="SearchTextBox" runat="server"></asp:TextBox>
<asp:Button ID="SearchButton" runat="server" Text="Search"/>
<br/><br/>
<asp:EntityDataSource ID="SearchEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Courses"
Include="Department">
</asp:EntityDataSource>
```

¹ database management system

توجه کنید که خصوصیت Include کنترل EntityDataSource برابر Department تنظیم شده است. در دیتابیس، جدول Course حاوی ستون Department نیست، بلکه دارای کلید خارجی DepartmentID است. اگر می‌خواستید مستقیماً نام department را همراه با اطلاعات دوره، با یک کوئری، از دیتابیس دریافت کنید، مجبور بودید جدول‌های Course و Department را با هم Join کنید. با قرار دادن مقدار خصوصیت include برابر Department، مشخص می‌کنید که Entity Framework باید کار گرفتن یک موجودیت مرتبط Department را زمانی که با یک موجودیت Course ارتباط دارد انجام دهد. سپس موجودیت Department در خصوصیت Department navigation متعلق به موجودیت Course ذخیره می‌شود. (به طور پیش فرض کلاس SchoolEntities که توسط طراح (Designer) مدل داده ایجاد شده، داده‌های مرتبط را در زمان نیاز و زمانی که شما کنترل data source را به آن کلاس مقید¹ می‌کنید، بر می‌گرداند، بنابراین تنظیم خصوصیت Include ضروری نیست. گرچه این کار باعث بهبود کارایی این صفحه خواهد شد، چون در غیر این صورت Entity Framework اتصال‌های جداگانه‌ای به دیتابیس به منظور برگرداندن داده‌ها، برای موجودیت‌های Course و موجودیت‌های مرتبط Department ایجاد خواهد کرد.)

بعد از کنترل EntityDataSource که اکنون ایجاد شد، کد زیر را برای ایجاد کنترل QueryExtender ای که به کنترل EntityDataSource مقید می‌شود، اضافه کنید:

```
<asp:QueryExtender ID="SearchQueryExtender" runat="server"
TargetControlID="SearchEntityDataSource">
<asp:SearchExpression SearchType="StartsWith" DataFields="Title">
<asp:ControlParameter ControlID="SearchTextBox"/>
</asp:SearchExpression>
<asp:OrderByExpression DataField="Department.Name" Direction="Ascending">
<asp:ThenBy DataField="Title" Direction="Ascending"/>
</asp:OrderByExpression>
</asp:QueryExtender>
```

عنصر SearchExpression مشخص می‌کند که شما دوره‌ای را می‌خواهید که عنوان آن با مقدار وارد شده در جعبه متنی برابر است. هر کاراکتری که وارد می‌شود، عملیات مقایسه انجام خواهد شد، چون خصوصیت SearchType برابر StartsWith تعیین شده است.

عنصر OrderByExpression مجموعه نتایج را بر اساس عنوان دوره همراه نام دانشکده مرتب می‌کند. توجه کنید چگونه نام دانشکده تعیین شده: Department.Name چون وابستگی موجودیت Course با موجودیت Department از نوع یک به یک است. خصوصیت Navigation مربوط به Department شامل یک موجودیت Department است. (اگر این وابستگی از

¹ Bound

نوع یک به چند بود، خصوصیت **Navigation** شامل یک مجموعه می بود.) برای گرفتن نام دانشکده، باید خصوصیت **Name** مربوط به موجودیت **Department** را مشخص کنید.

در پایان یک کنترل **GridView** برای نمایش لیست دوره ها اضافه کنید:

```
<asp:GridView ID="SearchGridView" runat="server" AutoGenerateColumns="False"
DataKeyNames="CourseID" DataSourceID="SearchEntityDataSource" AllowPaging="true">
<Columns>
<asp:TemplateField HeaderText="Department">
<ItemTemplate>
<asp:Label ID="Label2" runat="server" Text='<%=# Eval("Department.Name")
%>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
<asp:BoundField DataField="CourseID" HeaderText="ID"/>
<asp:BoundField DataField="Title" HeaderText="Title"/>
<asp:BoundField DataField="Credits" HeaderText="Credits"/>
</Columns>
</asp:GridView>
```

فیلد اول یک **template field** است که نام دانشکده را نمایش می دهد. همان گونه که در کنترل **QueryExtender** دیدید، عبارت مربوط به انعقاد داده¹، **Department.Name** را مشخص می کند.

صفحه را اجرا کنید. نمایش ابتدایی، لیستی از دوره ها را نشان می دهد که بر اساس دانشکده و سپس عنوان دوره مرتب شده اند.

¹ Databinding

COURSES BY DEPARTMENT

Select a Department

ID	Title	Credits
2021	Composition	3
2030	Poetry	2
2042	Literature	4

COURSES BY NAME

Enter a course name

Department	ID	Title	Credits
Economics	4041	Macroeconomics	3
Economics	4022	Microeconomics	3
Economics	4063	new course	5

"m" را وارد کرده و روی **Search** کلیک کنید تا همه دوره هایی که با "m" شروع می شوند را ببینید (جستجو به بزرگ و کوچک بودن حروف حساس نیست).

COURSES BY DEPARTMENT

Select a Department

ID	Title	Credits
2021	Composition	3
2030	Poetry	2
2042	Literature	4

COURSES BY NAME

Enter a course name

Department	ID	Title	Credits
Economics	4041	Macroeconomics	3
Economics	4022	Microeconomics	3

استفاده از عملگر "Like" برای مرتب سازی داده ها

شما می توانید نتایج مشابه به انواع جستجوی `StartsWith`، `Contains` و `EndsWith` مربوط به کنترل `QueryExtender` را با استفاده از عملگر `Like` در خصوصیت `Where` مربوط به کنترل `EntityDataSource` به دست آورید. در این قسمت شما نحوه استفاده از عملگر `Like` را به منظور جستجوی یک دانشجو بر اساس نام خواهید دید.

`Students.aspx` را در محیط `Source` باز کنید. بعد از کنترل `GridView` کد ساختیافته زیر را اضافه کنید:

```
<h2>Find Students by Name</h2>
Enter any part of the name
<asp:TextBox ID="SearchTextBox" runat="server" AutoPostBack="true"></asp:TextBox>
<asp:Button ID="SearchButton" runat="server" Text="Search"/>
<br/>
<br/>
<asp:EntityDataSource ID="SearchEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Person"
Where="it.EnrollmentDate is not null and (it.FirstMidName Like '%' + @StudentName +
'%' or it.LastName Like '%' + @StudentName + '%')">
<WhereParameters>
<asp:ControlParameter ControlID="SearchTextBox" Name="StudentName" PropertyName="Text"
Type="String" DefaultValue=""/>
</WhereParameters>
</asp:EntityDataSource>
<asp:GridView ID="SearchGridView" runat="server" AutoGenerateColumns="False"
DataKeyNames
="PersonID"
DataSourceID="SearchEntityDataSource" AllowPaging="true">
<Columns>
<asp:TemplateField HeaderText="Name" SortExpression="LastName, FirstMidName">
<ItemTemplate>
<asp:Label ID="LastNameFoundLabel" runat="server" Text='<%= Eval("LastName")
%>'></asp:Label>,
<asp:Label ID="FirstNameFoundLabel" runat="server" Text='<%= Eval("FirstMidName")
%>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Enrollment Date" SortExpression="EnrollmentDate">
<ItemTemplate>
<asp:Label ID="EnrollmentDateFoundLabel" runat="server" Text='<%=
Eval("EnrollmentDate", "{0:d}") %>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>
```


این کد شبیه چیزی است که قبلا مشاهده کردید، غیر از مقدار خصوصیت **Where**. قسمت دوم عبارت **Where** زیر رشته جستجو را مشخص می کند (**LIKE %FirstMidName% or LIKE %LastName%**) که نام و نام خانوادگی را برای آنچه که در جعبه متنی وارد شده جستجو خواهد کرد.

صفحه را اجرا کنید. در ابتدا شما همه دانشجویها را مشاهده خواهید کرد، چون مقدار پیش فرض برای پارامتر **StudentName** برابر **"%"** تعیین شده.

FIND STUDENTS BY NAME

Enter any part of the name

Name	EnrollmentDate
Barzdukas, Gytis	9/1/2005
Justice, Peggy	9/1/2001
Li, Yan	9/1/2002

حرف **"g"** را در جعبه متنی وارد کرده د روی **Search** کلیک کنید. شما لیستی از دانشجویها را که نام یا نام خانوادگی آنها حاوی **"g"** است را مشاهده خواهید کرد.

FIND STUDENTS BY NAME

Enter any part of the name

Name	EnrollmentDate
Barzdukas, Gytis	9/1/2005
Justice, Peggy	9/1/2001
Tang, Wayne	9/1/2005

تا اینجا شما عملیات نمایش، به روزرسانی، فیلتر کردن، مرتب سازی و گروهبندی داده ها را در مورد جدول ها به صورت تکی انجام داده اید. در بخش بعد کار با داده های دارای ارتباط را شروع خواهید کرد. (سناریوهای master-detail)

کار با داده های وابسته¹

در بخش قبل از کنترل **EntityDataSource** برای فیلتر کردن، مرتب کردن و گروه بندی داده ها استفاده کردید. در این بخش شما داده های وابسته (مرتبط) را نمایش خواهید داد و بروزرسانی خواهید کرد.

در این بخش، صفحه اساتید، که لیست اساتید در آن نمایش داده می شود را ایجاد خواهید کرد. زمانی که یک آموزگار را انتخاب می کنید، لیستی از دوره هایی را که توسط آموزگار گرفته شده، مشاهده خواهید کرد. وقتی یک دوره را انتخاب می کنید، لیستی از دانشجویهایی که در آن دوره ثبت نام کرده اند را خواهید دید. شما می توانید نام آموزگار، تاریخ استخدام و دفتر کار او را ویرایش کنید. دفتر کار مجموعه موجودیت جداگانه ای است که از طریق خصوصیت **Navigation** می توانید به آن دسترسی داشته باشید.

شما می توانید در قسمت **code** (کد برنامه نویسی مانند C#) یا در قسمت **markup** (کد ساخت یافته asp.net) داده های **master** را به داده های **detail** پیوند دهید. در این بخش هر دو روش را مشاهده خواهید کرد.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 18
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSE DETAILS

ID	2030
Title	Poetry
Credits	2
Department	English
Location	
URL	http://www.fineartschool.n

STUDENT GRADES

Name	Grade
Barzdukas, Gytis	3.50
Justice, Peggy	4.00

COURSES TAUGHT

	ID	Title	Department
Select	2030	Poetry	English

¹ Related Data

نمایش و بروزرسانی موجودیت های وابسته در کنترل GridView

یک صفحه وب به نام *Instructors.aspx* که از *Site.Master* به عنوان *Master page* استفاده می کند، ایجاد کنید. و کد ساختیافته زیر را به کنترل **Content** با نام **Content2** اضافه کنید.

```
<h2>Instructors</h2>
<div>
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Person"
Where="it.HireDate is not null" Include="OfficeAssignment" EnableUpdate="True">
</asp:EntityDataSource>
</div>
```

این کد ساختیافته یک کنترل **EntityDataSource** که اساتید را انتخاب می کند و قابلیت بروزرسانی دارد، ایجاد می کند. عنصر **Div** باعث قرارگیری اجزای درون آن در سمت چپ خواهد شد، بنابراین شما می توانید بعداً یک ستون در سمت راست قرار دهید. کد زیر را به منظور ایجاد کنترل **GridView** و کنترل **Lable** که از آن برای نمایش پیغام های خطا استفاده خواهید کرد، بین **</asp:EntityDataSource>** و **</div>** قرار دهید.

```
<asp:GridView ID="InstructorsGridView" runat="server" AllowPaging="True"
AllowSorting="True"
AutoGenerateColumns="False" DataKeyNames="PersonID"
DataSourceID="InstructorsEntityDataSource"
OnSelectedIndexChanged="InstructorsGridView_SelectedIndexChanged"
SelectedRowStyle-BackColor="LightGray"
onrowupdating="InstructorsGridView_RowUpdating">
<Columns>
<asp:CommandField ShowSelectButton="True" ShowEditButton="True"/>
<asp:TemplateField HeaderText="Name" SortExpression="LastName">
<ItemTemplate>
<asp:Label ID="InstructorLastNameLabel" runat="server" Text='<## Eval("LastName")
%>'></asp:Label>,
<asp:Label ID="InstructorFirstNameLabel" runat="server" Text='<##
Eval("FirstMidName") %>'></asp:Label>
</ItemTemplate>
<EditItemTemplate>
<asp:TextBox ID="InstructorLastNameTextBox" runat="server" Text='<##
Bind("FirstMidName") %>' Width="7em"></asp:TextBox>
<asp:TextBox ID="InstructorFirstNameTextBox" runat="server" Text='<##
Bind("LastName") %>' Width="7em"></asp:TextBox>
```

```

</EditItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Hire Date" SortExpression="HireDate">
<ItemTemplate>
<asp:Label ID="InstructorHireDateLabel" runat="server" Text='<%=# Eval("HireDate",
"{0:d}") %>'></asp:Label>
</ItemTemplate>
<EditItemTemplate>
<asp:TextBox ID="InstructorHireDateTextBox" runat="server" Text='<%=# Bind("HireDate",
"{0:d}") %>' Width="7em"></asp:TextBox>
</EditItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Office Assignment"
SortExpression="OfficeAssignment.Location">
<ItemTemplate>
<asp:Label ID="InstructorOfficeLabel" runat="server" Text='<%=#
Eval("OfficeAssignment.Location") %>'></asp:Label>
</ItemTemplate>
<EditItemTemplate>
<asp:TextBox ID="InstructorOfficeTextBox" runat="server"
Text='<%=# Eval("OfficeAssignment.Location") %>' Width="7em"
oninit="InstructorOfficeTextBox_Init"></asp:TextBox>
</EditItemTemplate>
</asp:TemplateField>
</Columns>
<SelectedRowStyle BackColor="LightGray"></SelectedRowStyle>
</asp:GridView>
<asp:Label ID="ErrorMessageLabel" runat="server" Text="" Visible="false"
ViewStateMode="Disabled">
</asp:Label>

```

کنترل **GridView** امکان انتخاب سطرها را فراهم می کند، سطر انتخابی را با رنگ پس زمینه خاکستری روشن برجسته می کند، و اداره کننده های وقفه را برای رخدادهای (که بعدا ایجاد خواهید کرد) **SelectedIndexChanged** و **Updating** مشخص می کند. همچنین برای خصوصیت **DataKeyNames** مقدار **PersonID** را تعیین می کند. بنابراین مقدار کلید سطر انتخابی می تواند به کنترل دیگری که بعدا ایجاد خواهید کرد فرستاده شود.

ستون آخر حاوی دفتر کار استاد است، که در خصوصیت **Navigation** موجودیت **Person** ذخیره شده است، چون از یک موجودیت وابسته به دست می آید. توجه کنید عنصر **EditItemTemplate** از **Eval** به جای **Bind** استفاده کرده است، چون کنترل **GridView** نمی تواند مستقیما به خصوصیت **Navigation** به منظور بروزرسانی آن مقید شود. شما دفتر کار را در قسمت کد(برنامه) بروزرسانی خواهید کرد. به این منظور شما نیاز به یک ارجاع به کنترل **TextBox** خواهید داشت، که آن را در رخداد **Init** کنترل **TextBox** دریافت و ذخیره خواهید کرد.

بعد از کنترل **GridView** یک کنترل **Lable** به منظور نمایش پیغام های خطا وجود دارد. خاصیت **Visible** این کنترل برابر **false** و وضعیت نمایش آن خاموش است، بنابراین تنها به وسیله کد برنامه و در زمان واکنش به یک خطا، قابل مشاهده می گردد.

فایل *Instructors.aspx.cs* را باز، و دستور زیر را اضافه کنید:

```
using ContosoUniversity.DAL;
```

یک کلاس خصوصی¹ دقیقاً بعد از تعریف نام **partial-class** برای نگه داری یک ارجاع به جعبه متنی دفتر کار اضافه کنید.

```
private TextBox instructorOfficeTextBox;
```

یک **Stub** برای گرداننده رخداد **SelectedIndexChanged** که بعداً آن را اضافه خواهید کرد، ایجاد کنید. همچنین یک گرداننده² برای رخداد **Init** کنترل **TextBox** مربوط به دفتر کار اضافه کنید، بنابراین شما می توانید یک ارجاع به کنترل **TextBox** را ذخیره کنید. شما می توانید این مرجع را برای گرفتن مقادیری که کاربر به منظور بروزرسانی موجودیتی که با خاصیت **Navigation** وابستگی دارد، استفاده کنید.

```
protected void InstructorsGridView_SelectedIndexChanged(object sender,EventArgs e)
{
}

protected void InstructorOfficeTextBox_Init(object sender,EventArgs e)
{
    instructorOfficeTextBox = sender as TextBox;
}
```

شما از رخداد **Updating** کنترل **GridView** برای بروزرسانی خاصیت **Location** موجودیت **OfficeAssignment** استفاده خواهید کرد. کد زیر را برای رخداد **Updating** اضافه کنید:

¹ Private

² Handler

```

protected void InstructorsGridView_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    using(var context =new SchoolEntities())
    {
        var instructorBeingUpdated =Convert.ToInt32(e.Keys[0]);
        var officeAssignment =(from o in context.OfficeAssignment
        where o.InstructorID== instructorBeingUpdated
        select o).FirstOrDefault();
        try
        {
            if(String.IsNullOrEmpty(instructorOfficeTextBox.Text)==false)
            {
                if(officeAssignment ==null)
                {
                    context.OfficeAssignment.AddObject(OfficeAssignment.CreateOfficeAssignment(
                    instructorBeingUpdated, instructorOfficeTextBox.Text,null));
                }
                else
                {
                    officeAssignment.Location= instructorOfficeTextBox.Text;
                }
            }
            else
            {
                if(officeAssignment !=null)
                {
                    context.DeleteObject(officeAssignment);
                }
            }
            context.SaveChanges();
        }
        catch(Exception)
        {
            e.Cancel=true;
            ErrorMessageLabel.Visible=true;
            ErrorMessageLabel.Text="Update failed.";
            //Add code to log the error.
        }
    }
}

```

زمانی که کاربر روی **Update** کنترل **GridView** کلیک کند، این کد اجرا خواهد شد. این کد، برای برگرداندن موجودیت **OfficeAssignment** که با موجودیت **Person** جاری، وابستگی دارد، از **LINQ to Entities** استفاده می کند. این کار را با استفاده از **PersonID** سطر انتخاب شده و به دست آمده از آرگومان رخداد، انجام خواهد داد.

- سپس کد برنامه، بر اساس کنترل **InstructorOfficeTextBox**، یکی از اعمال زیر را انجام خواهد داد:

- اگر **TextBox** دارای مقدار است و هیچ موجودیت **OfficeAssignment** ای برای **Update** وجود ندارد، یک موجودیت ایجاد می کند.
- اگر **TextBox** دارای مقدار است و یک موجودیت **OfficeAssignment** وجود دارد، مقدار خصوصیت **Location** را بروزرسانی می کند.
- اگر **TextBox** خالی است و یک موجودیت **OfficeAssignment** وجود دارد، این موجودیت را حذف می کند.

بعد از این کار، تغییرات را در دیتابیس ذخیره می کند. اگر استثنایی رخ دهد، یک پیغام خطا را نشان خواهد داد.

صفحه را اجرا کنید.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

روی **Edit** کلیک کنید، تمام فیلدها به صورت **TextBox** تغییر خواهند کرد.

INSTRUCTORS

	Name		Hire Date	Office Assignment
Update Cancel	Kim	Abercrombie	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi		8/6/2002	29 Adams
Edit Select	Harui, Roger		7/1/1998	37 Williams
Edit Select	Zheng, Roger		2/12/2004	143 Smith
Edit Select	Kapoor, Candace		1/15/2001	57 Adams
Edit Select	Serrano, Stacy		6/1/1999	271 Williams
Edit Select	Stewart, Jasmine		10/12/1997	131 Smith
Edit Select	Xu, Kristen		7/23/2001	203 Williams
Edit Select	Van Houten, Roger		12/7/2000	213 Smith

هر کدام از این مقادیر را که شامل **Office Assignment** است، تغییر دهید. روی **Update** کلیک کنید، مشاهده خواهید کرد که تغییرات در لیست اعمال شده.

نمایش موجودیت های وابسته در کنترل **Separate**

هر آموزگار می تواند یک یا چند دوره را تدریس کند، بنابراین شما یک کنترل **EntityDataSource** و یک کنترل **GridView** برای لیست کردن دوره هایی که با آموزگار انتخاب شده در کنترل **GridView** دارای ارتباط هستند، اضافه خواهید کرد. برای ایجاد عنوان و همچنین کنترل **EntityDataSource** برای موجودیت های **Course**، کد ساخت یافته زیر را بین کنترل **Label** نمایش دهنده پیغام خطا و تگ **</div>** قرار دهید:

```
<h3>Courses Taught</h3>
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Course"
Where="@PersonID IN (SELECT VALUE instructor.PersonID FROM it.People AS instructor)">
<WhereParameters>
<asp:ControlParameter ControlID="InstructorsGridView" Type="Int32" Name="PersonID"
PropertyName="SelectedValue"/>
</WhereParameters>
</asp:EntityDataSource>
```


پارامتر `Where` دربرگیرنده مقدار `PersonID` آموزگاری است که سطر آن در کنترل `InstructorsGridView` انتخاب شده است. پارامتر `Where` دارای یک دستور `Select` درونی است که تمام موجودیت های `Person` وابسته، که از خاصیت `People Navigation` موجودیت `Course` به دست می آیند را برمی گرداند و موجودیت `Course` را در صورتی که تنها یکی از موجودیت های وابسته `Person` حاوی مقدار `PersonID` انتخابی باشد، انتخاب می کند.

برای ایجاد کنترل `GridView`، کد ساختیافته زیر را بلافاصله بعد از کنترل `CoursesEntityDataSource` اضافه کنید (قبل از تگ `</div>`):

```
<asp:GridView ID="CoursesGridView" runat="server"
DataSourceID="CoursesEntityDataSource"
AllowSorting="True" AutoGenerateColumns="False"
SelectedRowStyle-BackColor="LightGray"
DataKeyNames="CourseID">
<EmptyDataTemplate>
<p>No courses found.</p>
</EmptyDataTemplate>
<Columns>
<asp:CommandField ShowSelectButton="True"/>
<asp:BoundField DataField="CourseID" HeaderText="ID" ReadOnly="True"
SortExpression="CourseID"/>
<asp:BoundField DataField="Title" HeaderText="Title" SortExpression="Title"/>
<asp:TemplateField HeaderText="Department" SortExpression="DepartmentID">
<ItemTemplate>
<asp:Label ID="GridViewDepartmentLabel" runat="server" Text='<%=
Eval("Department.Name") %>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>
```

اگر هیچ استادی انتخاب نشده باشد، هیچ دوره ای نمایش داده نمی شود، به همین خاطر یک عنصر `EmptyDataTemplate` گنجانده شده است.

صفحه را اجرا کنید.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSES TAUGHT

No courses found.

یک استاد که یک یا چند دوره به او اختصاص داده شده را انتخاب کنید، سپس دوره یا دوره ها در یک لیست نمایش داده می شوند. (توجه فرمائید: گرچه شمای دیتابیس اجازه وجود دوره های چندگانه را می دهد، اما در داده هایی که به صورت تست در دیتابیس قرار داده شده اند، هیچ آموزگاری بیش از یک دوره ندارد. شما می توانید دوره ها را خودتان با استفاده از پنجره **Server Explorer** به دیتابیس اضافه کنید یا از طریق صفحه *CoursesAdd.aspx*، که در بخش های بعدی اضافه خواهید کرد، این کار را انجام دهید.)

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSES TAUGHT

	ID	Title	Department
Select	2030	Poetry	English

کنترل `CoursesGridView` تنها تعدادی از فیلدهای `Course` را نشان می دهد. برای نمایش همه جزئیات برای یک موجودیت `Course`، شما از یک کنترل `DetailsView` برای `Course` ای که توسط کاربر انتخاب شده استفاده خواهید کرد. در `Instructors.aspx` کد ساختیافته زیر را بعد از تگ `</div>` اضافه کنید (مطمئن شوید این کد را بعد از تگ بسته `</div>` اضافه کنید، نه قبل از آن):

```
<div>
<h3>Course Details</h3>
<asp:EntityDataSource ID="CourseDetailsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Course"
AutoGenerateWhereClause="False" Where="it.CourseID =@CourseID"
Include="Department,OnlineCourse,OnsiteCourse,StudentGrade.Person"
OnSelected="CourseDetailsEntityDataSource_Selected">
<WhereParameters>
<asp:ControlParameter ControlID="CoursesGridView" Type="Int32" Name="CourseID"
PropertyName="SelectedValue"/>
</WhereParameters>
</asp:EntityDataSource>
<asp:DetailsView ID="CourseDetailsView" runat="server" AutoGenerateRows="False"
DataSourceID="CourseDetailsEntityDataSource">
<EmptyDataTemplate>
<p>
No course selected.</p>
</EmptyDataTemplate>
<Fields>
<asp:BoundField DataField="CourseID" HeaderText="ID" ReadOnly="True"
SortExpression="CourseID"/>
<asp:BoundField DataField="Title" HeaderText="Title" SortExpression="Title"/>
<asp:BoundField DataField="Credits" HeaderText="Credits" SortExpression="Credits"/>
<asp:TemplateField HeaderText="Department">
<ItemTemplate>
<asp:Label ID="DetailsViewDepartmentLabel" runat="server" Text='<%#
Eval("Department.Name") %>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Location">
<ItemTemplate>
<asp:Label ID="LocationLabel" runat="server" Text='<%# Eval("OnsiteCourse.Location")
%>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="URL">
<ItemTemplate>
<asp:Label ID="URLLabel" runat="server" Text='<%# Eval("OnlineCourse.URL")
%>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
</Fields>
</asp:DetailsView>
</div>
```

این کد یک کنترل EntityDataSource که به مجموعه موجودیت Course مقید شده ایجاد می کند. خصوصیت Where دوره ای را با استفاده از CourseID سطر انتخاب شده در کنترل GridView انتخاب می کند. این کد یک اداره کننده برای رخداد Selected تعیین می کند، که بعدا شما برای نمایش نمرات دانشجوها از آن استفاده خواهید کرد.

در `Instructors.aspx.cs` stub زیر را برای متد `CourseDetailsEntityDataSource_Selected` ایجاد کنید.

(بعدا این Stub را تکمیل خواهید کرد، اما در حال حاضر شما برای کامپایل و اجرای صفحه به آن نیاز دارید.)

```
protected void CourseDetailsEntityDataSource_Selected(object sender,
EntityDataSourceSelectedEventArgs e)
{
}
```

صفحه را اجرا کنید.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSE DETAILS

No course selected.

COURSES TAUGHT

No courses found.

در ابتدا جزئیات دوره نمایش داده نمی شود، چون هیچ دوره ای انتخاب نشده است. یک استاد که یک دوره به او اختصاص داده شده و سپس یک دوره را برای دیدن جزئیات آن انتخاب کنید.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	17 Smith
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSE DETAILS

ID	2030
Title	Poetry
Credits	2
Department	English
Location	
URL	http://www.finearts

COURSES TAUGHT

	ID	Title	Department
Select	2030	Poetry	English

استفاده از رخداد "Selected" کنترل **EntityDataSource** برای نمایش داده های وابسته

در پایان شما نیاز دارید همه دانشجویها و نمرات آنها را برای دوره انتخابی نمایش دهید. برای انجام این کار، از رخداد "Selected" کنترل **EntityDataSource** که به **DetailsView** مربوط به **Course** مقید شده استفاده خواهید کرد.

در **Instructors.aspx** کد ساختیافته زیر را بعد از کنترل **DetailsView** اضافه کنید:

```

<h3>Student Grades</h3>
<asp:ListView ID="GradesListView" runat="server">
<EmptyDataTemplate>
<p>No student grades found.</p>
</EmptyDataTemplate>
<LayoutTemplate>
<table border="1" runat="server" id="itemPlaceholderContainer">
<tr runat="server">
<th runat="server">
Name
</th>
<th runat="server">
Grade
</th>
</tr>
<tr id="itemPlaceholder" runat="server">
<tr>
<td>
<asp:Label ID="StudentLastNameLabel" runat="server" Text='<%= Eval("Person.LastName")
%>' />,
<asp:Label ID="StudentFirstNameLabel" runat="server" Text='<%=
Eval("Person.FirstMidName") %>' />
</td>
<td>
<asp:Label ID="StudentGradeLabel" runat="server" Text='<%= Eval("Grade") %>' />
</td>
</tr>
</ItemTemplate>
</asp:ListView>

```

این کد، یک کنترل **ListView** برای نمایش دانشجویها و نمرات آنها براساس دوره انتخاب شده ایجاد می کند. هیچ **Data source** ای تعیین نشده، چون در قسمت کد برنامه نویسی داده ها را به این کنترل مقید خواهید کرد. عنصر **EmptyDataTemplate** زمانی که **Course** ای انتخاب نشده است پیغامی را ارائه می دهد، در این صورت هیچ دانشجویی نمایش داده نمی شود. عنصر **LayoutTemplate** یک **Html table** برای نمایش لیست ایجاد می کند، و **ItemTemplate** ستون هایی که باید نمایش داده شوند را تعیین می کند. **student ID** و **student grade** مربوط به موجودیت **StudentGrade** هستند و **Student name** مربوط به موجودیت **Person** است که **Entity Framework** در خصوصیت **Person Navigation** مربوط به موجودیت **StudentGrade** آن را قابل دسترس می کند.

در `Instructors.aspx.cs` متد `CourseDetailsEntityDataSource_Selected` را با کد زیر جایگزین کنید:

```

protected void CourseDetailsEntityDataSource_Selected(object sender,
    EntityDataSourceSelectedEventArgs e)
{
    var course = e.Results.Cast<Course>().FirstOrDefault();
    if(course !=null)
    {
        var studentGrades = course.StudentGrade.ToList();
        GradesListView.DataSource= studentGrades;
        GradesListView.DataBind();
    }
}

```

آرگومان رخداد¹ این رویداد، داده های انتخابی را به صورت مجموعه ای ارائه می دهد. در صورتی که چیزی انتخاب نشده باشد هیچ عنصری نخواهد داشت و اگر یک موجودیت **Course** انتخاب شده باشد دارای یک عنصر خواهد بود. اگر یک موجودیت **Course** انتخاب شده باشد، از متد **First** برای تبدیل مجموعه² به یک شی استفاده می کند. سپس موجودیت های **StudentGrade** را از خصوصیت **Navigation** می گیرد، و آنها را به یک مجموعه تبدیل می کند، و کنترل **GradesListView** را به این مجموعه مقید می کند.

این روش برای نمایش نمرات، کارآمد است، اما شما می خواهید مطمئن شوید در اولین بار که صفحه نمایش داده می شود و هر وقت که **Course** انتخاب نشده است، پیغام مربوطه در **empty data template** نمایش داده می شود. به این منظور، کد زیر را که در دو جای مختلف فراخوانی خواهید کرد ایجاد کنید:

```

private void ClearStudentGradesDataSource()
{
    var emptyStudentGradesList =new List<StudentGrade>();
    GradesListView.DataSource= emptyStudentGradesList;
    GradesListView.DataBind();
}

```

برای نمایش **empty data template**، متد بالا را در اولین باری که صفحه نمایش داده می شود از طریق متد **Page_Load** و بار دیگر آن را از طریق متد **InstructorsGridView_SelectedIndexChanged** فراخوانی کنید، چون هنگام انتخاب یک استاد (**instructor**)، رخداد مربوط به آن، بالا می آید، در این هنگام **Course** های جدید در کنترل **GridView** نمایش دوره ها، بارگذاری می شوند، و هیچ دوره ای هنوز توسط کاربر انتخاب نشده است.

¹ event argument

² Collection

اینجا هر دو فراخوانی آورده شده است.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        ClearStudentGradesDataSource();
    }
}
protected void InstructorsGridView_SelectedIndexChanged(object sender,
EventArgs e)
{
    ClearStudentGradesDataSource();
}
```

صفحه را اجرا کنید.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSES TAUGHT

No courses found.

COURSE DETAILS

No course selected.

STUDENT GRADES

No student grades found.

یک آموزگار که یک دوره را برداشته، انتخاب کرده، و سپس دوره را انتخاب کنید.

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	17 Smith
Edit Select	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	Harui, Roger	7/1/1998	37 Williams
Edit Select	Zheng, Roger	2/12/2004	143 Smith
Edit Select	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	Xu, Kristen	7/23/2001	203 Williams
Edit Select	Van Houten, Roger	12/7/2000	213 Smith

COURSES TAUGHT

	ID	Title	Department
Select	2030	Poetry	English

COURSE DETAILS

ID	2030
Title	Poetry
Credits	2
Department	English
Location	
URL	http://www.finearts

STUDENT GRADES

Name	Grade
Barzdukas, Gytis	3.50
Justice, Peggy	4.00

تا الان شما روش هایی را برای کار با داده های وابسته (مرتبط) مشاهده کردید. در بخش بعد چگونگی افزودن ارتباط بین موجودیت های موجود، حذف ارتباط ها، و افزودن یک موجودیت جدید که با یک موجودیت موجود رابطه دارد را خواهید آموخت.

ادامه کار با داده های وابسته

در بخش قبل شما استفاده از کنترل `EntityDataSource` را به منظور کار با داده های مرتبط آغاز کردید. همچنین سطوح چندگانه ای از داده های سلسله مراتبی و ویرایش شده در خصوصیت `Navigation` را نمایش دادید. در این بخش شما کار با داده های مرتبط را با افزودن و حذف رابطه ها و افزودن موجودیت جدیدی که با موجودیت موجود دارای رابطه است، ادامه خواهید داد.

در آینده صفحه ای ایجاد خواهید کرد که دوره های اختصاص داده شده به دانشکده¹، را اضافه می کند. دانشکده ها در حال حاضر موجود اند، زمانی که `course` جدیدی ایجاد می کنید، همزمان رابطه جدیدی بین آن و دانشکده موجود برقرار خواهید کرد.

¹ department

ADD COURSES

ID	<input type="text"/>
Title	<input type="text"/>
Credits	<input type="text"/>
Department	Engineering <input type="button" value="v"/>
<input type="button" value="Insert"/> <input type="button" value="Cancel"/>	

همچنین شما یک صفحه ایجاد خواهید کرد که با ارتباط چند به چند کار می کند. با اختصاص دادن یک instructor به Course (اضافه کردن رابطه، بین دو موجودیتی که شما انتخاب می کنید) یا حذف یک Instructor از یک Course (حذف یک رابطه، بین دو موجودیتی که شما انتخاب می کنید). در دیتابیس اضافه کردن رابطه بین instructor و Course باعث افزودن سطر جدید به جدول وابستگی CourseInstructor می شود. حذف یک رابطه شامل حذف یک سطر از جدول وابستگی CourseInstructor می شود. هر چند، در Entity Framework شما این کار را با تنظیم خصوصیات Navigation و بدون رجوع صریح به جدول CourseInstructor، انجام می دهید.

ASSIGN INSTRUCTORS TO COURSES OR REMOVE FROM COURSES

Select an Instructor:

ASSIGN A COURSE

Select a Course:

REMOVE A COURSE

Select a Course:

اضافه کردن یک موجودیت همراه با رابطه ای به یک موجودیت موجود

صفحه وبی با نام *CoursesAdd.aspx* که از *Site.Master* استفاده می کند ایجاد کنید. کد ساختیافته زیر را به کنترل

Content با نام *Content2* اضافه کنید:

```
<h2>Add Courses</h2>
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Course"
EnableInsert="True" EnableDelete="True">
</asp:EntityDataSource>
<asp:DetailsView ID="CoursesDetailsView" runat="server" AutoGenerateRows="False"
DataSourceID="CoursesEntityDataSource" DataKeyNames="CourseID"
DefaultMode="Insert" oniteminserting="CoursesDetailsView_ItemInserting">
<Fields>
<asp:BoundField DataField="CourseID" HeaderText="ID"/>
<asp:BoundField DataField="Title" HeaderText="Title"/>
<asp:BoundField DataField="Credits" HeaderText="Credits"/>
<asp:TemplateField HeaderText="Department">
<InsertItemTemplate>
<asp:EntityDataSource ID="DepartmentsEntityDataSource" runat="server"
ConnectionString="
name=SchoolEntities"
DefaultContainerName="SchoolEntities" EnableDelete="True" EnableFlattening="False"
EntitySetName="Departments" EntityTypeFilter="Department">
</asp:EntityDataSource>
<asp:DropDownList ID="DepartmentsDropDownList" runat="server"
DataSourceID="DepartmentsEntityDataSource"
DataTextField="Name" DataValueField="DepartmentID"
oninit="DepartmentsDropDownList_Init">
</asp:DropDownList>
</InsertItemTemplate>
</asp:TemplateField>
<asp:CommandField ShowInsertButton="True"/>
</Fields>
</asp:DetailsView>
```

این کد ساختیافته یک کنترل *EntityDataSource* برای انتخاب *Course* ایجاد می کند، که قابلیت انجام عمل *Insert* را داشته و یک اداره کننده، برای رخداد *Inserting* تعیین می کند. شما از این اداره کننده رخداد، برای بروزرسانی خاصیت *Department navigation*، هنگام ایجاد یک موجودیت *Course* جدید، استفاده خواهید کرد.

همچنین این کد ساختیافته یک کنترل *DetailsView* برای اضافه کردن موجودیت های جدید *Course* ایجاد می کند. این کد از *Bound field* ها برای خصوصیت های موجودیت *Course* استفاده می کند. شما باید مقدار *CourseID* را وارد کنید

چون این، یک فیلد ID تولید شده توسط سیستم نیست. بلکه، شماره course ای است که باید به صورت دستی هنگام ایجاد یک Course تعیین شود.

شما از یک template field برای خاصیت Department Navigation استفاده می کنید، چون خاصیت های Navigation نمی توانند توسط کنترل های BoundField استفاده شوند. template field، یک لیست کشویی (drop-down list) برای انتخاب Department ارائه می دهد. لیست کشویی با استفاده از Eval به جای Bind به موجودیت Department مقید شده، باز به خاطر اینکه نمی توانید مستقیماً خصوصیت های Navigation را به منظور بروزرسانی، مقید کنید. شما یک اداره کننده برای رخدادهای Init کنترل DropDownList تعیین می کنید، تا بتوانید یک ارجاع به این کنترل را برای استفاده در کد برنامه، که بروزرسانی کلید خارجی DepartmentID را انجام می دهد، ذخیره کنید.

در *CoursesAdd.aspx.cs* درست بعد از تعریف partial-class یک فیلد کلاس برای نگه داری یک ارجاع به کنترل *DepartmentsDropDownList* اضافه کنید:

```
private DropDownList departmentDropDownList;
```

یک اداره کننده برای رخداد Init کنترل *DepartmentsDropDownList* اضافه کنید تا بتوانید یک ارجاع به این کنترل را ذخیره کنید. این کار به شما اجازه استفاده از مقداری که کاربر وارد کرده و همچنین استفاده از آن برای بروزرسانی مقدار *DepartmentID* موجودیت Course را می دهد.

```
protected void DepartmentsDropDownList_Init(object sender, EventArgs e)
{
    departmentDropDownList = sender as DropDownList;
}
```

یک اداره کننده، برای رخداد Inserting کنترل *DetailsView* اضافه کنید:

```
protected void CoursesDetailsView_ItemInserting(object sender,
    DetailsViewInsertEventArgs e)
{
    var departmentID = Convert.ToInt32(departmentDropDownList.SelectedValue);
    e.Values["DepartmentID"] = departmentID;
}
```

هنگامی که کاربر روی Insert کلیک می کند، رخداد Inserting قبل از درج رکورد جدید، بالا می آید. در قسمت کد، گرانداده رخداد، مقدار DepartmentID را از کنترل DropDownList گرفته و از آن برای مقداردهی خصوصیت DepartmentID موجودیت Course استفاده می کند.

Entity Framework مواظب اضافه کردن این Course به خاصیت Course Navigation موجودیت Department خواهد بود. همچنین Department را به خصوصیت Department Navigation موجودیت Course اضافه می کند.

صفحه را اجرا کنید.

ADD COURSES

ID	<input type="text"/>
Title	<input type="text"/>
Credits	<input type="text"/>
Department	Engineering ▼
Insert Cancel	

یک ID، Title، و شماره Credits وارد کرده و Department را انتخاب کنید، سپس روی Insert کلیک کنید.

صفحه *Courses.aspx* را اجرا کنید، و همان Department را انتخاب کنید تا Course جدید را ببینید.

COURSES BY DEPARTMENT

Select a Department: Engineering ▼

ID	Title	Credits
1050	Chemistry	4
1061	Physics	4
4062	New engineering course	5

کار با ارتباط های چند به چند

ارتباط بین مجموعه موجودیت های Course و Person از نوع چند به چند تنظیم شده است. یک موجودیت Course دارای یک خاصیت Navigation با نام Person است که می تواند شامل صفر، یک یا چند موجودیت وابسته Person باشد (اساتیدی که برای تدریس آن دوره اختصاص داده شده اند را ارائه می دهد). و یک موجودیت Person یک خاصیت Navigation با نام Course دارد که می تواند شامل صفر، یک یا چند موجودیت وابسته Course باشد (Course هایی را ارائه می دهد که برای تدریس به استاد مورد نظر اختصاص داده شده). یک استاد ممکن است چندین دوره را تدریس کند، و یک دوره ممکن است توسط چندین استاد تدریس شود. در این بخش، توسط بروزرسانی خصوصیت های navigation موجودیت های وابسته، روابط بین Person و Course را اضافه و حذف خواهید کرد.

یک صفحه وب با نام *InstructorsCourses.aspx* که از *Site.Master* استفاده می کند، ایجاد کنید و کد ساختیافته زیر را به کنترل **Content** با نام **Content2** اضافه کنید:

```
<h2>Assign Instructors to Courses or Remove from Courses</h2>
<br/>
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Person"
Where="it.HireDate is not null" Select="it.LastName + ', ' + it.FirstMidName AS Name,
it.PersonID">
</asp:EntityDataSource>
Select an Instructor:
<asp:DropDownList ID="InstructorsDropDownList" runat="server"
DataSourceID="InstructorsEntityDataSource"
AutoPostBack="true" DataTextField="Name" DataValueField="PersonID"
OnSelectedIndexChanged="InstructorsDropDownList_SelectedIndexChanged"
OnDataBound="InstructorsDropDownList_DataBound">
</asp:DropDownList>
<h3>
Assign a Course</h3>
<br/>
Select a Course:
<asp:DropDownList ID="UnassignedCoursesDropDownList" runat="server"
DataTextField="Title" DataValueField="CourseID">
</asp:DropDownList>
<br/>
<asp:Button ID="AssignCourseButton" runat="server" Text="Assign"
OnClick="AssignCourseButton_Click"/>
<br/>
<asp:Label ID="CourseAssignedLabel" runat="server" Visible="false"
Text="Assignments successful"></asp:Label>
<br/>
<h3>
```

```

Remove a Course</h3>
<br/>
Select a Course:
<asp:DropDownList ID="AssignedCoursesDropDownList" runat="server"
DataTextField="title" DataValueField="courseid">
</asp:DropDownList>
<br/>
<asp:Button ID="RemoveCourseButton" runat="server" Text="Remove"
OnClick="RemoveCourseButton_Click"/>
<br/>
<asp:Label ID="CourseRemovedLabel" runat="server" Visible="false"
Text="Removalsuccessful"></asp:Label>

```

این کد، یک کنترل `EntityDataSource` ایجاد می کند که `Name` و `PersonID` موجودیت های `Person` را برای اساتید برمی گرداند. یک کنترل `DropDownList` به یک کنترل `EntityDataSource` مقید شده است. کنترل `EntityDataSource` یک گراندده برای رخداد `DataBound` مشخص می کند. شما از این گراندده وقفه برای مقید کردن داده¹، بین دو لیست کشویی که `Course` ها را نمایش می دهد استفاده خواهید کرد.

همچنین این کد ساختیافته مجموعه کنترل های زیر را به منظور اختصاص دادن یک دوره به استاد انتخابی، ایجاد می کند:

- یک کنترل `DropDownList` برای انتخاب یک دوره به منظور اختصاص آن به استاد. این کنترل توسط `Course` هایی که به استاد انتخابی اختصاص داده نشده اند، پر می شود.
- یک کنترل `Button` برای شروع عملیات اختصاص دادن دوره به استاد.
- یکی کنترل `Label` برای نمایش پیغام خطا در صورت به مشکل خوردن عملیات تخصیص.

در پایان این کد، گروهی از کنترل ها را به منظور حذف یک `Course` از استاد انتخابی ایجاد می کند.

در `InstructorsCourses.aspx.cs` یک دستور `using` اضافه کنید:

```
using ContosoUniversity.DAL;
```

¹ Databound

یک متد برای دستکاری دو drop-down list که Course ها را نشان می دهند، اضافه کنید:

```
private void PopulateDropDownLists()
{
    using(var context =new SchoolEntities())
    {
        var allCourses =(from c in context.Course select c).ToList();
        var instructorID
=Convert.ToInt32(InstructorsDropDownList.SelectedValue);
        var instructor =(from p in context.Person.Include("Courses")
        where p.PersonID== instructorID select p).First();
        var assignedCourses = instructor.Course.ToList();
        var unassignedCourses =
allCourses.Except(assignedCourses.AsEnumerable()).ToList();
        UnassignedCoursesDropDownList.DataSource= unassignedCourses;
        UnassignedCoursesDropDownList.DataBind();
        UnassignedCoursesDropDownList.Visible=true;
        AssignedCoursesDropDownList.DataSource= assignedCourses;
        AssignedCoursesDropDownList.DataBind();
        AssignedCoursesDropDownList.Visible=true;
    }
}
```

این کد همه Course ها را از مجموعه موجودیت Course می گیرد و همچنین با توجه به استاد انتخاب شده، دوره هایی را از خصوصیت Course navigation موجودیت Person می گیرد. سپس دوره هایی را که به آن استاد اختصاص داده شده را معین می کند و بر طبق آن Dropdown list را پر می کند.

گرداننده ای برای رخداد click دکمه Assign اضافه کنید:

```
protected void AssignCourseButton_Click(object sender,EventArgs e)
{
    using(var context =new SchoolEntities())
    {
        var instructorID
=Convert.ToInt32(InstructorsDropDownList.SelectedValue);
        var instructor =(from p in context.Person
        where p.PersonID== instructorID
        select p).First();
        var courseID
=Convert.ToInt32(UnassignedCoursesDropDownList.SelectedValue);
        var course =(from c in context.Course
        where c.CourseID== courseID
        select c).First();
    }
}
```



```

        instructor.Course.Add(course);
    try
    {
        context.SaveChanges();
        PopulateDropDownLists();
        CourseAssignedLabel.Text="Assignment successful.";
    }
    catch(Exception)
    {
        CourseAssignedLabel.Text="Assignment unsuccessful.";
        //Add code to log the error.
    }
    CourseAssignedLabel.Visible=true;
}
}

```

این کد موجودیت Person را بر اساس استاد انتخاب شده، و موجودیت Course را بر اساس دوره انتخاب شده برمی گرداند، و دوره انتخاب شده را به خصوصیت Course navigation موجودیت Person استاد اضافه می کند. سپس تغییرات را در دیتابیس ذخیره می کند و dropdown list را دوباره پر می کند، بنابراین نتیجه فوراً قابل مشاهده خواهد بود.

یک گراننده، برای رخداد Click دکمه Remove اضافه کنید:

```

protected void RemoveCourseButton_Click(object sender,EventArgs e)
{
    using(var context =new SchoolEntities())
    {
        var instructorID =Convert.ToInt32(InstructorsDropDownList.SelectedValue);
        var instructor =(from p in context.Person
        where p.PersonID== instructorID
        select p).First();
        var courseID =Convert.ToInt32(AssignedCoursesDropDownList.SelectedValue);
        var courses = instructor.Course;
        var courseToRemove =new Course();
        foreach(Course c in courses)
        {
            if(c.CourseID== courseID)
            {
                courseToRemove = c;
                break;
            }
        }
    }
}

```

```

try
{
    courses.Remove(courseToRemove);
    context.SaveChanges();
    PopulateDropDownLists();
    CourseRemovedLabel.Text="Removal successful.";
}
catch(Exception)
{
    CourseRemovedLabel.Text="Removal unsuccessful.";
    //Add code to log the error.
}
CourseRemovedLabel.Visible=true;
}
}

```

این کد موجودیت Person را براساس استاد انتخاب شده، و موجودیت Course را بر اساس Course انتخاب شده بر می گرداند، و دوره انتخاب شده را از خاصیت Course navigation موجودیت Person استاد حذف می کند. سپس تغییرات را در دیتابیس ذخیره می کند و Dropdown list ها را دوباره پر می کند، بنابراین نتایج فوراً قابل مشاهده خواهد بود.

کد زیر را به متد Page_Load اضافه کنید، تا مطمئن شویم که پیغام خطا هنگامی که خطایی برای گزارش وجود ندارد قابل نمایش نیست. گرداننده های رخداد برای رخدادهای DataBound و SelectedIndexChanged مربوط به Dropdown list استاد برای پر کردن dropdown list های Course اضافه کنید:

```

protected void Page_Load(object sender, EventArgs e)
{
    CourseAssignedLabel.Visible = false;
    CourseRemovedLabel.Visible = false;
}
protected void InstructorsDropDownList_DataBound(object sender, EventArgs e)
{
    PopulateDropDownLists();
}
protected void InstructorsDropDownList_SelectedIndexChanged(object sender, EventArgs e)
{
    PopulateDropDownLists();
}

```

ASSIGN INSTRUCTORS TO COURSES OR REMOVE FROM COURSES

Select an Instructor:

ASSIGN A COURSE

Select a Course:

REMOVE A COURSE

Select a Course:

یک استاد را انتخاب کنید. لیست کشویی **Assign a Course** لیست دوره هایی را نشان می دهد که استاد آنها را تدریس نمی کند، و لیست کشویی **Remove a Course** لیست دوره هایی را نشان می دهد که در حال حاضر به استاد اختصاص داده شده است. در قسمت **Remove a Course** یک **Course** را انتخاب کرده و روی دکمه **Assign** کلیک کنید. این دوره به لیست کشویی **Remove a Course** منتقل می شود. یک دوره را از قسمت **Remove a Course** انتخاب و دکمه **Remove** را کلیک کنید. دوره انتخابی به لیست کشویی **Assign a Course** منتقل می شود.

تا الان شما روش های مختلف کار با داده های وابسته را مشاهده کردید. در بخش بعد چگونگی استفاده از وراثت را در مدل داده به منظور بهبود مانایی برنامه کاربردی خود، خواهید آموخت.

پیاده سازی وراثت Table-per-Hierarchy

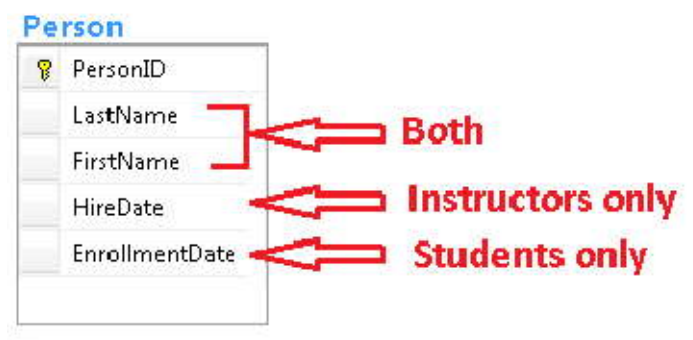
در بخش قبل، شما با داده های مرتبط به وسیله اضافه و حذف کردن رابطه ها و اضافه کردن یک موجودیت دارای ارتباط با موجودیتی که هم اکنون در مدل داده وجود دارد، کار کردید. در این بخش به شما نشان داده خواهد شد که چگونه وراثت را در مدل داده پیاده سازی کنید.

در برنامه نویسی شی گرا، شما می توانید از وراثت به منظور ساده سازی کار با کلاس های دارای ارتباط، استفاده کنید. برای مثال می توانید کلاس های **Instructor** و **Student** که از یک کلاس **Person** مشتق شده اند را ایجاد کنید. و می توانید همین گونه ساختارهای وراثتی بین موجودیت های **Entity Framework** ایجاد کنید.

در این قسمت شما صفحه وب جدیدی ایجاد نخواهید کرد. در عوض موجودیت های مشتق شده ای را به مدل داده اضافه خواهید کرد. و همچنین صفحه های موجود را به منظور استفاده از این موجودیت ها دستکاری خواهید کرد.

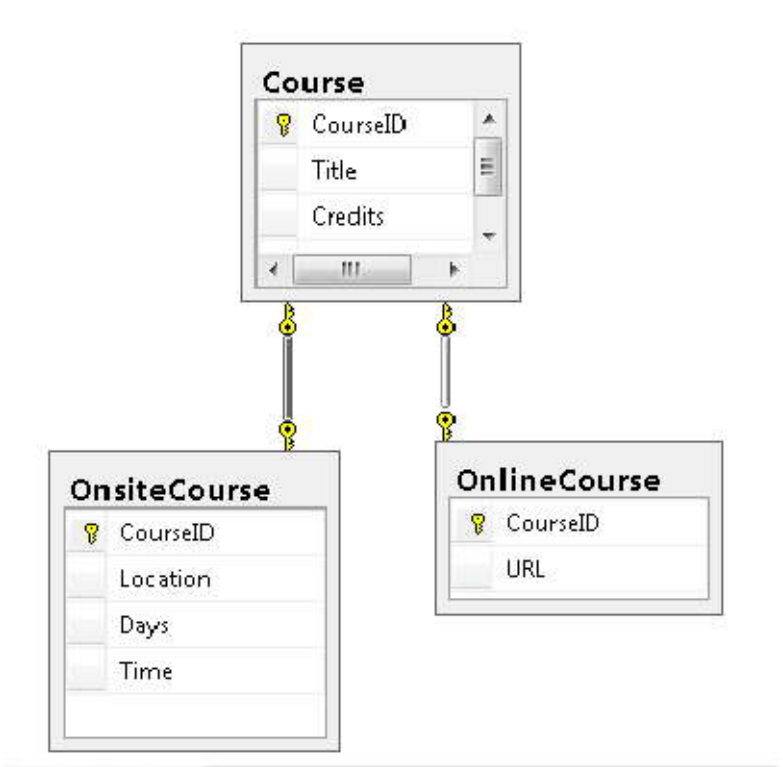
وراثت Table-per-Type در مقابل Table-per-Hierarchy

یک دیتابیس می تواند اطلاعات مربوط به اشیاء مرتبط را در یک یا چند جدول ذخیره کند. برای مثال در دیتابیس School، جدول Person حاوی اطلاعات هر دو موجودیت student و Instructor در یک جدول است. بعضی ستون ها فقط برای اساتید در نظر گرفته شده (HireDate)، بعضی فقط برای دانشجویها (EnrollmentDate)، و بعضی، برای هر دو (LastName، FirstName).



شما می توانید Entity Framework را برای ایجاد موجودیت های Instructor و Student که از موجودیت Person ارث می برند، تنظیم کنید. این ساختار وراثت یک موجودیت از یک جدول، وراثت (TPH) *table-per-hierarchy* نامیده می شود.

برای موجودیت های course، دیتابیس School از الگوی متفاوتی استفاده می کند. دوره های onsite و online در جدول های متفاوتی ذخیره شده اند، هر کدام کلید خارجی دارند که به جدول Course اشاره می کند. اطلاعات مشترک هر دو نوع Course تنها در جدول Course ذخیره می شوند.



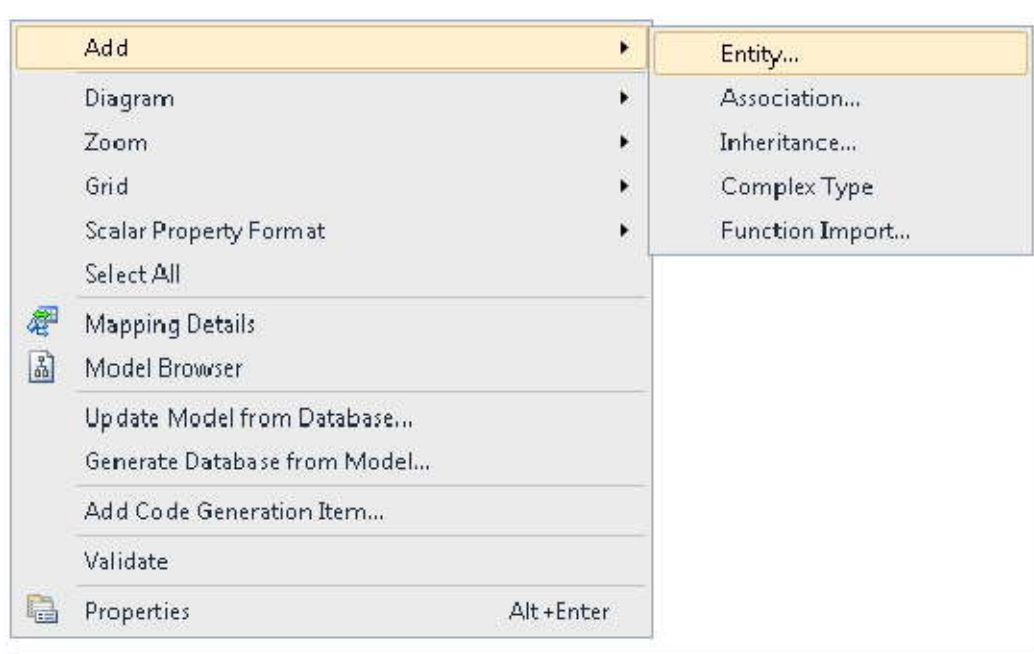
شما می توانید مدل داده Entity Framework را تنظیم کنید، تا موجودیت های OnsiteCourse و OnlineCourse از موجودیت Course ارث ببرند. این ساختار وراثت یک موجودیت از جدول های جداگانه با نوع های مختلف، که هر کدام از جدول های جداگانه به جدولی اشاره می کنند که داده های مشترک انواع دیگر را ذخیره می کند، *table per talbe(tpt)* نامیده می شود.

الگوهای وراثت TPH نسبت به الگو وراثت TPT، عموماً کارایی بهتری را در Entity Framework ارائه می دهد ، چون TPT می تواند منجر به کوئری های join پیچیده شود. این بخش از آموزش نشان می دهد که چگونه وراثت TPH را پیاده سازی کنید. شما این کار را با اجرای مراحل زیر انجام خواهید داد:

- یک موجودیت از نوع Instructor و Student که از Person مشتق شده ایجاد کنید.
- خصوصیت هایی را که به موجودیت های مشتق شده وابستگی دارند را از موجودیت Person به موجودیت های مشتق شده انتقال دهید.
- روی خصوصیت های انواع مشتق شده محدودیت قرار دهید.
- موجودیت Person را از نوع Abstract قرار دهید.
- هر کدام از موجودیت های مشتق شده را به جدول Person نگاشت کنید. به همراه شرطی که مشخص می کند آیا یک سطر Person، نوع (موجودیت) مشتق شده مورد نظر را ارائه می دهد یا خیر.

اضافه کردن موجودیت های Student و Instructor

فایل *SchoolModel.edmx* را باز کرده و در ناحیه خالی از محیط طراحی راست کلیک کرده، **Add** و سپس **Entity** را انتخاب کنید.

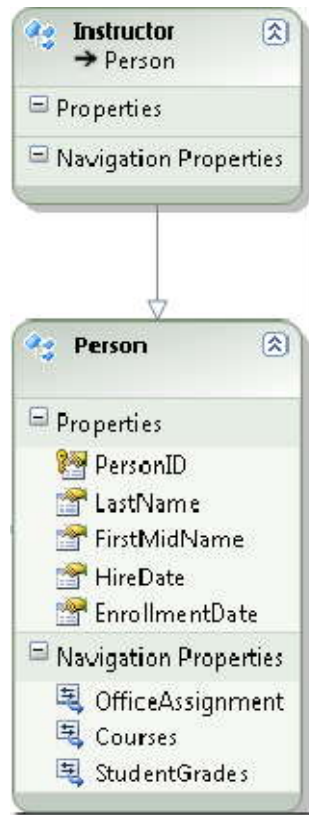


در کادر محاوره ای **Add Entity**، نام این موجودیت جدید را Instructor قرار داده و گزینه **Base type** آن را برابر Person قرار دهید.

The screenshot shows the 'Add Entity' dialog box. The 'Entity name' is 'Instructor', the 'Base type' is 'Person', and the 'Entity Set' is 'People'. In the 'Key Property' section, the 'Create key property' checkbox is unchecked, the 'Property name' is 'Id', and the 'Property type' is 'Int32'.

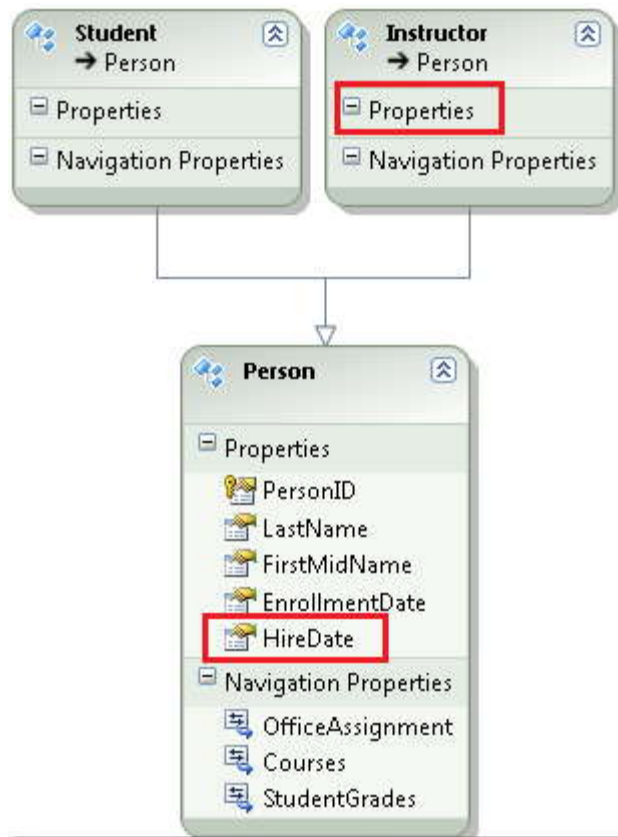
روی OK کلیک کنید. Designer (محیط طراحی مدل داده) یک موجودیت Instructor که از موجودیت Person مشتق شده، ایجاد می کند. موجودیت جدید هنوز هیچ خصوصیتی¹ ندارد.

¹ properties

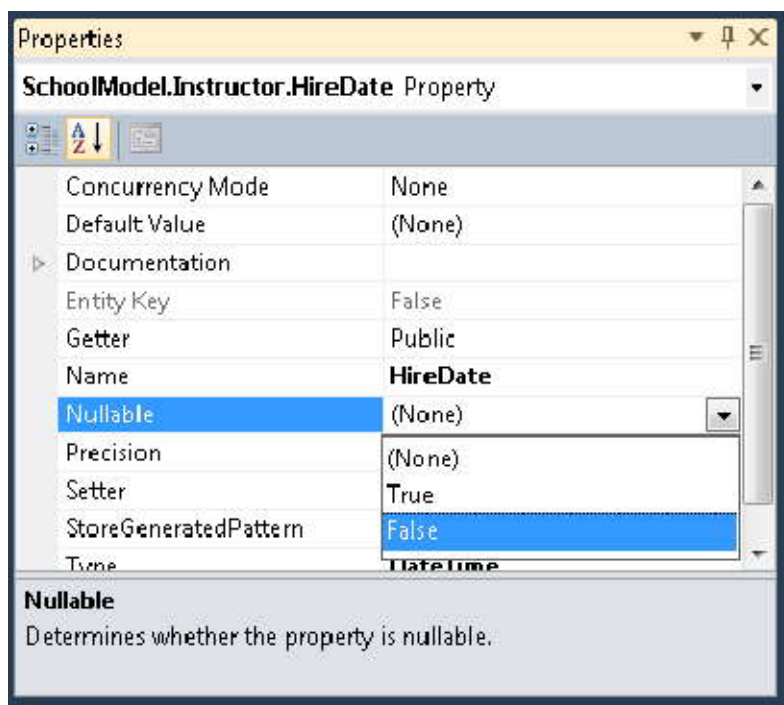


همین فرآیند را برای ایجاد موجودیت Student که از Person مشتق شده تکرار کنید.

فقط اساتید دارای hire date هستند بنابراین شما نیاز دارید که این خصوصیت را از موجودیت Person به موجودیت Instructor منتقل کنید. در موجودیت Person روی خصوصیت HireDate راست کلیک کرده و روی **Cut** کلیک کنید. سپس روی **Properties** موجودیت Instructor راست کلیک کرده و **Paste** را انتخاب کنید.

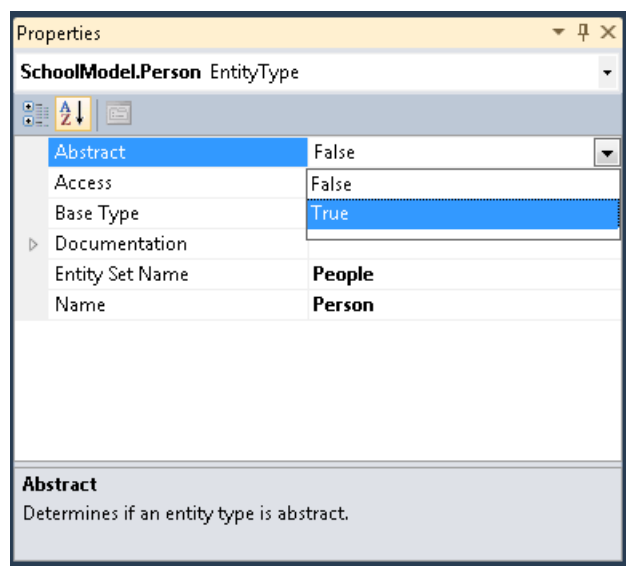


HireDate موجودیت Instructor نمی تواند Null باشد. روی خصوصیت HireDate راست کلیک کرده، روی **Properties** کلیک کنید و سپس در پنجره **Properties** مقدار Nullable را به False تغییر دهید.



این فرآیند را به منظور انتقال خصوصیت EnrollmentDate از موجودیت Person به موجودیت Student تکرار کنید. مطمئن شوید که برای خصوصیت EnrollmentDate مقدار Nullable را برابر False تنظیم کرده اید.

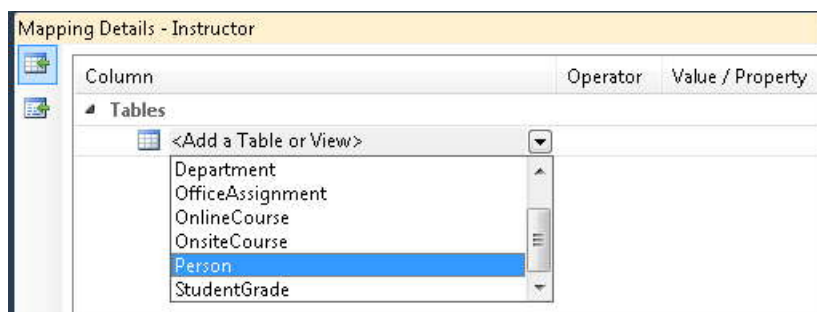
حال که یک موجودیت Person فقط دارای خصوصیت های مشترک موجودیت های Instructor و Student است(غیر از خصوصیت های Navigation که انتقالشان نمی دهید)، می تواند تنها، به عنوان موجودیت پایه در ساختار وراثت استفاده شود. بنابراین شما نیاز دارید مطمئن شوید که این موجودیت به عنوان موجودیت مستقل عمل نمی کند. روی موجودیت Person راست کلیک کرده، **Properties** را انتخاب کنید، در پنجره **Properties** مقدار خصوصیت **Abstract** را به **True** تغییر دهید.



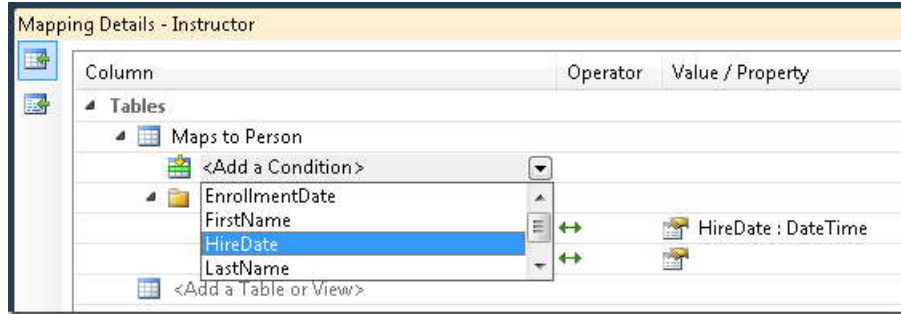
نگاشت موجودیت های Instructor و Student به جدول Person

حال نیاز دارید به Entity Framework بگویید که چگونه بین موجودیت های Instructor و Student در دیتابیس تفاوت قائل شود.

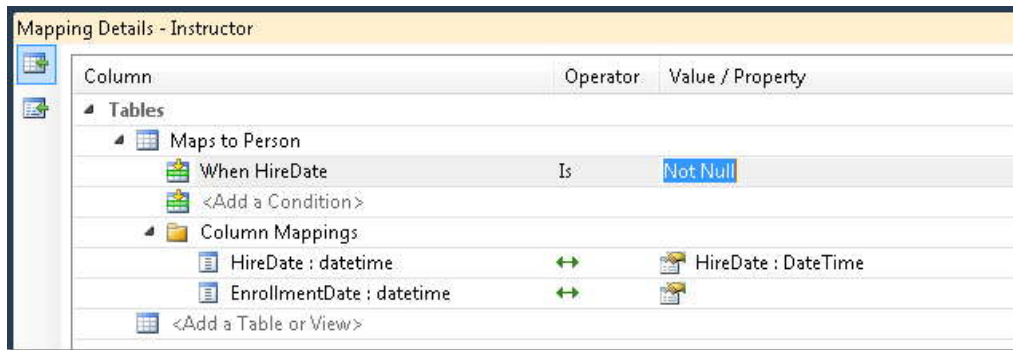
روی موجودیت Instructor راست کلیک کرده، **Table Mapping** را انتخاب کنید. در پنجره **Mapping Details** روی **Add** **a Table or View** کلیک کرده و **Person** را انتخاب کنید.



روی **Add a Condition** کلیک کرده و **HireDate** را انتخاب کنید.



مقدار **Operator** را به **Is** و مقدار **Value / Property** را به **Not Null** تغییر دهید.

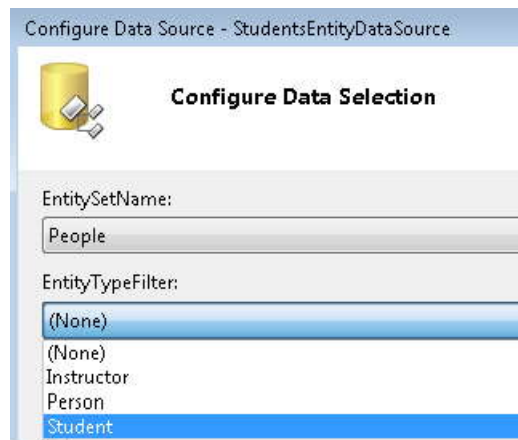


این فرآیند را برای موجودیت **Student** تکرار کنید، تا مشخص کند، که این موجودیت زمانی به جدول **Person** نگاشت می شود که ستون **EnrollmentDate** برابر **Null** نباشد. سپس مدل داده را ذخیره کرده و ببندید.

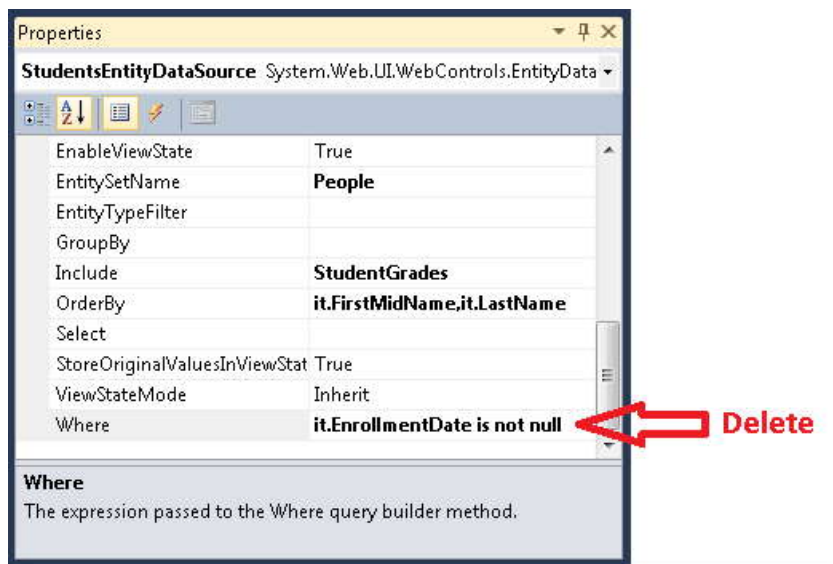
پروژه را به منظور ایجاد موجودیت های جدید به عنوان کلاس و قابل دسترس کردن آنها در محیط طراحی، **Build** کنید.

استفاده از موجودیت های Student و Instructor

زمانی صفحه ای برای کار با داده های student و instructor صفحه ای ایجاد می کردید، آن موجودیت ها را به مجموعه موجودیت Person مقید کرده، و به منظور محدود کردن داده های برگشتی به instructor و Student بر روی خصوصیت های HireDate و EnrollmentDate فیلتر انجام می دادید. هر چند، در حال حاضر، هر زمان که هر کنترل data source را به مجموعه موجودیت Person مقید کنید، می توانید تعیین کنید که نوع موجودیت Student یا instructor باید انتخاب شود. چون Entity Framework می داند که چگونه بین student و instructor در مجموعه موجودیت Person تفاوت قائل شود، شما می توانید تنظیمات خصوصیت Where را که به این منظور به صورت دستی وارد کرده اید، حذف کنید.



و در پنجره **Properties** شما می توانید مقادیر عبارت **Where** را که دیگر نیازی به آن ندارید، حذف کنید. همان طور که در مثال زیر نشان داده شده است.



هر چند، چون کد ساختیافته مربوط به کنترل EntityDataSource را برای استفاده از صفت ContextTypeName تغییر داده اید، شما نمی توانید ویزارد **Configure Data Source** را روی کنترل های EntityDataSource ای که تا کنون ایجاد کرده اید، اجرا کنید. بنابراین، به جای آن، باید تغییرات مورد نیاز را با تغییر دادن کد markup انجام دهید.

صفحه *Students.aspx* را باز کنید، در کنترل *StudentsEntityDataSource* صفت *Where* را حذف و صفت *EntityTypeFilter="Student"* را اضافه کنید. کد ساختیافته اکنون باید شبیه مثال زیر باشد:

```
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableDelete="True" EnableFlattening="False" EnableUpdate="True"
    EntitySetName="Person" Include="StudentGrade" OrderBy="it.LastName"
    EntityTypeFilter="Student">
</asp:EntityDataSource>
```

تنظیم صفت *EntityTypeFilter* باعث می شود اطمینان حاصل کنیم که کنترل *EntityDataSource* تنها، نوع موجودیت مورد نظر را انتخاب خواهد کرد. اگر می خواهید هر دو نوع موجودیت های *Student* و *Instructor* را برگردانید، نباید این صفت را تنظیم کنید. (فقط زمانی که از کنترل *EntityDataSource* برای دستیابی به داده های فقط خواندنی¹ استفاده می کنید، می توانید چند نوع موجودیت را با یک کنترل *EntityDataSource* برگردانید. اگر از کنترل *EntityDataSource* برای درج، بروزرسانی و حذف موجودیت ها استفاده می کنید و اگر مجموعه موجودیت، طوری مقید

¹ read-only

شده¹ است تا حاوی چند نوع موجودیت باشد، شما فقط می توانید با یک نوع موجودیت کار کنید، و باید صفت `EntityTypeFilter` را تنظیم کنید.)

این فرآیند را برای کنترل `SearchEntityDataSource` تکرار کنید، به جای حذف کامل صفت `Where` فقط قسمتی از این صفت، که موجودیت های `Student` را انتخاب می کند، حذف کنید. تگ باز این کنترل مانند مثال زیر خواهد بود:

```
<asp:EntityDataSource ID="SearchEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Person" EntityTypeFilter="Student"
Where="it.FirstMidName Like '%' + @StudentName + '%' or it.LastName Like '%' +
@StudentName + '%'">
```

صفحه را اجرا کنید، تا مطمئن شوید مانند گذشته کار می کند.

¹ bound

STUDENT LIST

	ID	Name	Enrollment Date	Number of Courses
Edit Delete	14	Walker, Alexandra	9/1/2000	2
Edit Delete	30	Shan, Alicia	9/1/2003	2
Edit Delete	28	White, Anthony	9/1/2001	2
Edit Delete	13	Anand, Arturo	9/1/2003	2
Edit Delete	22	Alexander, Carson	9/1/2005	3
Edit Delete	19	Bryant, Carson	9/1/2001	1
Edit Delete	15	Powell, Carson	9/1/2004	1
Edit Delete	26	Rogers, Cody	9/1/2002	2
Edit Delete	16	Jai, Damien	9/1/2001	1
Edit Delete	33	Gao, Erica	1/30/2003	0

[1](#) [2](#) [3](#)

FIND STUDENTS BY NAME

Enter any part of the name

Name	Enrollment Date
Barzdukas, Gytis	9/1/2005
Justice, Peggy	9/1/2001
Li, Yan	9/1/2002
Norman, Laura	9/1/2003
Olivotto, Nino	9/1/2005
Tang, Wayne	9/1/2005
Alonso, Meredith	9/1/2002
Lopez, Sophia	9/1/2004
Browning, Meredith	9/1/2000
Anand, Arturo	9/1/2003

[1](#) [2](#) [3](#)

صفحات زیر را که در بخش های پیشین ایجاد کرده اید تغییر دهید تا به جای استفاده از موجودیت های **Person** از موجودیت های جدید **Student** و **Instructor** استفاده کنند، سپس آنها را اجرا کنید تا ببینید همانند گذشته کار می کنند یا نه.

- در *StudentsAdd.aspx* صفت `EntityTypeFilter="Student"` را به کنترل `StudentsEntityDataSource` اضافه کنید. کد ساختیافته اکنون باید مانند مثال زیر باشد:

```
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EnableInsert="True" EntitySetName="Person" EntityTypeFilter="Student">
</asp:EntityDataSource>
```


ADD NEW STUDENTS

First Name	<input type="text"/>
Last Name	<input type="text"/>
Enrollment Date	<input type="text"/>
<input type="button" value="Insert"/> <input type="button" value="Cancel"/>	

- در `About.aspx` صفت `EntityTypeFilter="Student"` را به کنترل `StudentStatisticsEntityDataSource` اضافه و `Where="it.EnrollmentDate is not null"` را حذف کنید:

```
<asp:EntityDataSource ID="StudentStatisticsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="Person" EntityTypeFilter="Student"
    GroupBy="it.EnrollmentDate" OrderBy="it.EnrollmentDate"
    Select="it.EnrollmentDate, Count(it.EnrollmentDate) AS NumberOfStudents" >
</asp:EntityDataSource>
```

STUDENT BODY STATISTICS

Date of Enrollment	Students
9/1/2000	2
9/1/2001	5
9/1/2002	3
1/30/2003	1
9/1/2003	3
9/1/2004	5
9/1/2005	6
1/1/2011	1

- در `Instructors.aspx` و `InstructorsCourses.aspx` صفت `EntityTypeFilter="Instructor"` را به کنترل `InstructorsEntityDataSource` اضافه و `Where="it.HireDate is not null"` را حذف کنید. کد ساختیافته `Instructors.aspx` اکنون مانند مثال زیر خواهد بود:

```
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Person" EntityTypeFilter="Instructor"
Where="it.HireDate is not null" Include="OfficeAssignment" EnableUpdate="True">
</asp:EntityDataSource>
```

INSTRUCTORS

	ID	Name	Hire Date	Office Assignment
Edit Select	1	Abercrombie, Kim	3/11/1995	17 Smith
Edit Select	4	Fakhouri, Fadi	8/6/2002	29 Adams
Edit Select	5	Harui, Roger	7/1/1998	37 Williams
Edit Select	18	Zheng, Roger	2/12/2004	143 Smith
Edit Select	25	Kapoor, Candace	1/15/2001	57 Adams
Edit Select	27	Serrano, Stacy	6/1/1999	271 Williams
Edit Select	31	Stewart, Jasmine	10/12/1997	131 Smith
Edit Select	32	Xu, Kristen	7/23/2001	203 Williams
Edit Select	34	Van Houten, Roger	12/7/2000	213 Smith

COURSES TAUGHT

	ID	Title	Department
Select	2042	Literature	English
Select	3141	Trigonometry	Mathematics
Select	4022	Microeconomics	Economics
Select	4041	Macroeconomics	Economics
Select	4061	Quantitative	Economics
Select	4062	New engineering course	Engineering
Select	4063	new course	Economics

COURSE DETAILS

ID	2042
Title	Literature
Credits	4
Department	English
Location	225 Adams
URL	

STUDENT GRADES

ID	Name	Grade
6	Li, Yan	3.50
7	Norman, Laura	4.00
8	Olivotto, Nino	3.00

کد ساختنیافته *InstructorsCourses.aspx* اکنون مانند مثال زیر خواهد بود:

```
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
EntitySetName="Person" EntityTypeFilter="Instructor"
Where="it.HireDate is not null" Select="it.LastName + ', ' +
it.FirstMidName AS Name,it.PersonID">
</asp:EntityDataSource>
```

ASSIGN INSTRUCTORS TO COURSES OR REMOVE FROM COURSES

Select an Instructor:

ASSIGN A COURSE

Select a Course:

REMOVE A COURSE

Select a Course:

با انجام این تغییرات، به روش های متعدد، مانایی برنامه کاربردی Contoso University را بهبود بخشیدید. شما منطق انتخاب و اعتبارسنجی را از لایه واسط کاربری^۱ بیرون بردید. (.aspx markup) و آن را به بخش جدایی ناپذیر از لایه دسترسی به داده^۲ تبدیل کردید. این کار به ایزوله کردن کد برنامه شما در مقابل تغییراتی که ممکن است در آینده در شمای دیتابیس^۳ یا مدل داده^۴ ایجاد کنید، کمک می کند. برای مثال، می توانید تصمیم بگیرید که دانشجویان بتوانند به عنوان دستیار استاد استخدام شوند و بنابراین Hire Date خواهند داشت. شما سپس می توانید یک خصوصیت جدید برای متمایز کردن دانشجویان از اساتید و بروزرسانی مدل داده اضافه کنید. هیچ کدی در برنامه کاربردی نیاز به تغییر ندارد به جز جاهایی که می خواهید HireDate را برای دانشجویان نمایش دهید. مزیت دیگر اضافه کردن موجودیت های **Instructor** و **Student** این است که کد شما نسبت به زمانی که به اشیا **Person** (که در واقع **Student** و **Instructor** هستند) رجوع می کند، قابل فهم تر است.

تا الان یک روش برای پیاده سازی الگوی وراثت در Entity Framework را مشاهده کردید. در بخش بعد، چگونگی استفاده از رویه های ذخیره شده^۵ را به منظور داشتن کنترل بیشتر روی چگونگی دسترسی Entity Framework به دیتابیس. خواهید آموخت.

¹ UI Layer

² data access Layer

³ database schema

⁴ data model

⁵ stored procedure

استفاده از رویه های ذخیره شده

در بخش قبل الگوی وراثت `table-per-hierarchy` را پیاده سازی کردید. در این بخش چگونگی استفاده از رویه های ذخیره شده را به منظور داشتن کنترل بیشتر روی دسترسی به دیتابیس خواهید آموخت.

Entity Framework به شما اجازه می دهد تعیین کنید، که برای دسترسی به دیتابیس باید از رویه های ذخیره شده استفاده کند. برای هر نوع موجودیت می توانید رویه ذخیره شده ای برای ایجاد، بروزرسانی و حذف موجودیت های آن نوع، مشخص کنید. سپس به منظور انجام وظایفی، مانند برگرداندن مجموعه ای از موجودیت ها، می توانید در مدل داده، ارجاع هایی را به رویه ذخیره شده اضافه کنید.

استفاده از رویه های ذخیره شده نیاز معمولی است برای دسترسی به دیتابیس. در بعضی موارد ممکن است، مدیر دیتابیس نیاز داشته باشد تمام دسترسی ها به دیتابیس، به خاطر دلایل امنیتی از طریق رویه های ذخیره شده انجام گیرد. در موارد دیگر ممکن است بخواهید منطق تجاری¹ را در بعضی پروسس هایی که **Entity Framework** از آنها در موقع بروزرسانی دیتابیس استفاده می کند، ایجاد کنید. به طور مثال، موقعی که یک موجودیت حذف شده است ممکن است بخواهید آن را به یک دیتابیس آرشیو کپی کنید. یا زمانی که یک سطر بروزرسانی می شود، ممکن است بخواهید سطری در جدول **Logging** ایجاد کنید که مشخص کند چه کسی رکوردها را تغییر داده. این گونه وظایف² را در یک رویه ذخیره شده می توان انجام داد که هر وقت که **Entity Framework** موجودیتی را حذف یا بروزرسانی می کند صدا زده می شود.

مانند بخش قبل شما صفحه جدیدی ایجاد نخواهید کرد. در عوض برای بعضی صفحاتی که قبلا ایجاد کرده اید، روش دستیابی **Entity Framework** به دیتابیس را تغییر خواهید داد.

در این بخش رویه های ذخیره شده ای را در دیتابیس به منظور درج موجودیت های **Student** و **Instructor** ایجاد خواهید کرد. آنها را به مدل داده اضافه خواهید کرد، و تعیین می کنید که **Entity Framework** باید برای درج موجودیت های **Student** و **Instructor** در دیتابیس باید از آنها استفاده کند. همچنین رویه ذخیره شده ای را برای برگرداندن موجودیت های **Course** ایجاد خواهید کرد.

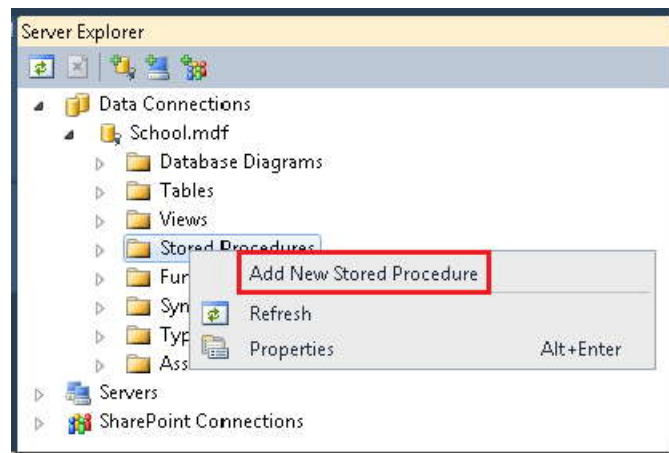
¹ Business logic

² task

ایجاد رویه های ذخیره شده در دیتابیس

(اگر از فایل *School.mdf* دانلود شده استفاده می کنید، می توانید از این بخش صرف نظر کنید چون رویه های ذخیره شده هم اکنون در آن ایجاد و ذخیره شده اند.)

در **Server Explorer**، گزینه *School.mdf* را گسترش دهید، روی **Stored Procedures** راست کلیک کرده و **Add New** **Stored Procedure** را انتخاب کنید.



دستورات SQL زیر را کپی کرده و درون پنجره stored procedure بچسبانید (paste). اسکلت رویه ذخیره شده را جایگزین نمائید.

```
CREATE PROCEDURE [dbo].[InsertStudent]
@LastName nvarchar(50),
@FirstName nvarchar(50),
@EnrollmentDate datetime
AS
INSERT INTO dbo.Person(LastName,
FirstName,
EnrollmentDate)
VALUES (@LastName,
@FirstName,
@EnrollmentDate);
SELECT SCOPE_IDENTITY() as NewPersonID;
```

```
dbo.StoredProce...DATA\SCHOOL.MDF)* x
1 CREATE PROCEDURE [dbo].[InsertStudent]
2     @LastName nvarchar(50),
3     @FirstName nvarchar(50),
4     @EnrollmentDate datetime
5 AS
6     INSERT INTO dbo.Person (LastName,
7                             FirstName,
8                             EnrollmentDate)
9     VALUES (@LastName,
10            @FirstName,
11            @EnrollmentDate);
12     SELECT SCOPE_IDENTITY() as NewPersonID;
```

موجودیت های **Student** دارای چهار خصوصیت **PersonID**, **LastName**, **FirstName** و **EnrollmentDate** هستند. دیتابیس به طور خودکار مقدار ID را تولید می کند. و رویه ذخیره شده، برای سه خصوصیت دیگر پارامترهایی را می پذیرد. رویه ذخیره شده کلید رکورد جدید را برمی گرداند تا Entity Framework بتواند آخرین نسخه از آن موجودیت را در حافظه نگه داری کند.

پنجره Stored Procedure را ذخیره کرده و ببندید.

به همین شیوه یک رویه **InsertInstructor** ایجاد کنید. با استفاده از دستور **SQL** زیر:

```
CREATE PROCEDURE [dbo].[InsertInstructor]
@LastName nvarchar(50),
@FirstName nvarchar(50),
@HireDate datetime
AS
INSERT INTO dbo.Person(LastName,
FirstName,
HireDate)
VALUES (@LastName,
@FirstName,
@HireDate);
SELECT SCOPE_IDENTITY() as NewPersonID;
```

همچنین رویه های Update را برای موجودیت های Student و Instructor ایجاد کنید. (اکنون دیتابیس دارای رویه DeletePerson است که برای هر دو موجودیت های Instructor و Student کار می کند.)

```
CREATE PROCEDURE [dbo].[UpdateStudent]
@PersonIDint,
@LastName nvarchar(50),
@FirstName nvarchar(50),
@EnrollmentDate datetime
AS
UPDATE Person SET LastName=@LastName,
FirstName=@FirstName,
EnrollmentDate=@EnrollmentDate
WHERE PersonID=@PersonID;
CREATE PROCEDURE [dbo].[UpdateInstructor]
@PersonID int,
@LastName nvarchar(50),
@FirstName nvarchar(50),
@HireDate datetime
AS
UPDATE Person SET LastName=@LastName,
FirstName=@FirstName,
HireDate=@HireDate
WHERE PersonID=@PersonID;
```

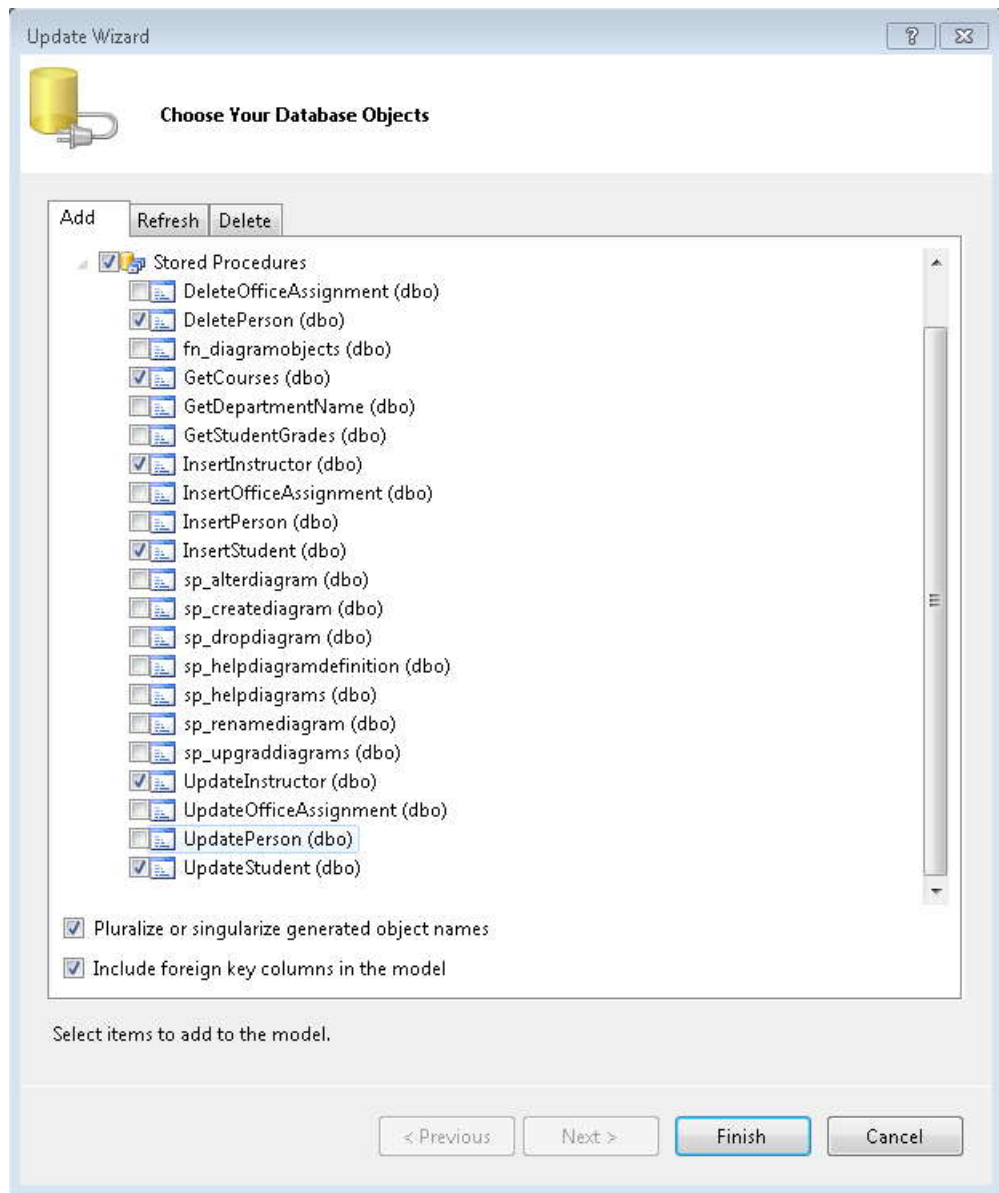
در این بخش سه تابع Insert، Update و Delete را برای هر نوع موجودیت نگاشت می کنیم. 4 Entity Framework به شما اجازه نگاشت یک یا دو مورد از این توابع را بدون نگاشت توابع دیگر، به رویه های ذخیره شده می دهد، با یک استثنا: اگر تابع update را بدون نگاشت تابع Delete، نگاشت دهید هنگامی که می خواهید موجودیتی را حذف کنید Entity Framework یک استثنا پرتاب خواهد کرد. در Entity Framework 3.5 اینگونه قابلیت انعطافی را در نگاشت رویه های ذخیره شده نداشتید: اگر تابعی را نگاشت می دادید مجبور بودید هر سه تابع را نگاشت دهید.

برای ایجاد رویه ذخیره شده ای که به جای بروزرسانی، داده ها را بخواند، با استفاده از دستور SQL زیر یک رویه ایجاد کنید که همه موجودیت های Course را انتخاب کنید:

```
CREATE PROCEDURE [dbo].[GetCourses]
AS
SELECT CourseID, Title, Credits, DepartmentID FROM dbo.Course
```

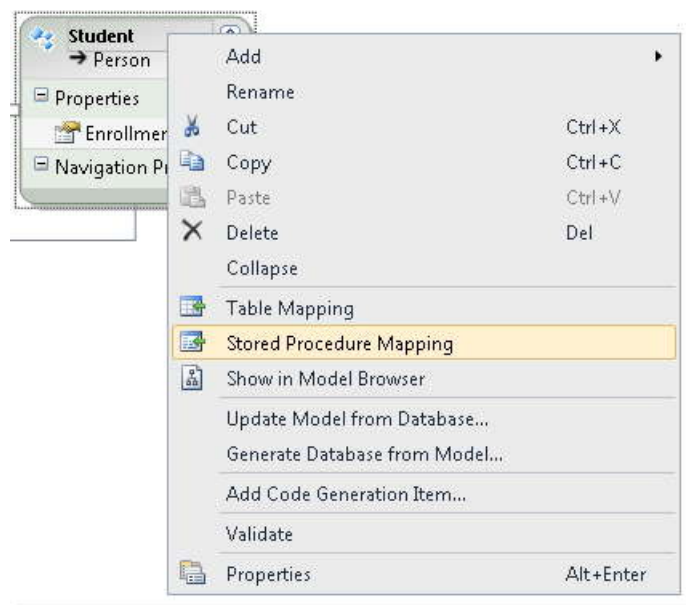
اضافه کردن رویه ذخیره شده به مدل داده

رویه های ذخیره شده، اکنون در دیتابیس ایجاد شدند، اما باید به مدل داده نیز اضافه شوند، تا برای Entity Framework قابل دستیابی باشند. *SchoolModel.edmx* را باز کرده، در محیط طراحی راست کلیک کرده و **Update Model from Database** را انتخاب کنید. در برگه **Add** کادر محاوره ای **Choose Your Database Objects**، گزینه **Stored Procedures** را گسترش داده، و رویه های ذخیره شده ای که تازه ایجاد کرده اید و رویه **DeletePerson** را انتخاب کرده و سپس روی **Finish** کلیک کنید.

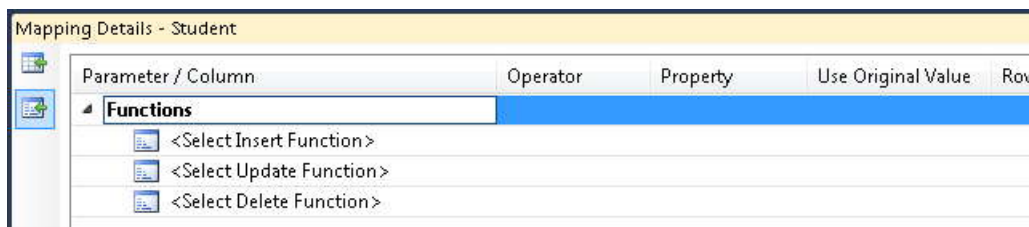


نگاشت رویه های ذخیره شده

در محیط طراحی مدل داده، روی موجودیت **Student** راست کلیک کرده، **Stored Procedure Mapping** را انتخاب کنید.

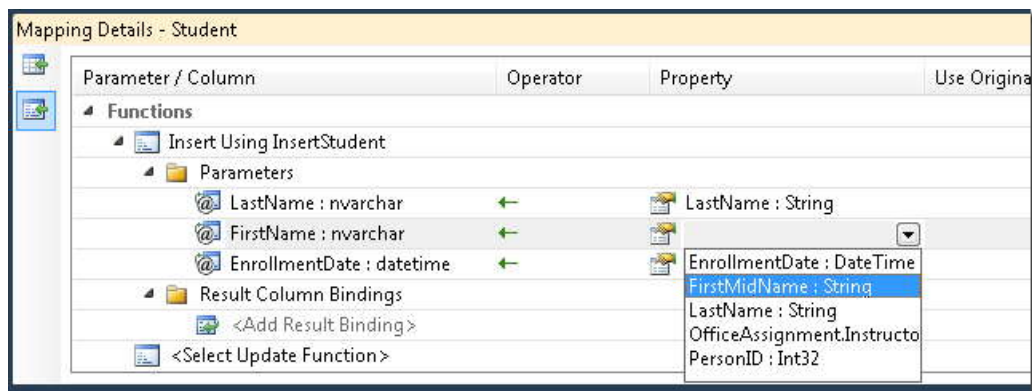


پنجره **Mapping Details** ظاهر می شود، که در آن می توانید رویه های ذخیره شده ای را مشخص کنید که Entity Framework باید از آنها برای درج، بروزرسانی و حذف موجودیت های این نوع استفاده کند.

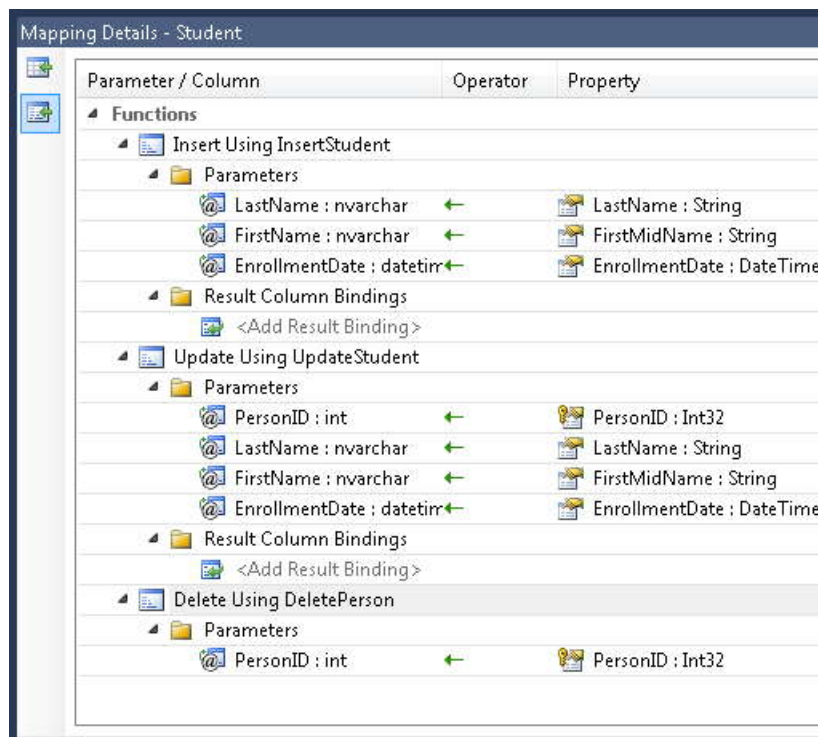


Insert function را برابر **InsertStudent** قرار دهید. این پنجره، لیستی از پارامترهای رویه های ذخیره شده را نشان می دهد، هر کدام باید به یک خصوصیت موجودیت نگاشت شوند. دو تای آنها به طور خودکار نگاشت می شوند چون نام آنها مشابه است. هیچ خصوصیت موجودیتی با نام **FirstName** وجود ندارد، بنابراین شما به طور دستی باید **FirstMidName** را از لیست کشویی

که خصوصیت های موجودیت های قابل دسترس را نشان می دهد انتخاب کنید. (این کار به خاطر این است که قبلا نام ویژگی FirstName را به FirstMidName تغییر داده اید.)



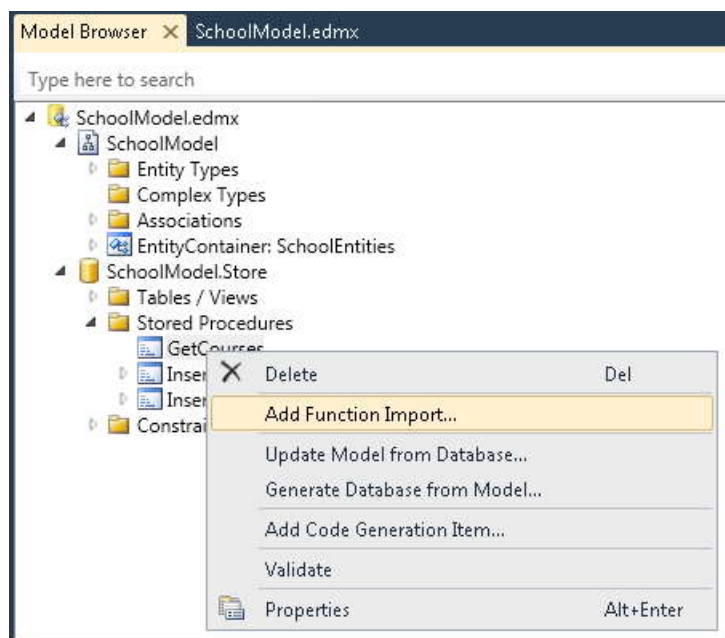
در همان پنجره **Mapping Details**، تابع **Update** را به رویه ذخیره شده **UpdateStudent** (اطمینان حاصل کنید که مقدار پارامتر **FirstName** برابر **FirstMidName** انتخاب شود، همان گونه که در مورد رویه ذخیره شده **Insert** انجام دادید) و تابع **Delete** را به رویه ذخیره شده **DeletePerson** نگاشت دهید.



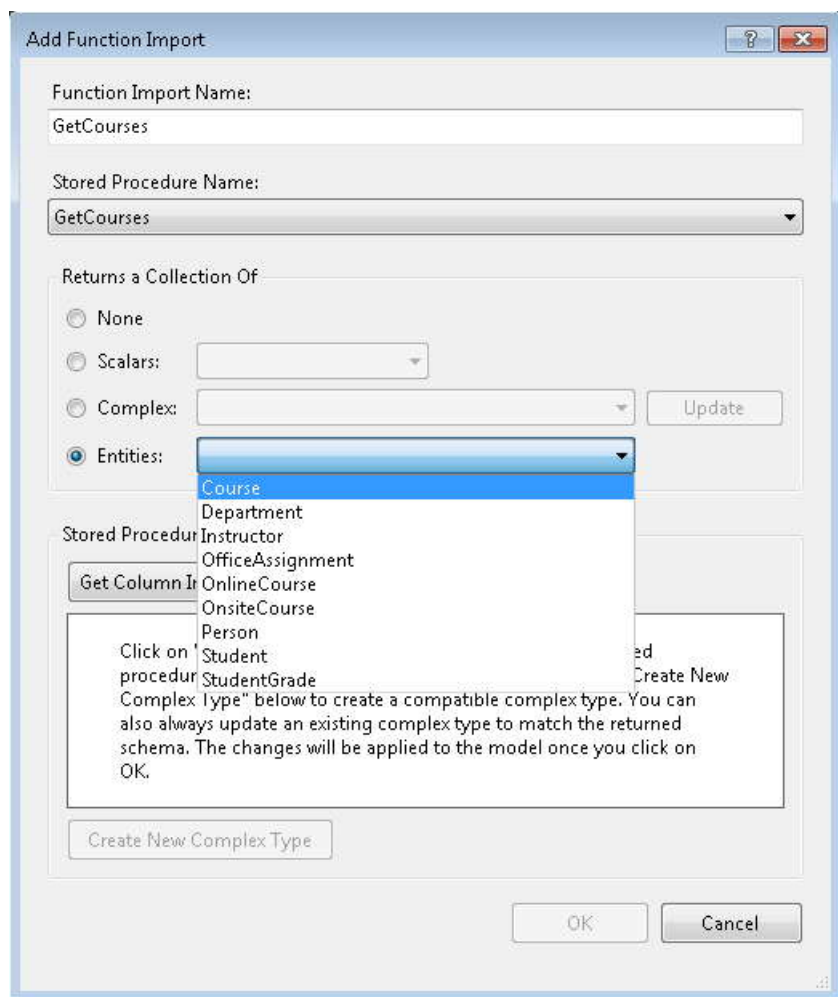
همین فرآیند را در مورد نگاشت رویه های ذخیره شده درج، حذف و بروزرسانی (برای اساتید) به موجودیت **Instructor** انجام دهید.

Parameter / Column	Operator	Property
Functions		
Insert Using InsertInstructor		
Parameters		
@ LastName : nvarchar	←	LastName : String
@ FirstName : nvarchar	←	FirstMidName : String
@ HireDate : datetime	←	HireDate : DateTime
Result Column Bindings		
<Add Result Binding>		
Update Using UpdateInstructor		
Parameters		
@ PersonID : int	←	PersonID : Int32
@ LastName : nvarchar	←	LastName : String
@ FirstName : nvarchar	←	FirstMidName : String
@ HireDate : datetime	←	HireDate : DateTime
Result Column Bindings		
<Add Result Binding>		
Delete Using DeletePerson		
Parameters		
@ PersonID : int	←	PersonID : Int32

برای رویه هایی که به جای بروزرسانی، داده ها را می خوانند، از پنجره **Model Browser** برای نگاشت آنها به نوع موجودیتی که برمی گرداند، استفاده کنید. در محیط طراحی مدل داده، در جای خالی راست کلیک کرده و **Model Browser** را انتخاب کنید. گره **SchoolModel.Store** و سپس **Stored Procedures** را باز کنید. سپس روی رویه ذخیره شده **GetCourses** راست کلیک کرده و **Add Function Import** را انتخاب کنید.



در کادر محاوره ای **Add Function Import** در قسمت **Returns a Collection Of** گزینه **Entities** و سپس **Course** را به عنوان موجودیت برگشتی انتخاب کنید. پس از انجام این کار روی **OK** کلیک کرده و فایل **.edmx** را ذخیره کنید و ببندید.



استفاده از رویه های ذخیره شده ی درج، بروزرسانی و حذف

پس از آنکه رویه های ذخیره شده درج، بروزرسانی و حذف را به مدل داده اضافه کردید و آنها را به موجودیت های مناسب نگاشت دادید، به طور خودکار توسط Entity Framework استفاده می شوند. حالا می توانید صفحه *StudentsAdd.aspx* را اجرا کنید. هر وقت که یک دانشجوی جدید اضافه می کنید Entity Framework از رویه ذخیره شده *InsertStudent* برای اضافه کردن سطر جدید به جدول *Student* استفاده خواهد کرد.

ADD NEW STUDENTS

First Name	<input type="text" value="New"/>
Last Name	<input type="text" value="Student"/>
Enrollment Date	<input type="text" value="09/10/2010"/>
Insert Cancel	

صفحه *Students.aspx* را اجرا کنید، مشاهده میکنید دانشجوی جدید در لیست نمایش داده می شود.

STUDENT LIST

	ID	Name	Enrollment Date	Number of Courses
Edit Delete	26	Rogers, Cody	9/1/2002	2
Edit Delete	28	White, Anthony	9/1/2001	2
Edit Delete	29	Griffin, Rachel	9/1/2004	1
Edit Delete	30	Shan, Alicia	9/1/2003	2
Edit Delete	33	Gao, Erica	1/30/2003	0
Edit Delete	35	Smith, John	1/1/2011	0
Edit Delete	38	Student, New	9/10/2010	0

[1](#) [2](#) [3](#)

برای بررسی اینکه تابع **Update** درست کار میکند، **Name** را تغییر دهید، و سپس برای بررسی تابع **Delete**، دانشجو را حذف کنید.

STUDENT LIST

	ID	Name	Enrollment Date	Number of Courses	
Edit Delete	26	Rogers, Cody	9/1/2002	2	
Edit Delete	28	White, Anthony	9/1/2001	2	
Edit Delete	29	Griffin, Rachel	9/1/2004	1	
Edit Delete	30	Shan, Alicia	9/1/2003	2	
Edit Delete	33	Gao, Erica	1/30/2003	0	
Edit Delete	35	Smith, John	1/1/2011	0	
Update Cancel	38	Student	<input type="text" value="ToBeDeleted"/>	9/10/2010	0

[1](#) [2](#) [3](#)

استفاده از رویه های ذخیره شده انتخاب^۱

Entity Framework به طور خودکار رویه های ذخیره شده را اجرا نمی کند (مانند GetCourses)، و شما نمی توانید از آنها همراه کنترل EntityDataSource استفاده کنید. برای استفاده از آنها، در قسمت کد برنامه آنها را فراخوانی کنید.

فایل *InstructorsCourses.aspx.cs* را باز کنید. متد **PopulateDropDownLists** از یک کوئری LINQ-to-Entities برای برگرداندن همه موجودیت های Course استفاده می کند تا بتواند در لیست گردش کرده و تعیین کند که کدام یک از آنها به یک استاد اختصاص داده شده است و کدام یک اختصاص داده نشده است:

```
var allCourses =(from c in context.Course
select c).ToList();
```

این کد را با کد زیر جایگزین کنید:

```
var allCourses = context.GetCourses();
```

اکنون این صفحه از متد GetCourses برای برگرداندن لیستی از همه Course ها، استفاده می کند. صفحه را به منظور بررسی اینکه مانند گذشته کار می کند، اجرا کنید.

(خصوصیت های Navigation موجودیت هایی که توسط یک رویه ذخیره شده برگردانده می شوند ممکن است به طور خودکار با داده های مرتبط به این موجودیت ها پر نشوند، این موضوع بستگی به تنظیمات پیش فرضObjectContext دارد. برای دیدن اطلاعات بیشتر به [Loading Related Objects](#) در کتابخانه MSDN مایکروسافت مراجعه نمائید)

در بخش بعد چگونگی استفاده از عاملیت داده های پویا^۲ را به منظور ساده تر کردن برنامه نویسی و تست قوانین^۳ data formatting (شکل دهی داده ها) و validation (اعتبارسنجی)، خواهید آموخت. به جای تعیین کردن نقشها در هر صفحه مانند

¹ select

² Dynamic Data functionality

³ Rule

رشته های قالب داده، یا اینکه یک فیلد لازم هست یا نه، می توانید در فراداده (metadata) مدل داده نقش هایی را تعیین کنید که به طور خودکار روی هر صفحه اعمال شوند.

استفاده از Dynamic Data Functionality برای شکل دهی و اعتبارسنجی داده ها

در بخش قبل رویه های ذخیره شده را اجرا کردید. در این بخش به شما نشان داده خواهد شد که عاملیت داده های پویا چگونه مزایای زیر را تامین می کند:

- فیلدها بر اساس نوع داده خود، به طور خودکار شکل دهی می شوند.
- فیلدها بر اساس نوع داده خود، اعتبارسنجی می شوند.
- می توانید فراداده ها را به منظور سفارشی سازی اعمال شکل دهی^۱ و اعتبارسنجی^۲ به مدل داده اضافه کنید. وقتی این کار را انجام می دهید، می توانید نقش های شکل دهی و اعتبارسنجی را فقط در یک محل اضافه کنید، و آنها به طور خودکار هر جایی که شما به فیلدهایی استفاده کننده از کنترل های عاملیت داده پویا دسترسی پیدا کردید، اعمال شود.

برای اینکه ببینید چگونه کار می کند، کنترل هایی که برای نمایش و ویرایش فیلدها استفاده می کنید، را در صفحه *Students.aspx* تغییر خواهید داد، و فراداده شکل دهی و اعتبارسنجی را به فیلدهای نام و تاریخ نوع موجودیت *Student* اضافه خواهید کرد.

¹ formatting

² Validation

STUDENT LIST

	Name	Enrollment Date	Number of Courses
Edit Delete	Alexander, Carson	9/1/2055	3
Edit Delete	Alonso, Meredith	9/1/2002	1
Edit Delete	Anand, Arturo	9/1/2003	2
Edit Delete	Barzdukas, Gytis	9/1/2005	2
Edit Delete	Browning, Meredith	9/1/2000	2
Edit Delete	Bryant, Carson	9/1/2001	1
Edit Delete	Carlson, Robyn	9/1/2005	1
Edit Delete	Gao, Erica	1/30/2003	0
Edit Delete	Griffin, Rachel	9/1/2004	1
Edit Delete	Holt, Roger	9/1/2004	1

FIND STUDENTS BY NAME

Enter any part of the name

Name	Enrollment Date
Barzdukas, Gytis	9/1/2005

استفاده از کنترل های DynamicControl و DynamicField

صفحه *Students.aspx* را باز کرده و در کنترل *StudentsGridView* فیلدهای الگوی **Name** و **EnrollmentDate** را با کد ساختیافته زیر جایگزین کنید:

```
<asp:TemplateField HeaderText="Name" SortExpression="LastName">
  <EditItemTemplate>
    <asp:DynamicControl ID="LastNameTextBox" runat="server" DataField="LastName"
      Mode="Edit"/>
    <asp:DynamicControl ID="FirstNameTextBox" runat="server" DataField="FirstMidName"
      Mode="Edit"/>
  </EditItemTemplate>
  <ItemTemplate>
    <asp:DynamicControl ID="LastNameLabel" runat="server" DataField="LastName"
      Mode="ReadOnly"/>,
    <asp:DynamicControl ID="FirstNameLabel" runat="server" DataField="FirstMidName"
      Mode="ReadOnly"/>
  </ItemTemplate>
</asp:TemplateField>
<asp:DynamicField DataField="EnrollmentDate" HeaderText="EnrollmentDate"
  SortExpression="EnrollmentDate"/>
```

¹ TemplateField

این کد ساخته‌شده به جای کنترل‌های `TextBox` و `Label`، از کنترل `DynamicControl`، در فیلد الگوی نام دانشجو، و از کنترل `DynamicField` برای تاریخ ثبت نام استفاده می‌کند. هیچ رشته قالبی^۱ تعیین نشده است.

یک کنترل `ValidationSummary`، بعد از کنترل `StudentsGridView` اضافه کنید.

```
<asp:ValidationSummary ID="StudentsValidationSummary" runat="server"
ShowSummary="true"
DisplayMode="BulletList" Style="color:Red"/>
```

در کنترل `SearchGridView` همانند آنچه در مورد کنترل `StudentsGridView` انجام دادید، کد مربوط به ستون‌های `Name` و `Enrollment` را جایگزین کنید. اما عنصر `EditItemTemplate` را حذف کنید. عنصر `Columns` کنترل `SearchGridView` اکنون حاوی کد زیر است:

```
<asp:TemplateField HeaderText="Name" SortExpression="LastName">
<ItemTemplate>
<asp:DynamicControl ID="LastNameLabel" runat="server" DataField="LastName"
Mode="ReadOnly"/>,
<asp:DynamicControl ID="FirstNameLabel" runat="server" DataField="FirstMidName"
Mode="ReadOnly"/>
</ItemTemplate>
</asp:TemplateField>
<asp:DynamicField DataField="EnrollmentDate" HeaderText="EnrollmentDate"
SortExpression="EnrollmentDate"/>
```

`Students.aspx.cs` را باز کرده و دستور `using` زیر را اضافه کنید:

```
using ContosoUniversity.DAL;
```

یک گرداننده برای رخداد `Init` این صفحه اضافه کنید:

```
protected void Page_Init(object sender, EventArgs e)
{
    StudentsGridView.EnableDynamicData(typeof(Student));
    SearchGridView.EnableDynamicData(typeof(Student));
}
```

این کد تعیین می کند که Dynamic Data، قالب بندی و اعتبارسنجی در کنترل های data-bound فیلدهای موجودیت Student را انجام می دهد. اگر هنگام اجرای صفحه، پیغام خطایی مانند مثال زیر را دریافت کردید، عموماً به معنی این است که فراموش کرده اید که متد EnableDynamicData را در Page_Init فراخوانی کنید:

```
Could not determine a MetaTable. A MetaTable could not be determined for the data source 'StudentsEntityDataSource' and one could not be inferred from the request URL.
```

صفحه را اجرا کنید.

STUDENT LIST

	Name	Enrollment Date	Number of Courses
Edit Delete	Alexander, Carson	9/1/2055 12:00:00 AM	3
Edit Delete	Alonso, Meredith	9/1/2002 12:00:00 AM	1
Edit Delete	Anand, Arturo	9/1/2003 12:00:00 AM	2
Edit Delete	Barzdukas, Gytis	9/1/2005 12:00:00 AM	2
Edit Delete	Browning, Meredith	9/1/2000 12:00:00 AM	2
Edit Delete	Bryant, Carson	9/1/2001 12:00:00 AM	1
Edit Delete	Carlson, Robyn	9/1/2005 12:00:00 AM	1
Edit Delete	Gao, Erica	1/30/2003 12:00:00 AM	0
Edit Delete	Griffin, Rachel	9/1/2004 12:00:00 AM	1
Edit Delete	Holt, Roger	9/1/2004 12:00:00 AM	1

[1](#) [2](#) [3](#)

FIND STUDENTS BY NAME

Enter any part of the name

Name	Enrollment Date
Barzdukas, Gytis	9/1/2005 12:00:00 AM
Justice, Peggy	9/1/2001 12:00:00 AM
Li, Yan	9/1/2002 12:00:00 AM
Norman, Laura	9/1/2003 12:00:00 AM
Olivotto, Nino	9/1/2005 12:00:00 AM

در ستون **Enrollment Date**، زمان همراه با تاریخ نمایش داده شده است. چون نوع این خصوصیت **DateTime** است. بعداً آن را تغییر خواهیم داد.

فعلاً، توجه کنید که **Dynamic Data** به طور خودکار اعتبارسنجی ابتدایی داده ها را انجام می دهد. برای مثال، روی **Edit** کلیک کنید، **date field** را پاک کنید، روی **Update** کلیک کنید، و مشاهده می کنید که **Dynamic Data** به طور خودکار این فیلد را یک **required field** قرار می دهد، چون مقدار آن در مدل داده **nullable** نیست. در این هنگام بعد از فیلد مورد نظر یک ستاره و در کنترل **ValidationSummary** یک پیغام خطا نمایش می دهد.

STUDENT LIST

	Name	Enrollment Date
Update Cancel	Alexander Carson	*
Edit Delete	Alonso, Meredith	9/1/2002
Edit Delete	Anand, Arturo	9/1/2003
Edit Delete	Barzdukas, Gytis	9/1/2005
Edit Delete	Browning, Meredith	9/1/2000
Edit Delete	Bryant, Carson	9/1/2001
Edit Delete	Carlson, Robyn	9/1/2005
Edit Delete	Gao, Erica	1/30/2003
Edit Delete	Griffin, Rachel	9/1/2004
Edit Delete	Holt, Roger	9/1/2004

1 2 3

- The EnrollmentDate field is required.

می توانید کنترل ValidationSummary را حذف کنید. چون با نگه داشتن موس روی ستاره نیز، می توانید پیغام خطا را مشاهده کنید:

Enrollment Date	Number of Courses
	* 3
9/1/2002	1
9/1/2003	2

The EnrollmentDate field is required.

Dynamic Data همچنین بررسی می کند که تاریخ وارد شده در **Enrollment Date** یک تاریخ معتبر است یا خیر.

STUDENT LIST

	Name	Enrollment Date
Update Cancel	Alexander Carson	1/32/2010
Edit Delete	Alonso, Meredith	9/1/2002 12:00:00 AM
Edit Delete	Anand, Arturo	9/1/2003 12:00:00 AM
Edit Delete	Barzdukas, Gytis	9/1/2005 12:00:00 AM
Edit Delete	Browning, Meredith	9/1/2000 12:00:00 AM
Edit Delete	Bryant, Carson	9/1/2001 12:00:00 AM
Edit Delete	Carlson, Robyn	9/1/2005 12:00:00 AM
Edit Delete	Gao, Erica	1/30/2003 12:00:00 AM
Edit Delete	Griffin, Rachel	9/1/2004 12:00:00 AM
Edit Delete	Holt, Roger	9/1/2004 12:00:00 AM

1 2 3

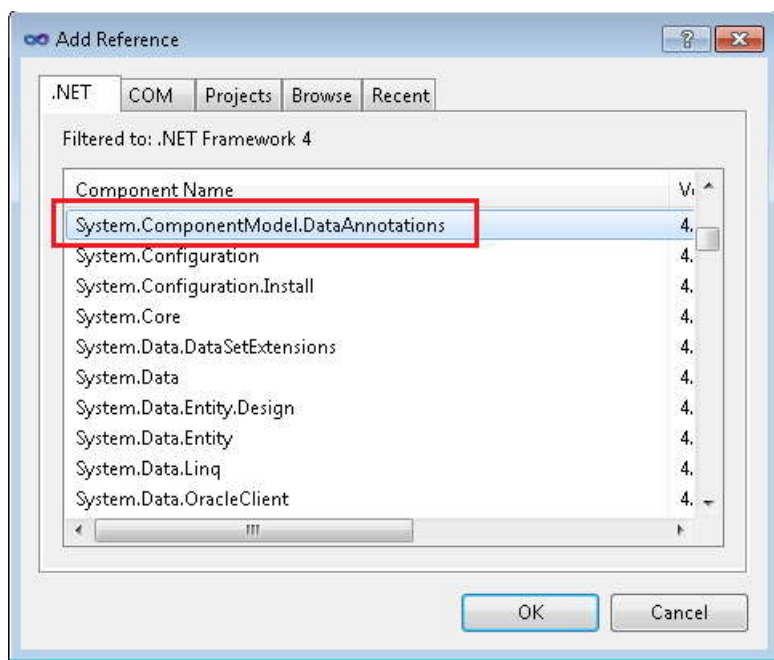
- The value is not valid.

همان طور که مشاهده می کنید، این یک پیغام خطای کلی است. در بخش بعد خواهید دید چگونه پیغام های خطا و همچنین قوانین قالب بندی و اعتبارسنجی را سفارشی کنید.

اضافه کردن فراداده به مدل داده

به طور نمونه، شما می خواهید عملکردی که توسط Dynamic Data ارائه می شود را سفارشی کنید. برای مثال، ممکن است بخواهید چگونگی نمایش داده ها و محتوای پیغام خطا را تغییر دهید. یا همچنین بخواهید قوانین اعتبارسنجی داده ها را به منظور افزایش کارایی نسبت به آنچه Dynamic Data بر اساس انواع داده ارائه می دهد، سفارشی کنید. برای انجام این کار، کلاس های partial متناظر با انواع موجودیت ها را ایجاد کنید.

در **Solution Explorer**، روی پروژه **ContosoUniversity** راست کلیک کنید، **Add Reference** را انتخاب کرده و یک ارجاع به **System.ComponentModel.DataAnnotations** اضافه کنید.



در پوشه DAL، یک فایل کلاس جدید ایجاد کرده و نام آن را *Student.cs* قرار دهید. و کد الگوی درون آن را با کد زیر جایگزین نمایید:

```
Using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
namespace ContosoUniversity.DAL
{
    [MetadataType(typeof(StudentMetadata))]
    public partial class Student
    {
    }
    public class StudentMetadata
    {
        [DisplayFormat(DataFormatString="{0:d}", ApplyFormatInEditMode=true)]
        public DateTime EnrollmentDate{get;set;}
        [StringLength(25, ErrorMessage="First name must be 25 characters or less in length.")]
    }
}
```

```

[Required(ErrorMessage="First name is required.")]
public String FirstMidName{get;set;}
[StringLength(25,ErrorMessage="Last name must be 25 characters or less in
length.")]
[Required(ErrorMessage="Last name is required.")]
public String LastName{get;set;}
}
}

```

این کد یک کلاس `partial` برای موجودیت `Student` ایجاد می کند. صفت `MetadataType` در مورد این کلاس `Partial` اعمال می شود تا تعیین کند که شما از این کلاس برای مشخص کردن فراداده استفاده می کنید. کلاس فراداده می تواند هر نامی داشته باشد، اما استفاده از نام موجودیت به علاوه "Metadata" یک شیوه معمول است.

صفت هایی که به خصوصیت های کلاس فراداده اعمال می شوند، مشخص کننده قالب بندی، اعتبارسنجی، نقش ها، و پیغام های خطا هستند. صفت هایی که اینجا نشان داده شده اند، نتایج زیر را خواهند داشت:

- **EnrollmentDate** به صورت تاریخ نمایش داده می شود. (بدون ساعت)
- هر دو فیلد نام باید ۲۵ کاراکتر یا کمتر باشند، و یک پیغام خطای سفارشی در نظر گرفته شده.
- هر دو فیلد نام ضروری هستند، و یک پیغام خطای سفارشی در نظر گرفته شده.

صفحه `Students.aspx` را دوباره اجرا کرده، مشاهده می کنید که اکنون تاریخ ها بدون ساعت هستند:

STUDENT LIST			
	Name	Enrollment Date	Number of Courses
Edit Delete	Alexander, Carson	9/1/2055	3
Edit Delete	Alonso, Meredith	9/1/2002	1
Edit Delete	Anand, Arturo	9/1/2003	2

یک سطر را `Edit` کرده و مقادیر فیلدهای نام را پاک کنید. به محض ترک این فیلد و قبل از کلیک روی `Update` یک ستاره که مشخص کننده خطا در مورد آن فیلد است ظاهر می شود. هنگامی که روی `Update` کلیک می کنید متن خطایی که شما تعیین کرده اید نمایش داده می شود.

STUDENT LIST

		Name	
Update	Cancel	<input type="text" value="*"/>	<input type="text" value="*"/>
Edit	Delete	Alonso, Meredith	
Edit	Delete	Anand, Arturo	
Edit	Delete	Barzdukas, Gytis	
Edit	Delete	Browning, Meredith	
Edit	Delete	Bryant, Carson	
Edit	Delete	Carlson, Robyn	
Edit	Delete	Gao, Erica	
Edit	Delete	Griffin, Rachel	
Edit	Delete	Holt, Roger	
1 2 3			

- Last name is required.
- First name is required.

یک نام به طول بیش از ۲۵ کاراکتر وارد کرده و روی **Update** کلیک کنید. متن خطایی را که شما تعیین کرده اید در این صفحه نمایش داده می شود.

STUDENT LIST

		Name	
Update	Cancel	<input type="text" value="Alexander more than 25 c"/>	<input type="text" value="Carson more than 25 char"/>
Edit	Delete	Alonso, Meredith	
Edit	Delete	Anand, Arturo	
Edit	Delete	Barzdukas, Gytis	
Edit	Delete	Browning, Meredith	
Edit	Delete	Bryant, Carson	
Edit	Delete	Carlson, Robyn	
Edit	Delete	Gao, Erica	
Edit	Delete	Griffin, Rachel	
Edit	Delete	Holt, Roger	
1 2 3			

- Last name must be 25 characters or less in length.
- First name must be 25 characters or less in length.

حال که قوانین قالب بندی و اعتبارسنجی را در فراداده مدل داده تنظیم کرده اید، این قوانین به هر صفحه ای که اجازه تغییر فیلدهای نام و تاریخ ثبت نام دانشجو را می دهد، اعمال خواهد شد. مادامی که از کنترل های `DynamicControl` و `DynamicField` استفاده می کنید. این کار، میزان کدهای اضافی که مجبور هستید بنویسید را کاهش می دهد و برنامه نویسی و

تست را آسان تر میکنند، و باعث اطمینان از این موضوع می شود که قالب بندی و اعتبارسنجی در سرتاسر برنامه دارای استحکام و سازگاری است.

کنترل `ObjectDataSource`

کنترل `EntityDataSource` به شما این امکان را می دهد تا یک برنامه را به سرعت ایجاد کنید، اما به طور نمونه شما را ملزم می کند که مقدار زیادی از منطق تجاری¹ و منطق دسترسی به داده² را در صفحات `aspx` نگه داری کنید. اگر پیش بینی می کنید برنامه شما پیچیده تر شود و نیاز به نگه داری مداوم داشته باشد، می توانید به منظور ایجاد ساختار یک برنامه چند لایه³ که قابلیت نگه داری بیشتری دارد، بیشتر زمان توسعه را در قسمت جلویی قرار دهید. برای پیاده سازی این ساختار، لایه ارائه⁴ را از لایه منطق تجاری (BLL) و لایه دسترسی به داده (DAL) جدا کنید. یک راه پیاده سازی این ساختار استفاده از کنترل `ObjectDataSource` به جای کنترل `EntityDataSource` است. زمانی که از کنترل `ObjectDataSource` استفاده می کنید، کد دسترسی به داده را پیاده سازی کرده و سپس آن را در صفحات `aspx` ای که از کنترل `ObjectDataSource` استفاده می کنند، فراخوانی کنید. این به شما اجازه می دهد که مزایای یک روش *n-tier* (چند لایه) و استفاده از کنترل `Web Form` را برای دسترسی به داده ها ترکیب کنید.

همچنین کنترل `ObjectDataSource` نسبت به روش های دیگر قابلیت انعطاف بیشتری برای شما فراهم می کند. چون خودتان کد دسترسی به داده ها را می نویسید، انجام اعمالی بیش از، خواندن، درج، بروزرسانی یا حذف یک نوع موجودیت خاص (وظایفی هستند که کنترل `EntityDataSource` برای اجرا طراحی می کند) آسان تر است. به طور مثال، می توانید عملیات ثبت وقایع⁵، هر زمانی که موجودیتی بروزرسانی می شود، آرشیو داده ها هر وقت یک موجودیت حذف می شود، یا چک کردن و بروزرسانی داده های مرتبط در صورت نیاز، هنگام درج یک سطر با مقدار کلید خارجی، را انجام دهید.

¹ Business logic

² data-access

³ n-tier

⁴ presentation layer

⁵ logging

کلاس های منطق تجاری و مخزن

یک کنترل `ObjectDataSource` با فراخوانی کلاسی که شما ایجاد کرده اید کار می کند. این کلاس حاوی متدهایی است که داده ها را بر می گردانند یا بروزرسانی می کنند، و شما نام این متدها را در کد ساختیافته کنترل `ObjectDataSource` ذکر می کنید. در طول عملیات `rendering` (تهیه صفحه) یا `postback` (ارسال صفحه از سمت کاربر به سرور)، کنترل `ObjectDataSource` متدهایی را که شما مشخص کرده اید را صدا میزند.

در کنار عمل های اصلی `CRUD`¹، کلاسی که شما به منظور استفاده از کنترل `ObjectDataSource` ایجاد کرده اید، ممکن است هنگامی که `ObjectDataSource` عملیات خواندن یا بروزرسانی را انجام می دهد، نیاز به اجرای منطق تجاری داشته باشد. برای مثال، هنگامی که یک دانشکده را بروزرسانی می کنید، ممکن است نیاز باشد که بررسی کنید، هیچ دانشکده دیگری دارای مدیر این دانشکده نیست، چون یک شخص نمی تواند مدیر بیش از یک دانشکده باشد.

در بعضی از مستندات `ObjectDataSource` مانند [ObjectDataSource Class overview](#)، کنترل `ObjectDataSource` کلاسی را صدا میزند که به یک شی تجاری² که حاوی منطق تجاری و منطق دسترسی به داده است ارجاع شده است. در این بخش کلاس های جداگانه ای برای منطق تجاری و منطق دسترسی به داده ایجاد خواهید کرد. کلاسی که منطق دسترسی به داده را در بر می گیرد انبار³ یا مخزن نامیده می شود. کلاس منطق تجاری حاوی هر دوی متدهای منطق تجاری و متدهای منطق دسترسی به داده است، اما کلاس دسترسی به داده، کلاس مخزن را به منظور انجام وظایف⁴ دسترسی به داده فراخوانی می کند.

همچنین یک لایه انتزاعی بین `BLL` و `DAL` که تست واحد خودکار `BLL` را ساده می کند، ایجاد خواهید کرد. این لایه انتزاعی به وسیله ایجاد یک واسط و استفاده از آن هنگامی که مخزن را در کلاس منطق تجاری نمونه سازی می کنید، پیاده سازی می شود. این لایه این امکان را به شما می دهد که کلاس منطق تجاری را با یک ارجاع به هر شئی که واسط مخزن را پیاده سازی می کند تهیه کنید. برای عملیات معمول، یک شی مخزن که با `Entity Framework` کار می کند، تهیه کنید. برای تست، یک شی مخزن فراهم کنید. که با داده های ذخیره شده به شیوه ای که شما بتوانید به راحتی آنها را دستکاری کنید، کار کند، مانند متغیرهای کلاس که به عنوان مجموعه⁵ تعریف شده است.

تصویر زیر تفاوت بین کلاس منطق تجاری که شامل منطق دسترسی به داده، و بدون مخزن است و کلاسی که دارای مخزن است را نشان می دهد.

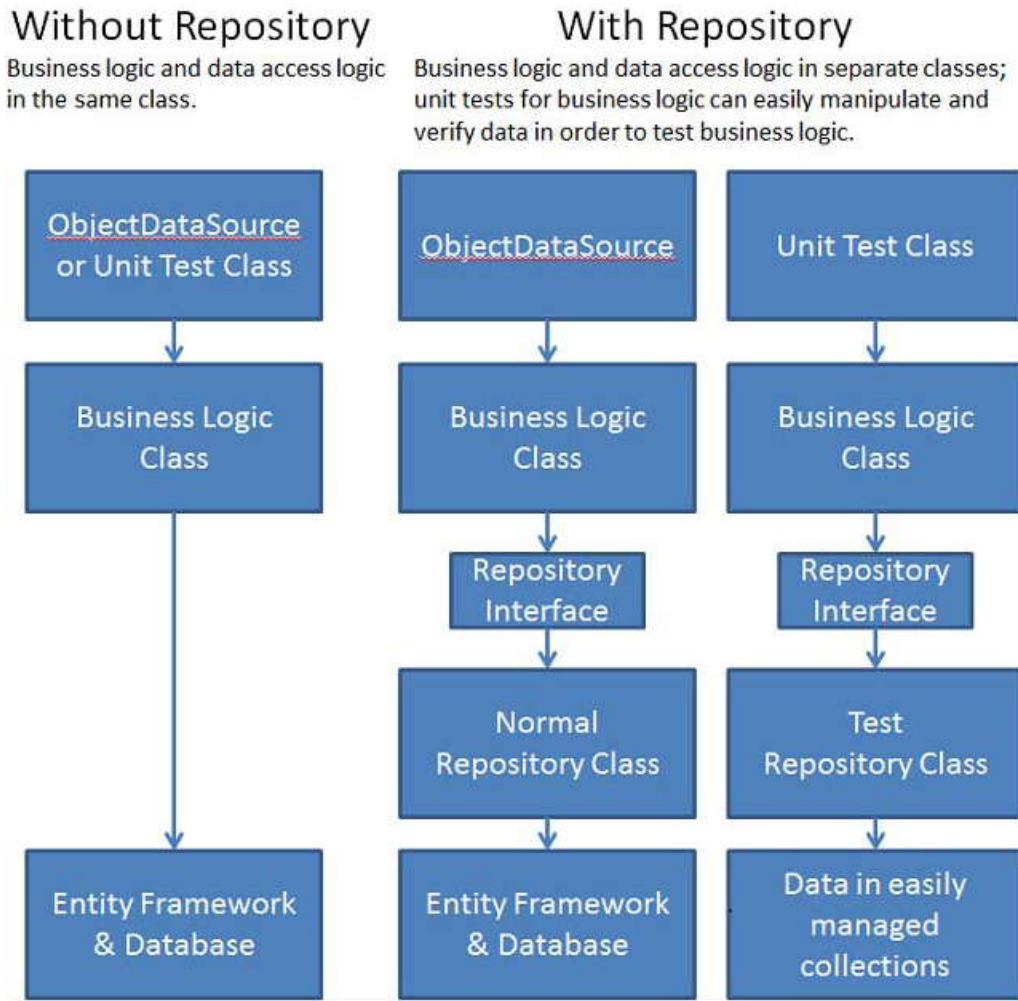
¹ Create, read, update, delete

² business object

³ repository

⁴ task

⁵ Collection



کار را با ایجاد صفحاتی ایجاد خواهید کرد که درون آنها کنترل `ObjectDataSource` به طور مستقیم به یک مخزن مقید شده است، زیرا فقط وظایف اصلی دسترسی به داده را انجام می دهد. در بخش بعد یک کلاس منطق تجاری همراه منطق اعتبارسنجی ایجاد خواهید کرد و کنترل `ObjectDataSource` را به جای کلاس مخزن به این کلاس مقید خواهید کرد. همچنین واحدهای تستی برای منطق اعتبارسنجی ایجاد خواهید کرد. در بخش های دیگر عملکردهای مرتب سازی¹ و فیلتر کردن² را اضافه خواهید کرد.

¹ sorting
² filtering

صفحاتی که در این بخش ایجاد می کنید، با مجموعه موجودیت Department مدل داده ای که در بخش های قبل ایجاد کرده اید کار می کند.

DEPARTMENTS				
	Name	Budget	Start Date	Administrator
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy

ADD DEPARTMENTS	
Name	<input type="text"/>
Budget	<input type="text"/>
Start Date	<input type="text"/>
Administrator	Abercrombie, Kim <input type="button" value="v"/>
Insert Cancel	

بروزرسانی دیتابیس و مدل داده

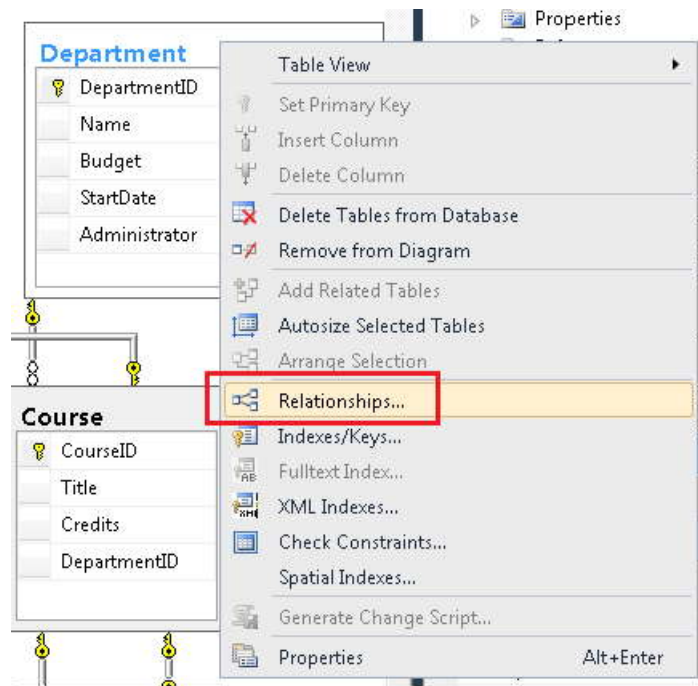
این بخش را با اعمال دو تغییر در دیتابیس شروع خواهید کرد، که نیاز به همسان سازی تغییرات در مدل داده ای که قبلا ایجاد کرده اید دارد. در یکی از بخش های پیشین، جهت همگام سازی مدل داده با دیتابیس بعد از تغییر دیتابیس، تغییراتی را در محیط طراحی مدل داده به صورت دستی ایجاد کردید. این بخش از ابزار **Update Model From Database** محیط طراحی جهت بروزرسانی خودکار مدل داده استفاده خواهید کرد.

اضافه کردن رابطه به دیتابیس

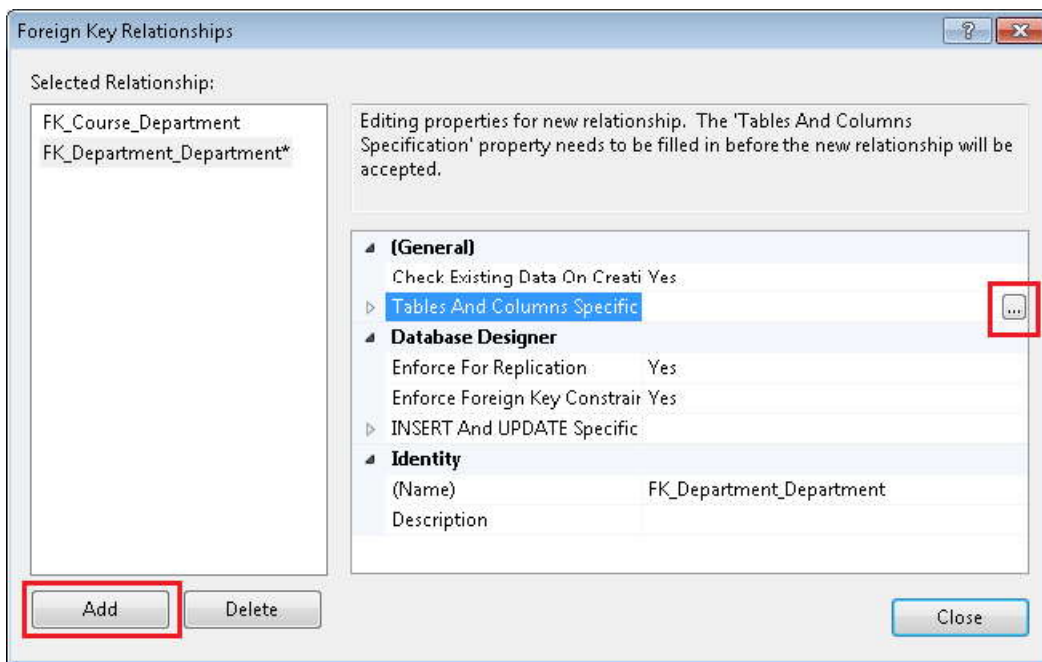
در Visual Studio، برنامه وب Contoso University را که در بخش های قبل ایجاد کرده اید باز کرده، و سپس دیاگرام دیتابیس SchoolDiagram را باز کنید.

اگر به جدول Department در دیاگرام دیتابیس نگاه کنید، مشاهده خواهید کرد که یک ستون Administrator دارد. این ستون کلید خارجی به جدول Person است، اما هیچ رابطه کلید خارجی در دیتابیس تعریف نشده است. شما نیاز دارید این رابطه را ایجاد کرده و مدل داده را بروزرسانی کنید تا Entity Framework بتواند به طور خودکار از این رابطه استفاده کند.

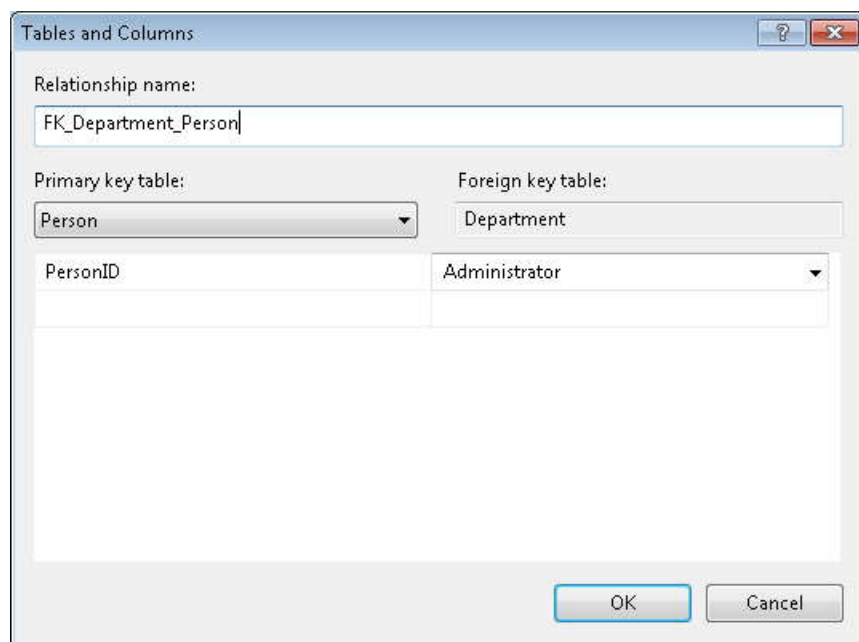
در دیاگرام دیتابیس، روی جدول Department راست کلیک کرده و Relationships را انتخاب کنید.



در کادر Foreign Key Relationships روی Add و سپس روی دکمه مقابل Tables and Columns Specification کلیک کنید.



در کادر محاوره ای **Tables and Columns**، جدول و فیلد کلید اصلی (primary key) را برابر **Person** و **PersonID** قرار دهید و جدول و فیلد کلید خارجی را **Department** و **Administrator** قرار دهید. (وقتی این کار را انجام دادید، نام رابطه از **FK_Department_Department** به **FK_Department_Person** تغییر خواهد کرد.)



در کادر **Tables and Columns** روی **OK** کلیک کرده و در کادر **Foreign Key Relationships** روی **Close** کلیک و تغییرات را ذخیره کنید. اگر از شما سوال شد که آیا می خواهید جدول های **Person** و **Department** را ذخیره کنید، روی **Yes** کلیک کنید.

توجه کنید: اگر قبلا سطرهای جدول **Person** را که با داده های ستون **Administrator** برابر اند را حذف کرده اید، قادر به ذخیره کردن تغییرات نخواهید بود. در این صورت از ویرایشگر جدول در **Server Explorer** استفاده کنید تا مطمئن شوید مقدار **Administrator** در هر سطر **Department** حاوی **ID** رکوردی است که واقعا در جدول **Person** وجود دارد.

بعد از اینکه تغییرات را ذخیره کردید، نمی توانید سطری از جدول **Person** را در صورتی که شخص، مدیر یک دانشکده باشد حذف کنید. در یک نرم افزار تولیدی، ممکن است شما هنگامی که یک محدودیت دیتابیس از عملیات حذف جلوگیری می کند، پیغام خطای معینی را ارائه دهید، یا حذف آبخاری را تعیین کنید.¹ برای دیدن یک مثال از چگونگی مشخص کردن حذف آبخاری، به [The Entity Framework and ASP.NET – Getting Started Part 2](#) مراجعه کنید.

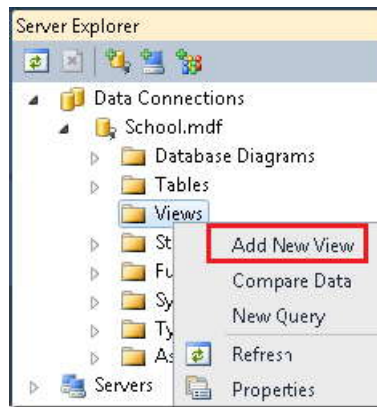
اضافه کردن دید² به دیتابیس

در صفحه **Departments.aspx** جدیدی که می خواهید ایجاد کنید، شما نیاز به یک **drop-down list** برای اساتید دارید، همراه نام ها با قالب "last, first" تا کاربران بتوانند مدیران دانشکده ها را انتخاب کنند. برای اینکه راحت تر این کار انجام شود، در دیتابیس یک دید ایجاد می کنید. دید تنها شامل داده هایی است که مورد نیاز لیست کشویی هستند: نام کامل (با قالب بندی مناسب) و کلید رکورد.

در **Server Explorer**، گزینه **School.mdf** را گسترش داده، روی پوشه **Views** راست کلیک و **Add New View** را انتخاب کنید.

¹ cascading

² View



هنگامی که کادر محاوره ای **Add Table** ظاهر می شود روی **Close** کلیک کنید، و دستورات SQL زیر را در کادر¹ SQL بچسبانید.

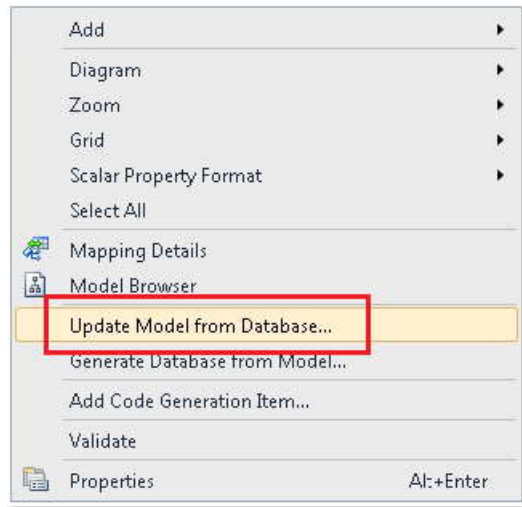
```
SELECT LastName+', '+FirstName AS FullName, PersonID
FROM dbo.Person
WHERE (HireDate IS NOT NULL)
```

دید را با نام **vinstructorName** ذخیره کنید.

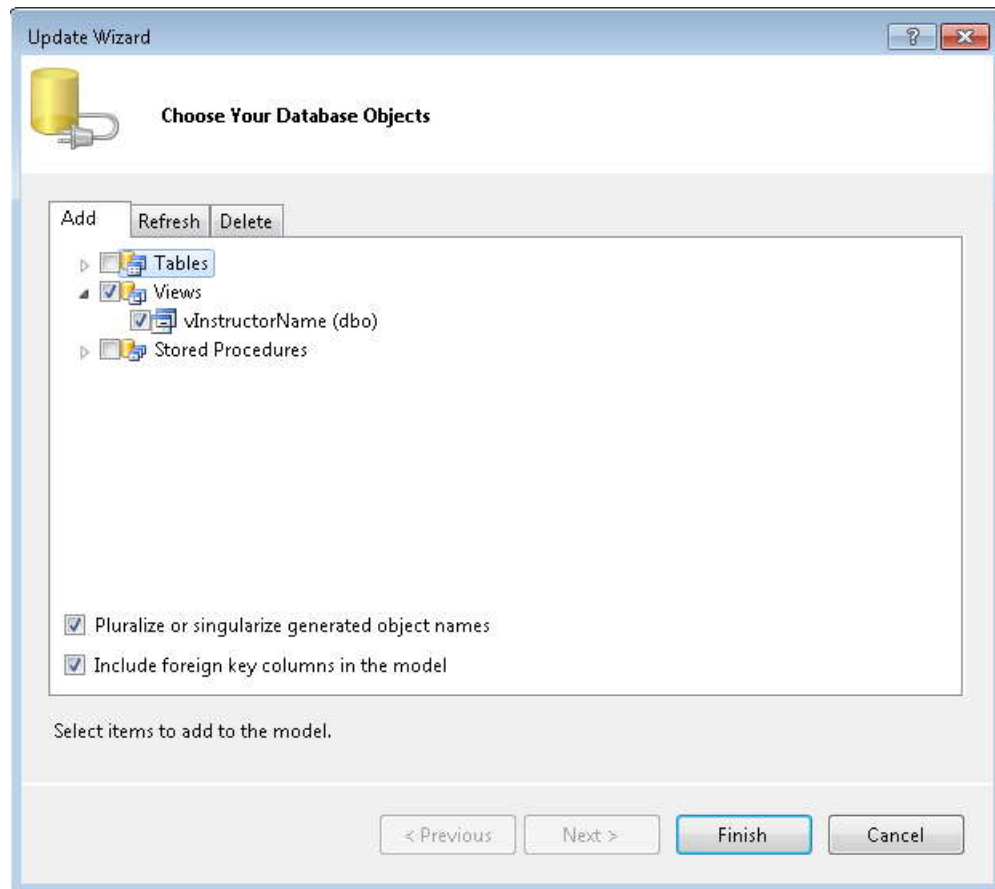
به روزرسانی مدل داده

در پوشه **DAL** فایل **SchoolModel.edmx** را باز کرده، روی قسمت طراحی راست کلیک کرده و **Update Model** را انتخاب کنید. **From Database** را انتخاب کنید.

¹ SQL pane

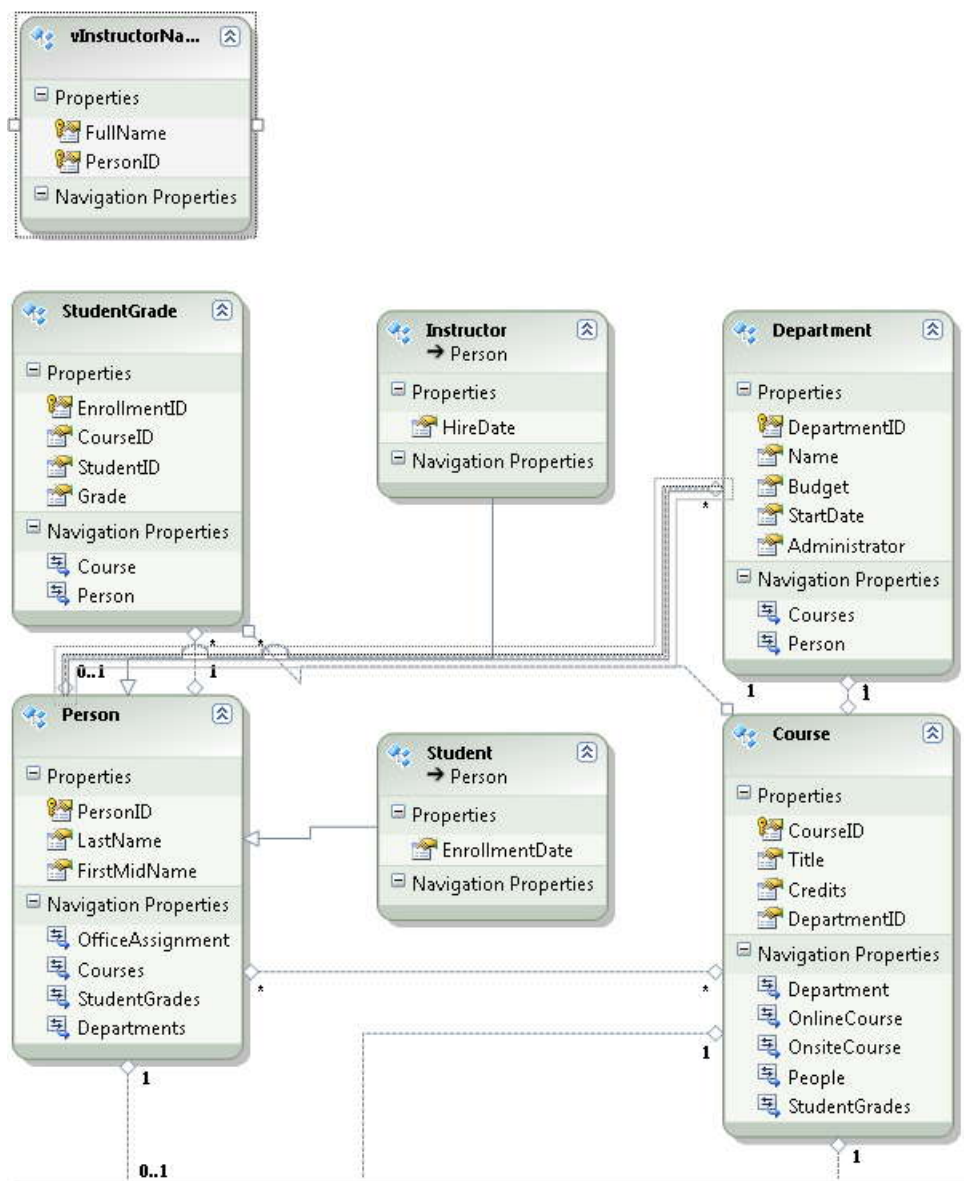


در کادر محاوره ای **Choose Your Database Objects**، برگه **Add** را انتخاب کرده و دیدی را که اکنون ایجاد کرده اید، انتخاب کنید.



روی **Finish** کلیک کنید.

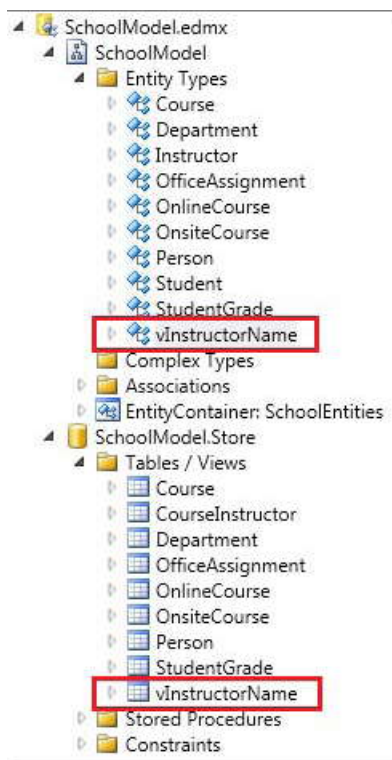
در محیط طراحی، مشاهده می کنید که این ابزار (Designer) یک موجودیت **vInstructorName** و یک وابستگی بین موجودیت های **Person** و **Department** اضافه کرده است.



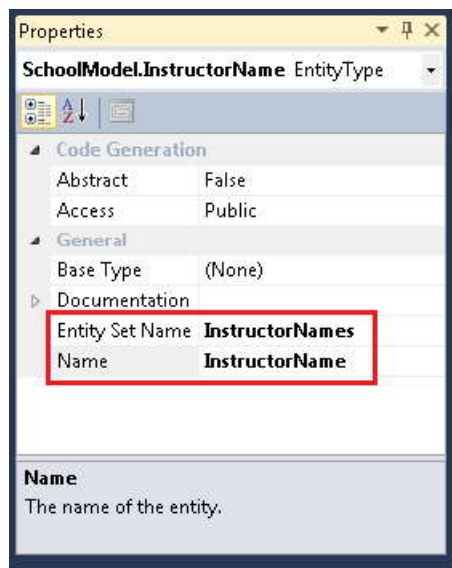
توجه: در پنجره **Output** و **Error List** ممکن است پیام خطاری را ببینید که شما را نسبت به کلید اصلی ایجاد شده برای **vInstructorName** را آگاه می کند. (این عملی مورد انتظار است.)

زمانی که در کد برنامه نویسی به یک موجودیت `vInstructorName` رجوع می کنید نیاز نیست از قرارداد پیشوند گذاری دیتابیس، با حرف کوچک "v" استفاده کنید. بنابراین نام این مجموعه موجودیت را در مدل داده تغییر خواهیم داد.

Model Browser را باز کنید. `vInstructorName` به عنوان یک نوع موجودیت و یک دید، لیست شده است.



در زیرمجموعه **SchoolModel** (نه **SchoolModel.Store**) روی `vInstructorName` راست کلیک کرده و **Properties** را انتخاب کنید. در پنجره **Properties** خصوصیت **Name** را به "InstructorName" و خصوصیت **Entity Set Name** را به "InstructorNames" تغییر دهید.



مدل داده را ذخیره کرده و ببندید، و سپس پروژه را دوباره بسازید (rebuild).

استفاده از کلاس مخزن و کنترل ObjectDataSource

یک فایل کلاس در پوشه DAL ایجاد کنید، نام آن را *SchoolRepository.cs* قرار داده، و کد زیر را جایگزین کد موجود

کنید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using ContosoUniversity.DAL;
namespace ContosoUniversity.DAL
{
    public class SchoolRepository:IDisposable
    {
        private SchoolEntities context =new SchoolEntities();
        public IEnumerable<Department>GetDepartments()
        {
            return context.Department.Include("Person").ToList();
        }
        private bool disposedValue =false;
        protected virtual void Dispose(bool disposing)
        {
            if(!this.disposedValue)
            {
                if(disposing)
                {
                    context.Dispose();
                }
            }
            this.disposedValue =true;
        }
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }
    }
}
```

این کد یک متد *GetDepartments* ارائه می دهد، که تمام موجودیت های مجموعه موجودیت *Department* را بر می گرداند.. به خاطر اینکه می دانید در آینده برای هر سطر برگشتی به خصوصیت *Person navigation* دسترسی خواهید داشت، *eager loading* را با استفاده از متد *Include* تعیین کنید. همچنین این کلاس برای اطمینان از قطع شد اتصال دیتابیس در زمان از بین رفتن شیء، واسط *IDisposable* را پیاده سازی می کند.

توجه: یک شیوه معمول، ایجاد یک کلاس مخزن برای هر نوع موجودیت است. در این بخش، یک کلاس مخزن برای چند نوع موجودیت استفاده می شود. برای اطلاعات بیشتر در مورد الگوی مخزن، به مطالب ارسالی در [the Entity Framework team's blog](#) و [Julie Lerman's blog](#) مراجعه کنید.

متد `GetDepartments` به منظور اطمینان از اینکه مجموعه برگشتی حتی پس از از بین رفتن شیء مخزن، قابل استفاده است، به جای شیء `IQueryable` شیء `IEnumerable` را بر می گرداند. هنگام دسترسی به یک شیء `IQueryable` این شیء باعث ایجاد دسترسی به دیتابیس می شود. اما شیء مخزن ممکن است هنگامی که کنترل `databound` سعی می کند داده ها را ارائه دهد، از بین برود. به جای شیء `IEnumerable` می توانید نوع دیگری از مجموعه را برگردانید، مانند یک شیء `IList`. هر چند برگرداندن شیء `IEnumerable`، به شما اطمینان می دهد که می توانید تمام عملیات های پردازش لیست های فقط خواندنی را مانند حلقه های `foreach` و پرس و جوهای LINQ اجرا کنید، اما نمی توانید عضوی را به این مجموعه اضافه یا از آن حذف کنید.

یک صفحه `Departments.aspx` که از صفحه `Site.Master` استفاده می کند ایجاد کرده، و کد ساختیافته زیر را در کنترل `Content` با نام `Content2` اضافه کنید.

```
<h2>Departments</h2>
<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.DAL.SchoolRepository"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  SelectMethod="GetDepartments">
</asp:ObjectDataSource>
<asp:GridView ID="DepartmentsGridView" runat="server" AutoGenerateColumns="False"
  DataSourceID="DepartmentsObjectDataSource">
<Columns>
<asp:CommandField ShowEditButton="True" ShowDeleteButton="True"
  ItemStyle-VerticalAlign="Top">
</asp:CommandField>
<asp:DynamicField DataField="Name" HeaderText="Name" SortExpression="Name"
  ItemStyle-VerticalAlign="Top"/>
<asp:DynamicField DataField="Budget" HeaderText="Budget" SortExpression="Budget"
  ItemStyle-VerticalAlign="Top"/>
<asp:DynamicField DataField="StartDate" HeaderText="Start Date"
  ItemStyle-VerticalAlign="Top"/>
<asp:TemplateField HeaderText="Administrator" SortExpression="Person.LastName"
  ItemStyle-VerticalAlign="Top">
<ItemTemplate>
<asp:Label ID="AdministratorLastNameLabel" runat="server" Text='<%=
Eval("Person.LastName") %>'></asp:Label>,
<asp:Label ID="AdministratorFirstNameLabel" runat="server" Text='<%=
Eval("Person.FirstMidName") %>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>
```

این کد ساختیافته یک کنترل `GridView` برای نمایش داده ها و نیز یک کنترل `ObjectDataSource` ایجاد می کند که از کلاس مخزنی که شما ایجاد کرده اید استفاده می کند. کنترل `GridView` دستورات `Edit` و `Delete` را مشخص می کند، اما هنوز شما کدی برای انجام این دستورات اضافه نکرده اید.

ستون های متعددی از کنترل های `DynamicField` استفاده می کنند. بنابراین می توانید از مزایای عملیات قالب بندی و اعتبارسنجی خودکار بهره مند شوید. به این منظور، باید متد `EnableDynamicData` را در گرداننده رخداد `Page_Init` فراخوانی کنید. (کنترل های `DynamicControl` در فیلد `Administrator` استفاده نمی شوند چون آنها با خصوصیت های `Navigation` کار نمی کنند).

صفات `Vertical-Align="Top"` بعداً اهمیت پیدا خواهند کرد. هنگامی که به یک `GridView`، یک ستون که دارای کنترل `GridView` تودرتو است اضافه می کنید.

فایل `Departments.aspx.cs` را باز کرده، و دستور `Using` زیر را اضافه کنید:

```
using ContosoUniversity.DAL;
```

سپس گرداننده زیر را برای رخداد `Init` این صفحه اضافه کنید:

```
protected void Page_Init(object sender, EventArgs e)
{
    DepartmentsGridView.EnableDynamicData(typeof(Department));
}
```

در پوشه `DAL` یک فایل کلاس با نام `Department.cs` ایجاد و کد موجود را با کد زیر جایگزین کنید:


```

using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
namespace ContosoUniversity.DAL
{
    [MetadataType(typeof(DepartmentMetaData))]
    public partial class Department
    {
    }
    public class DepartmentMetaData
    {
        [DataType(DataType.Currency)]
        [Range(0,1000000,ErrorMessage="Budget must be less than $1,000,000.00")]
        public Decimal Budget{get;set;}
        [DisplayFormat(DataFormatString="{0:d}",ApplyFormatInEditMode=true)]
        public DateTime StartDate{get;set;}
    }
}

```

این کد به مدل داده، فراداده اضافه می کند. مشخص می کند که خصوصیت Budget موجودیت Department با وجود اینکه نوع آن Decimal است اما مقدار پول را نشان می دهد و مشخص می کند که مبلغ باید بین 0 و \$1,000,000.00 باشد. همچنین مشخص می کند که خصوصیت StartDate باید به صورت یک تاریخ و با قالب mm/dd/yyyy نشان داده شود.

صفحه Departments.aspx را اجرا کنید.

DEPARTMENTS					
	Name	Budget	Start Date	Administrator	
Edit	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	Delete
Edit	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis	Delete
Edit	English	\$120,000.00	9/1/2007	Li, Yan	Delete
Edit	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy	Delete

توجه کنید با وجود اینکه شما در کد ساختنیافته صفحه Departments.aspx رشته قالب را برای ستون های Budget و StartDate مشخص نکرده اید، قالب بندی مبلغ و تاریخ توسط کنترل های DynamicField و با استفاده از فراداده ای که در فایل Department.cs ایجاد کردید، روی آنها اعمال شده است.

اضافه کردن عملیات های درج و حذف

کد زیر را به منظور ایجاد یک متد Insert و یک متد Delete اضافه کنید. همچنین این کد حاوی متدی با نام GenerateDepartmentID است که مقدار مجاز کلید رکورد بعدی را برای استفاده توسط متد Insert ایجاد می کند. این متد نیاز است چون دیتابیس برای ایجاد خودکار این مقدار برای جدول Department پیگیری نشده است.

```
public void InsertDepartment(Department department)
{
    try
    {
        department.DepartmentID=GenerateDepartmentID();
        context.Department.AddObject(department);
        context.SaveChanges();
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}

public void DeleteDepartment(Department department)
{
    try
    {
        context.Department.Attach(department);
        context.Department.DeleteObject(department);
        context.SaveChanges();
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}

private Int32 GenerateDepartmentID()
{
    Int32 maxDepartmentID =0;
    var department =(from d in GetDepartments()
                    orderby d.DepartmentID descending
                    select d).FirstOrDefault();
    if(department !=null)
    {
        maxDepartmentID = department.DepartmentID+1;
    }
    return maxDepartmentID;
}
```

متد DeleteDepartment، متد Attach را به منظور دوباره برقرار کردن پیوندی که در شیء اداره کننده وضعیت (object state manager) شیء Context، بین موجودیتی در حافظه و سطر موجود در دیتابیس نگه داری می شود، فراخوانی می کند. این متد باید قبل از متد SaveChanges فراخوانی شود.

اصطلاح *object context* به کلاس Entity Framework ای اشاره می کند که از کلاسObjectContext ای که شما برای دستیابی به مجموعه موجودیت ها¹ و موجودیت ها² استفاده می کنید، مشتق شده. در کد این پروژه این کلاس SchoolEntities نامیده شده و یک نمونه از آن همیشه context نامیده می شود. اداره کننده وضعیت شیء مربوط به شیء Context کلاسی است که از کلاس ObjectStateManager مشتق شده. شیء اتصال³ از اداره کننده وضعیت شیء برای ذخیره کردن اشیای موجودیت و نگه داری آنها در حالت همگام با سطرهای متناظر در سطرهای جدول های دیتابیس، استفاده می کند.

وقتی موجودیتی را می خوانید، شیء Context آن را در object state manager ذخیره می کند و پیگیری می کند که ارائه (representation) این شیء (موجودیت) همگام با دیتابیس باشد. به طور مثال، وقتی مقدار خصوصیتی را تغییر می دهید، یک پرچم (flag) تنظیم می شود تا نشان دهد که خصوصیتی که شما تغییر دادید، دیگر با دیتابیس همسان نیست. سپس وقتی که متد SaveChanges را فراخوانی می کنید object context می داند که چه کاری را باید در دیتابیس انجام دهد چون object state manager دقیقاً تفاوت بین وضعیت جاری موجودیت و وضعیت آن در دیتابیس را می داند.

هر چند این فرآیند نوعاً در برنامه های وب کار نمی کند، چون نمونه object context که یک موجودیت را می خواند، همراه اداره کننده وضعیت شیء مربوط به آن، بعد از ارائه (Render) صفحه از بین می روند. نمونه object context که باید تغییرات را اعمال کند، نمونه جدیدی است که برای فرآیندPostBack ایجاد شده است. در مورد، متد DeleteDepartment کنترل ObjectDataSource دوباره نسخه اصلی موجودیت را از مقادیر موجود در وضعیت دید⁴، ایجاد می کند، اما این کار یعنی ایجاد کردن موجودیت Department در اداره کننده وضعیت شیء صورت نمی گیرد. اگر در مورد این موجودیتی که دوباره ایجاد شده است، متد DeleteObject را فراخوانی کرده اید، احتمالاً این فراخوانی fail شده است (شکست خورده)، چون object context نمی داند که آیا این موجودیت همگام با دیتابیس هست یا خیر. هر چند فراخوانی متد Attach باعث می شود به وجود آمدن عملیات پیگیری، بین موجودیت دوباره ایجاد شده و مقادیر موجود در دیتابیس، می شود. (مشابه زمانی در ابتدای کار و هنگامی که این موجودیت در نمونه قبلی object context خوانده شد، به طور خودکار عملیات پیگیری برقرار شد.)

¹ Entity set

² Entity

³ object contact

⁴ View state

موقعی هستند که شما نمی خواهید `object context` موجودیت ها را در اداره کننده وضعیت شیء پیگیری کند، در این صورت می توانید پرچم ها را برای جلوگیری از انجام این کار توسط شیء `object context` تنظیم کنید. مثالهایی در این باره در بخش های بعد نشان داده شده اند.

متد `SaveChanges`

کلاس ساده مخزن، قواعد اصلی چگونگی انجام عملگرهای `CRUD` را نشان می دهد. در این مثال متد `SaveChanges` بعد از هر بروزرسانی، فراخوانی شده است. در یک نرم افزار تولیدی ممکن است بخواهید متد `SaveChanges` را از متد دیگری فراخوانی کنید، تا هنگامی که دیتابیس بروزرسانی می شود کنترل بیشتری داشته باشید. (در انتهای این بخش، لینکی به یک `white paper` را مشاهده خواهید کرد که بخشی از الگوی کار، در مورد یکی از شیوه های هماهنگ سازی بروزرسانی های مرتبط را مطرح می کند). همچنین توجه کنید، در این مثال، متد `DeleteDepartment` حاوی کدی برای اداره کردن تصادم های همزمانی نیست. برای این کار در بخش آینده کدی را اضافه خواهیم کرد.

برگرداندن نام اساتید برای انتخاب در زمان درج

کاربران باید بتوانند هنگام اضافه کردن دانشکده جدید یک مدیر را از لیست کشویی اساتید انتخاب کنند. بنابراین برای ایجاد متدی که با استفاده از دیدی که قبلا ایجاد کرده اید لیست اساتید را برگرداند، کد زیر را به `SchoolRepository.cs` اضافه کنید،

```
public IEnumerable<InstructorName> GetInstructorNames()
{
    return context.InstructorNames.OrderBy("it.FullName").ToList();
}
```

ایجاد یک صفحه برای درج دانشکده ها

صفحه *DepartmentsAdd.aspx* که از *Site.Master* استفاده می کند را ایجاد کنید، و کد ساختیافته زیر را در کنترل

Content با نام Content2 اضافه کنید:

```
<h2>Departments</h2>
<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.DAL.SchoolRepository"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  InsertMethod="InsertDepartment">
</asp:ObjectDataSource>
<asp:DetailsView ID="DepartmentsDetailsView" runat="server"
  DataSourceID="DepartmentsObjectDataSource" AutoGenerateRows="False"
  DefaultMode="Insert" OnItemInserting="DepartmentsDetailsView_ItemInserting">

  <Fields>
  <asp:DynamicField DataField="Name" HeaderText="Name"/>
  <asp:DynamicField DataField="Budget" HeaderText="Budget"/>
  <asp:DynamicField DataField="StartDate" HeaderText="Start Date"/>
  <asp:TemplateField HeaderText="Administrator">
  <InsertItemTemplate>
  <asp:ObjectDataSource ID="InstructorsObjectDataSource" runat="server"
    TypeName="ContosoUniversity.DAL.SchoolRepository"
    DataObjectTypeName="ContosoUniversity.DAL.InstructorName"
    SelectMethod="GetInstructorNames">
  </asp:ObjectDataSource>
  <asp:DropDownList ID="InstructorsDropDownList" runat="server"
    DataSourceID="InstructorsObjectDataSource"
    DataTextField="FullName" DataValueField="PersonID"
    OnInit="DepartmentsDropDownList_Init">
  </asp:DropDownList>
  </InsertItemTemplate>
  </asp:TemplateField>
  <asp:CommandField ShowInsertButton="True"/>
  </Fields>
  </asp:DetailsView>
  <asp:ValidationSummary ID="DepartmentsValidationSummary" runat="server"
    ShowSummary="true" DisplayMode="BulletList"/>
```

این کد ساختیافته دو کنترل *ObjectDataSource* ایجاد می کند، یکی برای درج موجودیت های *Department* جدید و یکی برای برگرداندن نام اساتید برای کنترل *DropDownList* که برای انتخاب مدیران دانشکده ها استفاده می شود. این کد یک کنترل *DetailsView* برای وارد کردن دانشکده جدید، و اداره کننده ای برای رخداد *ItemInserting* کنترل *DetailsView* ایجاد می کند، تا بتوانید کلید خارجی *Administrator* را مقداردهی کنید. در پایان یک کنترل *ValidationSummary* برای نمایش پیغام های خطا وجود دارد.

DepartmentsAdd.aspx.cs را باز کرده و دستور Using زیر را وارد کنید:

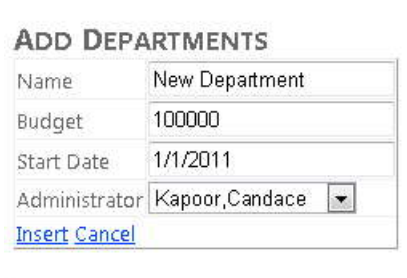
```
using ContosoUniversity.DAL;
```

متدها و متغیرهای کلاسی زیر را اضافه کنید:

```
private DropDownList administratorsDropDownList;
protected void Page_Init(object sender,EventArgs e)
{
    DepartmentsDetailsView.EnableDynamicData(typeof(Department));
}
protected void DepartmentsDropDownList_Init(object sender,EventArgs e)
{
    administratorsDropDownList = sender as DropDownList;
}
protected void DepartmentsDetailsView_ItemInserting(object
sender,DetailsViewItemEventArgs e)
{
    e.Values["Administrator"]= administratorsDropDownList.SelectedValue;
}
```

متد **Page_Init** باعث فعال سازی Dynamic Data functionality می شود. اداره کننده رخداد Init کنترل **DropDownList** یک ارجاع به این کنترل را ذخیره می کند، و اداره کننده رخداد **Inserting** کنترل **DetailsView** از آن ارجاع برای دریافت مقدار **PersonID** استاد انتخاب شده و بروزرسانی خصوصیت کلید خارجی موجودیت **Department** استفاده می کند.

صفحه را اجرا کرده و دانشکده جدیدی اضافه کرده و سپس روی لینک **Insert** کلیک کنید.



ADD DEPARTMENTS	
Name	New Department
Budget	100000
Start Date	1/1/2011
Administrator	Kapoor,Candace
Insert Cancel	

مقادیر را برای دانشکده دیگری وارد کنید. عددی بزرگتر از 1,000,000.00 در فیلد Budget وارد کرده و به فیلد دیگری پرش کنید. در فیلد budget ستاره ای ظاهر می شود، و اگر نشانگر موس را روی آن نگه دارید، پیغام خطایی را که شما در فراداده برای آن مشخص کرده اید نمایش داده می شود.

ADD DEPARTMENTS

Name	New Department 2	
Budget	10000000	*
Start Date	1/1/2011	
Administrator	Abercrombie, Kim	▼
Insert Cancel		

Budget must be less than \$1,000,000.00

روی **Insert** کلیک کنید، پیغام خطایی که توسط کنترل **ValidationSummary** نمایش داده می شود را در پایین صفحه خواهید دید.

ADD DEPARTMENTS

Name	New Department 2	
Budget	100000000	*
Start Date	1/1/2011	
Administrator	Abercrombie, Kim	▼
Insert Cancel		

▪ Budget must be less than \$1,000,000.00

بعد، مرورگر را ببندید و صفحه **Departments.aspx** را باز کنید. قابلیت حذف¹ را به صفحه **Departments.aspx** با اضافه کردن صفت **DeleteMethod** به کنترل **ObjectDataSource** و صفت **DataKeyNames** به کنترل **GridView** اضافه کنید. تگ شروع این کنترل مانند مثال زیر خواهد بود:

¹ Delete

```

<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.DAL.SchoolRepository"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  SelectMethod="GetDepartments"
  DeleteMethod="DeleteDepartment">
</asp:ObjectDataSource>
<asp:GridView ID="DepartmentsGridView" runat="server" AutoGenerateColumns="False"
  DataSourceID="DepartmentsObjectDataSource" DataKeyNames="DepartmentID">

```

صفحه را اجرا کنید.

DEPARTMENTS					
	Name	Budget	Start Date	Administrator	
Edit	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	Delete
Edit	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis	Delete
Edit	English	\$120,000.00	9/1/2007	Li, Yan	Delete
Edit	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy	Delete
Edit	New Department	\$100,000.00	1/1/2011	Kapoor, Candace	Delete

دانشکده ای را که هنگام اجرای *DepartmentsAdd.aspx* اضافه کردید، حذف کنید.

اضافه کردن عملیات بروزرسانی

SchoolRepository.cs را باز کرده و متد **Update** زیر را اضافه کنید:

```
public void UpdateDepartment(Department department, Department origDepartment)
{
    try
    {
        context.Department.Attach(origDepartment);
        context.ApplyCurrentValues("Department", department);
        context.SaveChanges();
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}
```

هنگامی که روی **Update** در صفحه *Departments.aspx* کلیک می کنید، کنترل **ObjectDataSource** دو موجودیت **Department** را به منظور ارسال به متد **UpdateDepartment** ایجاد می کند، یکی حاوی مقادیر اصلی ذخیره شده در وضعیت دید است، و دیگری حاوی مقادیر وارد شده در کنترل **GridView**. کد درون متد **UpdateDepartment** موجودیت **Department** که حاوی مقادیر اصلی است را به متد **Attach** ارسال می کند به منظور برقراری پیگیری همسان بودن این موجودیت و آنچه در دیتابیس است. سپس متد موجودیت **Department** را که دارای مقادیر جدید است به متد **ApplyCurrentValues** ارسال می کند. شی **Context** مقادیر جدید و قدیم را مقایسه می کند، اگر مقدار جدید با مقدار قدیم متفاوت باشد، شی **Context** مقدار خصوصیت را تغییر می دهد. متد **SaveChanges** سپس، تنها ستون های تغییر یافته را در دیتابیس بروزرسانی می کند. (گرچه، اگر عمل بروزرسانی برای این موجودیت به یک رویه ذخیره شده نگاشت شده باشد، تمام سطر بروزرسانی می شود، بدون توجه به اینکه کدام ستون تغییر کرده).

فایل *Departments.aspx* را باز کرده، و صفات زیر را به کنترل **DepartmentsObjectDataSource** اضافه کنید:

- `UpdateMethod="UpdateDepartment"`

- `ConflictDetection="CompareAllValues"`

باعث می شود مقادیر قدیمی در وضعیت دید ذخیره شود، بنابراین می توانند در متد **Update** با مقادیر جدید مقایسه شوند.

• OldValuesParameterFormatString="orig{0}"

کنترل را آگاه می کند که نام مقادیر پارامترهای اصلی، origDepartment است.

اکنون تگ شروع کنترل ObjectDataSource مانند شبیه زیر می باشد:

```
<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.DAL.SchoolRepository"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  SelectMethod="GetDepartments" DeleteMethod="DeleteDepartment"
  UpdateMethod="UpdateDepartment"
  ConflictDetection="CompareAllValues"
  OldValuesParameterFormatString="orig{0}">
```

صفت `OnRowUpdating="DepartmentsGridView_RowUpdating"` را به کنترل `GridView` اضافه کنید. از این صفت برای مقداردهی مقدار خصوصیت `Administrator` بر اساس سطری که کاربر آن را از لیست کشویی انتخاب می کند، استفاده خواهید کرد.

تگ شروع `GridView` اکنون شبیه مثال زیر است:

```
<asp:GridView ID="DepartmentsGridView" runat="server" AutoGenerateColumns="False"
  DataSourceID="DepartmentsObjectDataSource" DataKeyNames="DepartmentID"
  OnRowUpdating="DepartmentsGridView_RowUpdating">
```

یک کنترل `EditItemTemplate` برای ستون `Administrator` به کنترل `GridView` اضافه کنید. دقیقاً بعد از کنترل `ItemTemplate` مربوط به ستون `Administrator`:

```

<EditItemTemplate>
<asp:ObjectDataSource ID="InstructorsObjectDataSource" runat="server"
DataObjectTypeName="ContosoUniversity.DAL.InstructorName"
SelectMethod="GetInstructorNames"
TypeName="ContosoUniversity.DAL.SchoolRepository">
</asp:ObjectDataSource>
<asp:DropDownList ID="InstructorsDropDownList" runat="server"
DataSourceID="InstructorsObjectDataSource"
SelectedValue='<%# Eval("Administrator") %>'
DataTextField="FullName" DataValueField="PersonID"
OnInit="DepartmentsDropDownList_Init" >
</asp:DropDownList>
</EditItemTemplate>

```

کنترل `EditItemTemplate` شبیه به کنترل `InsertItemTemplate` در صفحه `DepartmentsAdd.aspx` است. تفاوت این است که مقدار اولیه این کنترل برای استفاده از صفت `SelectedValue` تنظیم شده است.

همانند صفحه `DepartmentsAdd.aspx` قبل از کنترل `GridView` یک کنترل `ValidationSummary`، اضافه کنید:

```

<asp:ValidationSummary ID="DepartmentsValidationSummary" runat="server"
ShowSummary="true" DisplayMode="BulletList"/>

```

`Departments.aspx.cs` را باز کرده، و فوراً بعد از تعریف کلاس `Partial`، کد زیر را به منظور ایجاد فیلد خصوصی¹ برای ارجاع به کنترل `DropDownList` اضافه کنید:

```
private DropDownList administratorsDropDownList;
```

سپس گرداننده ای برای رخداد `Init` کنترل `DropDownList` و رخداد `RowUpdating` کنترل `GridView` اضافه کنید:

¹ private

```

protected void DepartmentsDropDownList_Init(object sender,EventArgs e)
{
    administratorsDropDownList = sender as DropDownList;
}
protected void DepartmentsGridView_RowUpdating(object
sender,GridViewUpdateEventArgs e)
{
    e.NewValues["Administrator"]= administratorsDropDownList.SelectedValue;
}

```

گرداننده رخداد **Init**، یک ارجاع به کنترل **DropDownList** را در فیلد کلاس ذخیره می کند. گرداننده رخداد مربوط به **RowUpdating** از این ارجاع برای گرفتن مقادیری که کاربر وارد کرده و اعمال آن به خصوصیت **Administrator** موجودیت **Department** استفاده می کند.

از صفحه **DepartmentsAdd.aspx** برای اضافه کردن دانشکده جدید استفاده کنید، سپس صفحه **Departments.aspx** را اجرا کرده و روی گزینه **Edit** سطری که اضافه کرده اید، کلیک کنید.

توجه: به خاطر داده های نامعتبر در دیتابیس، نمی توانید سطری را که شما اضافه نکرده اید(سطری که هم اکنون در دیتابیس هست، یا قبلا بوده) **Edit** کنید مدیرهایی که توسط دیتابیس ایجاد شده اند دانشجو هستند. اگر سعی کنید یکی از آنها را ویرایش کنید، صفحه خطایی دریافت را مشاهده خواهید کرد که گزارش خطایی مانند:

'InstructorsDropDownList' has a SelectedValue which is invalid because it does not exist in the list of items.

را می دهد.

DEPARTMENTS				
	Name	Budget	Start Date	Administrator
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy
Update Cancel	New Department	100000.0000	1/1/2011	Kapoor, Candace

اگر مبلغ نامعتبری برای **Budget** وارد کنید و سپس روی **update** کلیک کنید، ستاره و پیغام خطایی همانند آنچه در صفحه **Departments.aspx** دیدید، مشاهده خواهید کرد.

مقدار یک فیلد را تغییر دهید یا یک مدیر متفاوت انتخاب کنید و روی **Update** کلیک کنید. تغییرات نشان داده شده.

DEPARTMENTS				
	Name	Budget	Start Date	Administrator
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy
Edit Delete	New Department	\$100,000.00	1/1/2011	Kapoor, Candace

در این بخش استفاده از کنترل **ObjectDataSource** برای انجام اعمال اصلی **CRUD (create, read, update, delete)** در **Entity Framework** توضیح داده شد. شما یک برنامه چندلایه¹ ساخته اید، اما لایه منطق تجاری هنوز خیلی با لایه دسترسی به داده به هم تابیده شده اند، که باعث پیچیده شدن تست واحد می شود. در بخش بعد چگونگی پیاده سازی الگوی مخزن برای ساده کردن تست واحد را مشاهده خواهید کرد.

¹ n-tier

اضافه کردن لایه منطق تجاری و تست واحد

در بخش قبل شما یک برنامه وب چندلایه ایجاد کردید که از Entity Framework و کنترل ObjectDataSource استفاده می کند. در این بخش به شما نشان داده خواهد شد، که چگونه منطق تجاری را اضافه کنید در حالی که لایه منطق تجاری (BLL) و لایه منطق دسترسی به داده (DAL) جدای از هم نگه داشته می شوند. همچنین چگونگی ایجاد تست های واحد خودکار برای BLL نشان داده خواهد شد.

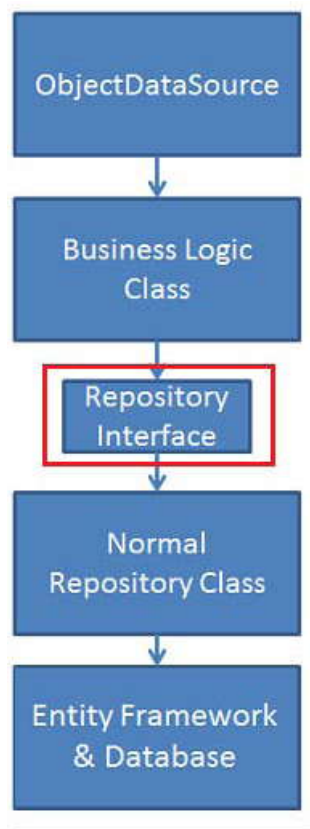
در این بخش وظایف زیر را کامل خواهید کرد:

- ایجاد یک واسط مخزن که متدهای مورد نیاز دسترسی به داده را تعریف می کند.
- پیاده سازی واسط مخزن در کلاس مخزن.
- یک کلاس منطق تجاری ایجاد کنید که کلاس مخزن را برای اجرای عملیات های دسترسی به داده فراخوانی می کند.
- کنترل ObjectDataSource را به جای کلاس مخزن به کلاس منطق تجاری متصل کنید.
- یک پروژه تست واحد و یک کلاس مخزن که از مجموعه های درون حافظه برای ذخیره داده هایش استفاده می کند ایجاد کنید.
- یک تست واحد برای منطق تجاری که می خواهید به کلاس منطق تجاری اضافه کنید، ایجاد کنید، سپس تست را اجرا کنید، شکست آن را مشاهده خواهید کرد.
- منطق تجاری را در کلاس منطق تجاری پیاده سازی کنید، سپس دوباره تست را اجرا کنید، مشاهده می کنید که با موفقیت تست را می گذراند.

از صفحات *Departments.aspx* و *DepartmentsAdd.aspx* که در بخش قبل ایجاد کردید، استفاده می کنیم.

ایجاد واسط مخزن

با ایجاد یک واسط مخزن شروع خواهیم کرد.



در پوشه DAL یک فایل کلاس جدید ایجاد کنید، و نام آن را *ISchoolRepository.cs* قرار دهید، و کد موجود را با کد زیر جایگزین کنید:

```
using System;
using System.Collections.Generic;
namespace ContosoUniversity.DAL
{
    public interface ISchoolRepository:IDisposable
    {
        IEnumerable<Department>GetDepartments();
        void InsertDepartment(Department department);
        void DeleteDepartment(Department department);
        void UpdateDepartment(Department department,Department origDepartment);
        IEnumerable<InstructorName>GetInstructorNames();
    }
}
```

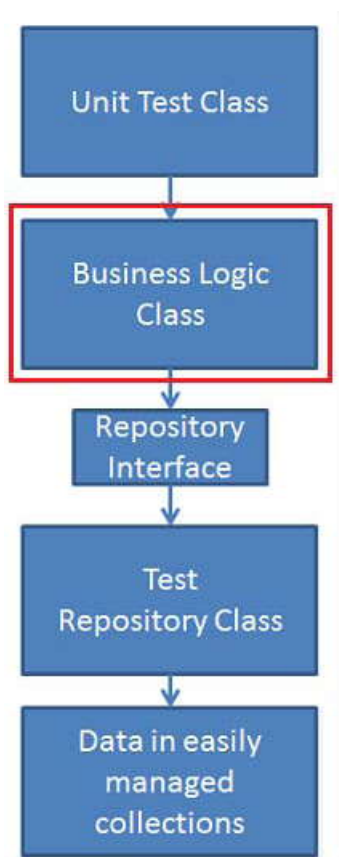
این واسط برای هر کدام از متدهای CRUD (create, read, update, delete) که در کلاس مخزن ایجاد کرده اید، یک متد ایجاد می کند.

در کلاس **SchoolRepository** موجود در *SchoolRepository.cs*، تعیین کنید که این کلاس از واسط **ISchoolRepository** استفاده کند.

```
public class SchoolRepository : IDisposable, ISchoolRepository
```

ایجاد کلاس منطق تجاری

در گام بعد، کلاس منطق تجاری را ایجاد خواهید کرد. این کار را به خاطر این انجام می دهید که بتوانید منطق تجاری که توسط کنترل **ObjectDataSource** اجرا خواهد شد، اضافه کنید. گرچه فعلا این کار را انجام نخواهید داد. اکنون کلاس منطق تجاری جدید فقط عملیات های CRUD مشابه آنچه مخزن انجام می دهد را انجام خواهد داد.



یک پوشه با نام *BLL* ایجاد کنید. (در یک برنامه جهان واقعی یا *real-world* لایه منطق تجاری کاملاً، به عنوان یک کتابخانه کلاس به کار برده می شود- یک پروژه جداگانه – اما برای آسان کردن این بخش آموزشی، کلاس های *BLL* در یک پوشه پروژه نگه داشته می شوند.)

در پوشه *BLL* یک فایل کلاس جدید با نام *SchoolBL.cs* ایجاد کنید و کد موجود را با کد زیر جایگزین کنید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using ContosoUniversity.DAL;
namespace ContosoUniversity.BLL
{
    public class SchoolBL:IDisposable
    {
        private ISchoolRepository schoolRepository;
        public SchoolBL()
        {
            this.schoolRepository =new SchoolRepository();
        }
        public SchoolBL(ISchoolRepository schoolRepository)
        {
            this.schoolRepository = schoolRepository;
        }
        public IEnumerable<Department>GetDepartments()
        {
            return schoolRepository.GetDepartments();
        }
        public void InsertDepartment(Department department)
        {
            try
            {
                schoolRepository.InsertDepartment(department);
            }
            catch(Exception ex)
            {
                //Include catch blocks for specific exceptions first,
                //and handle or log the error as appropriate in each.
                //Include a generic catch block like this one last.
                throw ex;
            }
        }
        public void DeleteDepartment(Department department)
        {
        }
    }
}
```

```

    try
    {
        schoolRepository.DeleteDepartment(department);
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}
public void UpdateDepartment(Department department,Department origDepartment)
{
    try
    {
        schoolRepository.UpdateDepartment(department, origDepartment);
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}
public IEnumerable<InstructorName>GetInstructorNames()
{
    return schoolRepository.GetInstructorNames();
}
private bool disposedValue =false;
protected virtual void Dispose(bool disposing)
{
    if(!this.disposedValue)
    {
        if(disposing)
        {
            schoolRepository.Dispose();
        }
    }
    this.disposedValue =true;
}
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
}
}

```

این کد، متدهای CRUD را شبیه آنچه قبلا در کلاس مخزن، دیدید ایجاد می کند، اما به جای دسترسی مستقیم به متدهای Entity Framework، متدهای مخزن را فراخوانی می کند.

متغیر کلاسی که ارجاع به کلاس مخزن را نگه داری می کند، از نوع واسط تعریف شده، و کدی که کلاس مخزن را نمونه سازی¹ می کند در دو سازنده گنجانده شده است. سازنده² بدون پارامتر توسط کنترل ObjectDataSource استفاده می شوند. این سازنده یک نمونه از کلاس SchoolRepository که قبلا ایجاد کردید، می سازد. سازنده دیگر به کدی که کلاس منطق تجاری را نمونه سازی می کند اجازه دریافت اشیائی را می دهد. که واسط مخزن را به کار می گیرند.

متدهای CRUD که کلاس مخزن را فراخوانی می کنند، و هر دو سازنده ها، امکان استفاده از کلاس منطق تجاری به همراه هر مخزن داده پستی که شما انتخاب می کنید، را فراهم می سازند. کلاس منطق تجاری نیازی به دانستن اینکه چگونه کلاس فراخوانی شده توسط او، ماندگاری داده ها را تامین می کند، ندارد (اغلب *persistence ignorance* نامیده می شود). این کار باعث آسان شدن تست واحد می شود، چون می توانید کلاس منطق تجاری را به یک پیاده سازی مخزن متصل کنید، که از چیزهایی ساده ای مانند مجموعه های List درون حافظه برای ذخیره داده ها استفاده می کند،

توجه: از نظر فنی، اشیای موجودیت، هنوز persistence-ignorant نیستند، چون از کلاسی نمونه سازی شده اند که از کلاس EntityObject موجود در Entity Framework ارث می برند. برای انجام persistence ignorance به جای استفاده از اشیائی که از کلاس EntityObject ارث می برند می توانید از *plain old CLR objects*، یا *POCOs*، استفاده کنید. استفاده از POCOs فراتر از بحث این کتاب است. برای دیدن اطلاعات بیشتر به [Testability and Entity Framework 4.0](#) در سایت MSDN میکروسافت مراجعه کنید.

اکنون می توانید کنترل های ObjectDataSource را به جای مخزن به کلاس منطق تجاری متصل کنید، و بررسی کنید که آیا همه چیز مانند گذشته کار می کند یا خیر.

در Departments.aspx و DepartmentsAdd.aspx مقادیر

`TypeName="ContosoUniversity.DAL.SchoolRepository"` را به

`TypeName="ContosoUniversity.BLL.SchoolBL"` تغییر دهید. (در مجموع چهار مورد وجود دارد).

صفحات Departments.aspx و DepartmentsAdd.aspx را برای بررسی اینکه مانند گذشته کار می کنند، اجرا کنید.

¹ instantiate

² constructor

DEPARTMENTS

	Name	Budget	Start Date	Administrator
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy

ADD DEPARTMENTS

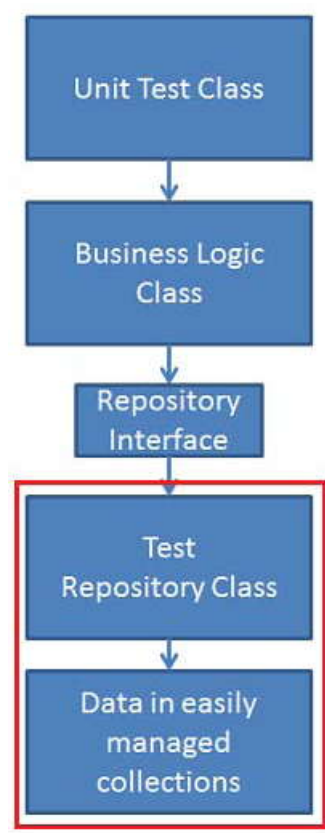
Name	<input type="text"/>
Budget	<input type="text"/>
Start Date	<input type="text"/>
Administrator	Abercrombie, Kim ▾
Insert Cancel	

ایجاد یک پروژه تست واحد، و پیاده سازی مخزن

با استفاده از قالب **Test Project** یک پروژه جدید به solution اضافه کنید، و نام آن را **ContosoUniversity.Tests** قرار دهید.

در پروژه تست، یک ارجاع به **System.Data.Entity**، و یک ارجاع پروژه به **ContosoUniversity** اضافه کنید.

اکنون می توانید کلاس مخزنی را که از آن همراه با تست های واحد استفاده می کنید، ایجاد کنید. محل ذخیره سازی داده ها برای این مخزن، درون کلاس است.



در پروژه تست، یک فایل کلاس با نام *MockSchoolRepository.cs* ایجاد کنید، و کد موجود را با کد زیر جایگزین کنید:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ContosoUniversity.DAL;
using ContosoUniversity.BLL;
namespace ContosoUniversity.Tests
{
    class MockSchoolRepository:ISchoolRepository,IDisposable
    {
        List<Department> departments =new List<Department>();
        List<InstructorName> instructors =new List<InstructorName>();
        public IEnumerable<Department>GetDepartments()
        {
            return departments;
        }
        public void InsertDepartment(Department department)
        {
            departments.Add(department);
        }
    }
}
  
```

```

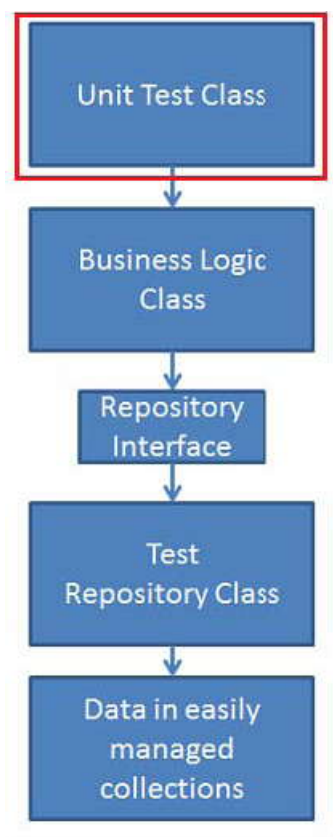
public void DeleteDepartment(Department department)
{
    departments.Remove(department);
}
public void UpdateDepartment(Department department, Department origDepartment)
{
    departments.Remove(origDepartment);
    departments.Add(department);
}
public IEnumerable<InstructorName>GetInstructorNames()
{
    return instructors;
}
public void Dispose()
{
}
}
}

```

این کلاس مخزن دارای متدهای CRUD است مانند کلاس مخزنی که مستقیماً به Entity Framework دسترسی دارد، اما با مجموعه های List درون حافظه به جای دیتابیس استفاده می کنند. این ویژگی باعث می شود تنظیم و آماده سازی کلاس تست و بررسی تست های واحد در مورد منطق تجاری آسان تر شود.

ایجاد تست های واحد

الگوی پروژه **Test** یک کلاس تست برای شما ایجاد کرده، و وظیفه بعدی شما تغییر دادن این کلاس با اضافه کردن متدهای تست واحد به آن برای منطق تجاری که می خواهید به کلاس منطق تجاری اضافه کنید است.



در دانشگاه Contoso، هر استاد تنها می تواند مدیر یک دانشکده باشد، و شما نیاز دارید که منطق تجاری را برای انجام این قانون، اضافه کنید. با اضافه کردن تست ها و اجرای آنها شروع می کنیم تا ببینید که در ابتدا شکست می خورند. سپس کدی را اضافه خواهیم کرد و به امید پذیرفته شدن آنها، دوباره تست ها را اجرا می کنیم.

فایل `UnitTest1.cs` را باز کرده و دستورات `Using` را برای لایه های منطق تجاری و دسترسی به داده ای که در پروژه `ContosoUniversity` ایجاد کرده اید، اضافه کنید:

```

using ContosoUniversity.BLL;
using ContosoUniversity.DAL;
  
```

متد `TestMethod1` را با متدهای زیر جایگزین کنید:

```

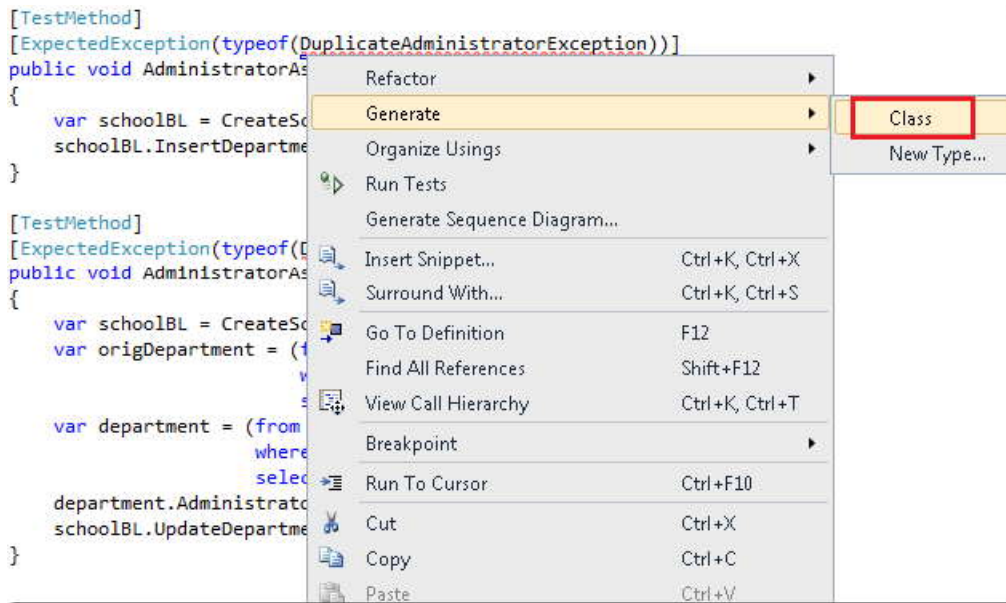
private SchoolBL CreateSchoolBL()
{
    var schoolRepository =new MockSchoolRepository();
    var schoolBL =new SchoolBL(schoolRepository);
    schoolBL.InsertDepartment(new Department(){Name="FirstDepartment",
        DepartmentID=0,Administrator=1,Person=new
Instructor(){FirstMidName="Admin"
        ,LastName="One"}});
    schoolBL.InsertDepartment(new Department(){Name="SecondDepartment",
        DepartmentID=1,Administrator=2,Person=new
Instructor(){FirstMidName="Admin",
        LastName="Two"}});
    schoolBL.InsertDepartment(new Department(){Name="ThirdDepartment",
        DepartmentID=2,Administrator=3,Person=new
Instructor(){FirstMidName="Admin",LastName="Three"}});
    return schoolBL;
}
[TestMethod]
[ExpectedException(typeof(DuplicateAdministratorException))]
public void AdministratorAssignmentRestrictionOnInsert()
{
    var schoolBL =CreateSchoolBL();
    schoolBL.InsertDepartment(new Department(){Name="FourthDepartment",
        DepartmentID=3,Administrator=2,Person=new
Instructor(){FirstMidName="Admin",
        LastName="Two"}});
}
[TestMethod]
[ExpectedException(typeof(DuplicateAdministratorException))]
public void AdministratorAssignmentRestrictionOnUpdate()
{
    var schoolBL =CreateSchoolBL();
    var origDepartment =(from d in schoolBL.GetDepartments()
    where d.Name=="Second Department"
    select d).First();
    var department =(from d in schoolBL.GetDepartments()
    where d.Name=="Second Department"
    select d).First();
    department.Administrator=1;
    schoolBL.UpdateDepartment(department, origDepartment);
}

```

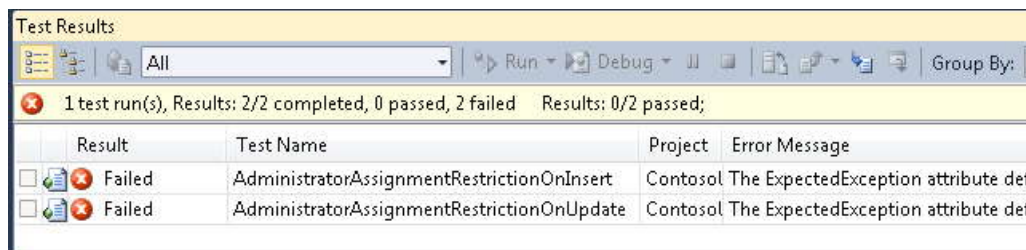
متد CreateSchoolBL، یک نمونه از کلاس مخزنی که برای پروژه تست واحد ایجاد کرده اید را می سازد، که سپس به نمونه جدیدی از کلاس منطق تجاری ارسال می شود. سپس این متد از کلاس منطق تجاری برای درج سه دانشکده که در متدهای تست استفاده می کنید، استفاده خواهد کرد.

در صورتی که کسی سعی کند دانشکده جدیدی با یک مدیر دانشکده موجود درج کند، یا کسی سعی کند مدیر دانشکده ای را با تنظیم ID شخصی که هم اکنون مدیر دانشکده دیگری است بروزرسانی کند، متدهای تست بررسی می کنند که کلاس منطق تجاری استثنایی را پرتاب کند.

شما هنوز کلاس استثنا را ایجاد نکرده اید، بنابراین این کد کامپایل نخواهد شد. برای اینکه کامپایل شود، روی **DuplicateAdministratorException** راست کلیک کرده و **Generate** و سپس **Class** را انتخاب کنید.



این عمل یک کلاس در پروژه تست ایجاد می کند که شما بعد از اینکه کلاس استثنا را در پروژه اصلی ایجاد کردید و پس از پیاده سازی منطق تجاری می توانید آن (کلاس موجود در پروژه تست) را حذف کنید. پروژه تست را اجرا کنید. همان طور که انتظار می رفت، تست رد شد.



اضافه کردن منطق تجاری برای موفقیت در تست

در گام بعد، منطق تجاری را پیاده سازی خواهید کرد، که قرار دادن مدیر یک دانشکده را برای دانشکده دیگر غیر ممکن می کند. اگر کاربری دانشکده ای را ویرایش کرده و پس از انتخاب کسی که هم اکنون مدیر است، روی **Update** کلیک کند، از لایه منطق تجاری یک استثنا پرتاب می کنیم و سپس آن را در لایه ارائه¹ می گیریم (همچنین می توانیم قبل از رندر کردن صفحه، استاتیدی که هم اکنون مدیر هستند را از لیست کشویی حذف کنیم، اما اینجا هدف، کار کردن با لایه منطق تجاری است).

با ایجاد کلاس استثنا شروع کنید، کلاسی که در زمانی که کاربر سعی دارد یک استاد را مدیر بیش از یک دانشکده قرار دهد، پرتاب می شود. در پروژه اصلی یک فایل کلاس در پوشه BLL و با نام *DuplicateAdministratorException.cs* ایجاد کنید، و کد موجود را با کد زیر جایگزین کنید:

```
using System;
namespace ContosoUniversity.BLL
{
    public class DuplicateAdministratorException:Exception
    {
        public DuplicateAdministratorException(string message):base(message)
        {
        }
    }
}
```

فایل موقت *DuplicateAdministratorException.cs* را که قبلا در پروژه تست، برای کامپایل شدن پروژه ایجاد کردید، حذف کنید. در پروژه اصلی *SchoolBL.cs* را باز کرده، کد زیر که حاوی منطق اعتبارسنجی است را وارد اضافه کنید. (این کد به متدی ارجاع دارد که بعدا ایجاد خواهید کرد.)

¹ Presentation

```

private void ValidateOneAdministratorAssignmentPerInstructor(Department department)
{
    if(department.Administrator!=null)
    {
        var duplicateDepartment =schoolRepository.GetDepartmentsByAdministrator(
            department.Administrator.GetValueOrDefault()).FirstOrDefault();
        if(duplicateDepartment !=null && duplicateDepartment.DepartmentID
            != department.DepartmentID)
        {
            throw new DuplicateAdministratorException(String.Format(
                "Instructor {0} {1} is already administrator of the {2} department.",
                duplicateDepartment.Person.FirstMidName,
                duplicateDepartment.Person.LastName,
                duplicateDepartment.Name));
        }
    }
}

```

این متد را هنگامی که می خواهید موجودیت های Department را درج یا بروزرسانی کنید به منظور چک کردن این موضوع که آیا دانشکده دیگری دارای مدیر مشابهی هست یا خیر، فراخوانی خواهید کرد،

کد بالا متدی را برای جستجوی دیتابیس و یافتن موجودیت Department ای که مقدار خصوصیت Administrator آن برابر با موجودیتی است که در حال درج یا بروزرسانی است ، فراخوانی می کند. اگر یک موجودیت یافت شود، این کد یک استثنا پرتاب می کند. اگر موجودیتی که در حال درج یا بروزرسانی است هیچ مقداری برای Administrator نداشته باشد، اعتبارسنجی مورد نیاز نیست، و هیچ استثنایی پرتاب نمی شود. اگر این متد هنگام بروزرسانی فراخوانی شود حتی در صورتی که موجودیت Department مشابهی برای موجودیتی که در حال بروزرسانی است یافت شود، هیچ استثنایی پرتاب نخواهد شد.

متد جدید را از طریق متدهای Insert و Update فراخوانی کنید:

```

public void InsertDepartment(Department department)
{
    ValidateOneAdministratorAssignmentPerInstructor(department);
    try
    ...

public void UpdateDepartment(Department department,Department origDepartment)
{
    ValidateOneAdministratorAssignmentPerInstructor(department);
    try
    ...

```

در `ISchoolRepository.cs` تعریف زیر را برای متد جدید دسترسی داده اضافه کنید:

```
IEnumerable<Department> GetDepartmentsByAdministrator(Int32 administrator);
```

در `SchoolRepository.cs` دستور Using زیر را اضافه کنید:

```
using System.Data.Objects;
```

در `SchoolRepository.cs` متد جدید دسترسی به داده زیر را اضافه کنید:

```
public IEnumerable<Department> GetDepartmentsByAdministrator(Int32 administrator)
{
    return new ObjectQuery<Department>("SELECT VALUE d FROM Department as d",
        context, MergeOption.NoTracking).Include("Person").Where(d =>
        d.Administrator ==
        administrator).ToList();
}
```

این کد موجودیت های `Department` دارای مدیر را بر می گرداند. تنها یک دانشکده باید پیدا شود (if any). اما، چون هیچ محدودیتی در دیتابیس ساخته نشده، نوع بازگشتی به صورت مجموعه از دانشکده های یافت شده است.

به طور پیش فرض، هنگامی که شیء `Context` موجودیت ها را از دیتابیس بر می گرداند، وضعیت آنها را در "اداره کننده وضعیت شیء" خودش پیگیری می کند. پارامتر `MergeOption.NoTracking` مشخص می کند که عملیات پیگیری برای این کوئری انجام نشود. این کار، ضروری است چون کوئری مورد نظر شما ممکن است دقیقاً موجودیتی را که شما سعی در بروزرسانی آن دارید، برگرداند، بنابراین قادر نخواهید بود آن موجودیت را attach کنید. برای مثال، اگر شما در صفحه `Departments.aspx` دانشکده تاریخ را ویرایش کنید و مدیر را بدون تغییر رها کنید، این کوئری دانشکده تاریخ را بر خواهد گرداند. اگر `NoTracking` تنظیم نشده باشد، شیء `Context` ممکن است در حال حاضر، موجودیت دانشکده تاریخ را در اداره کننده وضعیت شیء، داشته باشد. و هنگامی که موجودیت دانشکده تاریخ را که از طریق وضعیت دید دوباره ایجاد شده است، attach می کنید، شیء `Context` استثنایی را پرتاب می کند، که می گوید:

"An object with the same key already exists in the

`ObjectContextManager`. The `ObjectContextManager` cannot track multiple objects with the

same key".

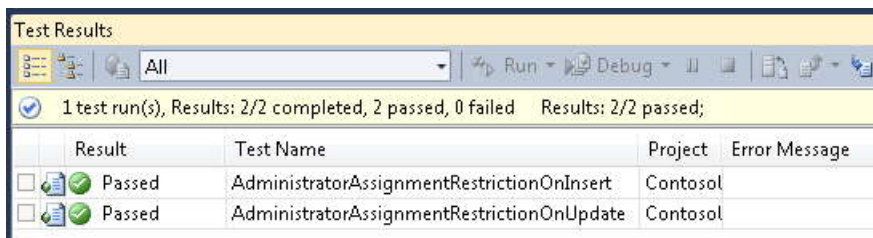
راه دیگر برای مشخص کردن `MergeOption.NoTracking` این است که می توانید فقط برای این کوئری، یک شیء `Context` جدید ایجاد کنید. چون شیء `Context` جدید دارای اداره کننده وضعیت شیء خود خواهد بود، دیگر هنگام فراخوانی متد `Attach` تصادمی رخ نخواهد داد. شیء `Context` جدید فراداده و اتصال دیتابیس¹ را با شیء `Context` اصلی به اشتراک می گذارد، بنابراین امتیاز منفی عملکرد این شیوه جایگزین، کمترین مقدار خواهد بود. شیوه نشان داده شده در اینجا، گزینه `NoTracking` را معرفی می کند، که برای `Context` های دیگر مفید خواهد بود. در مورد گزینه `NoTracking` در بخش بعد بحث شده است.

در پروژه تست، به `MockSchoolRepository.cs` متد دسترسی به داده جدیدی اضافه کنید:

```
public IEnumerable<Department>GetDepartmentsByAdministrator(Int32 administrator)
{
    return(from d in departments
           where d.Administrator== administrator
           select d);
}
```

این کد، همانند مخزن پروژه `ContosoUniversity` از `LINQ to Entities`، برای عملیات انتخاب داده استفاده می کند.

پروژه تست را دوباره اجرا کنید. این بار، تست ها پذیرفته می شوند.



The screenshot shows a 'Test Results' window with a toolbar at the top containing icons for 'All', 'Run', and 'Debug'. Below the toolbar, a summary bar indicates '1 test run(s), Results: 2/2 completed, 2 passed, 0 failed' and 'Results: 2/2 passed;'. The main area contains a table with the following data:

	Result	Test Name	Project	Error Message
<input type="checkbox"/>	Passed	AdministratorAssignmentRestrictionOnInsert	Contosol	
<input type="checkbox"/>	Passed	AdministratorAssignmentRestrictionOnUpdate	Contosol	

¹ Database connection

اداره کردن استثناهای ObjectDataSource

در پروژه **ContosoUniversity** صفحه **Departments.aspx** را اجرا کرده و سعی کنید برای یک دانشکده مدیری را انتخاب کنید که که هم اکنون مدیر دانشکده دیگری است.

(به یاد داشته باشید تنها دانشکده هایی که در این بخش اضافه کرده اید را می توانید ویرایش کنید، چون دیتابیس با داده های نامعتبر پیش بارگذاری می شود.) شما صفحه خطای سرور زیر را مشاهده دریافت خواهید کرد:

Server Error in '/' Application.

Instructor Fadi Fakhouri is already administrator of the Economics department.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: ContosoUniversity.BLL.DuplicateAdministratorException: Instructor Fadi Fakhouri is already administrator of the Economics department.

Source Error:

```
Line 135:         if (duplicateDepartment != null && duplicateDepartment.D
Line 136:         {
Line 137:             throw new DuplicateAdministratorException(String.Fo
Line 138:                 "Instructor {0} {1} is already administrator of
Line 139:                 duplicateDepartment.Person.FirstMidName,
```

Source File: C:\Contoso University\cs\ContosoUniversity\BLL\SchoolBL.cs **Line:** 137

لازم نیست، کاربران این صفحه خطا را ببینند، بنابراین نیاز دارید که کد اداره خطا¹ را اضافه کنید. **Departments.aspx** را باز کنید اداره کننده ای برای رخداد **OnUpdated** مربوط به **DepartmentsObjectDataSource** تعیین کنید. کد ساخت یافته مربوط به **ObjectDataSource** اکنون شبیه مثال زیر خواهد بود:

¹ error-handling

```

<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.BLL.SchoolBL"
  DataObjectType="ContosoUniversity.DAL.Department"
  SelectMethod="GetDepartments" DeleteMethod="DeleteDepartment"
  UpdateMethod="UpdateDepartment"
  ConflictDetection="CompareAllValues"
  OldValuesParameterFormatString="orig{0}"
  OnUpdated="DepartmentsObjectDataSource_Updated">
</asp:ObjectDataSource>

```

در `Departments.aspx.cs`، دستور `Using` زیر را اضافه کنید:

```
using ContosoUniversity.BLL;
```

اداره کننده زیر را برای رخداد `Updated` اضافه کنید:

```

protected void DepartmentsObjectDataSource_Updated(object
  sender, ObjectDataSourceStatusEventArgs e)
{
  if(e.Exception!=null)
  {
    if(e.Exception.InnerException is DuplicateAdministratorException)
    {
      var duplicateAdministratorValidator =new CustomValidator();
      duplicateAdministratorValidator.IsValid=false;
      duplicateAdministratorValidator.ErrorMessage="Update failed: "+
        e.Exception.InnerException.Message;
      Page.Validators.Add(duplicateAdministratorValidator);
      e.ExceptionHandled=true;
    }
  }
}

```

اگر هنگامی که کنترل `ObjectDataSource` سعی در بروزرسانی دارد، یک استثنا را بگیرد، این استثنا را در آرگومان رخداد (e) به اداره کننده ارسال می کند. کد درون اداره کننده رخداد بررسی می کند که آیا این استثنا یک استثنا تکرار مدیر است یا خیر. اگر این گونه باشد، این کد، یک کنترل اعتبارسنجی ایجاد می کند که حاوی یک پیغام خطا برای نمایش در کنترل `ValidationSummary` است.

صفحه را اجرا کرده و دوباره سعی کنید که یک مدیر را برای دو دانشکده قرار دهید. این بار کنترل ValidationSummary یک پیغام خطا نمایش می دهد.

DEPARTMENTS

- Update failed: Instructor Fadi Fakhouri is already administrator of the Economics department.

	Name	Budget	Start Date	Administrator
Edit Delete	Economics	\$999,000.00	9/1/2007	Fakhouri, Fadi
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy
Edit Delete	New Department	\$100,000.00	1/1/2011	Kapoor, Candace

همین تغییرات را در صفحه DepartmentsAdd.aspx اعمال کنید. در DepartmentsAdd.aspx یک اداره کننده برای رخداد OnInserted کنترل DepartmentsObjectDataSource تعیین کنید. در آخر کد ساختیافته شبیه مثال زیر خواهد بود:

```
<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.BLL.SchoolBL"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  InsertMethod="InsertDepartment"
  OnInserted="DepartmentsObjectDataSource_Inserted">
```

در DepartmentsAdd.aspx.cs دستور Using زیر را وارد کنید:

```
using ContosoUniversity.BLL;
```

اداره کنند رخداد زیر را اضافه کنید:


```

protected void DepartmentsObjectDataSource_Inserted(object
sender, ObjectDataSourceStatusEventArgs e)
{
    if(e.Exception!=null)
    {
        if(e.Exception.InnerException is DuplicateAdministratorException)
        {
            var duplicateAdministratorValidator =new CustomValidator();
            duplicateAdministratorValidator.IsValid=false;
            duplicateAdministratorValidator.ErrorMessage="Insert failed: "+
            e.Exception.InnerException.Message;
            Page.Validators.Add(duplicateAdministratorValidator);
            e.ExceptionHandled=true;
        }
    }
}

```

حالا می توانید صفحه *DepartmentsAdd.aspx* را، برای بررسی اینکه همچنان تلاش به منظور ایجاد یک مدیر برای بیش از یک دانشکده را به درستی اداره می کند یا خیر، تست کنید.

به این ترتیب پیاده سازی الگوی مخزن با استفاده از کنترل *ObjectDataSource* در *Entity Framework* توضیح داده شد. برای اطلاعات بیشتر در مورد الگوی مخزن و قابلیت تست به سند [Testability and Entity Framework 4.0](#) مایکروسافت مراجعه کنید.

در بخش بعد چگونگی اضافه کردن عملکردهای مرتب سازی و فیلترینگ به برنامه را مشاهده خواهید کرد.

مرتب سازی و فیلتر کردن

در بخش قبل یک الگوی مخزن در یک برنامه وب چندلایه پیاده سازی کردید که از Entity Framework و کنترل `ObjectDataSource` استفاده می کند. این بخش چگونگی مرتب سازی و فیلتر کردن و اداره کردن سناریوی `master-detail` را نشان می دهد.

- یک جعبه متنی¹ که به کاربران اجازه انتخاب دانشکده ها را به وسیله نام آنها می دهد.
- لیستی از دوره های هر دانشکده که در `Grid` نشان داده شده.
- قابلیت مرتب سازی به وسیله کلیک روی عناوین ستون ها.

DEPARTMENTS

Enter any part of the name or leave blank to see all

	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis	ID	Title
					1050	Chemistry
					1061	Physics
					3000	new course
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan	ID	Title
					2021	Composition
					2030	Poetry
					2042	Literature
Edit Delete	History	\$1,000,000.00	1/10/2011	Zheng, Roger	ID	Title
					1010	US History

¹ Textbox

اضافه کردن قابلیت مرتب سازی ستون های GridView

صفحه *Departments.aspx* را باز کرده، و صفت `SortParameterName="sortExpression"` را به کنترل `ObjectDataSource` به نام `DepartmentsObjectDataSource` اضافه کنید. (بعدهاً متد `GetDepartments` که پارامتری به نام `sortExpression` را می گیرد، ایجاد خواهید کرد.) کد ساختیافته شبیه مثال زیر خواهد بود:

```
<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.BLL.SchoolBL"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  SelectMethod="GetDepartments" DeleteMethod="DeleteDepartment"
  UpdateMethod="UpdateDepartment"
  ConflictDetection="CompareAllValues"
  OldValuesParameterFormatString="orig{0}"
  OnUpdated="DepartmentsObjectDataSource_Updated"
  SortParameterName="sortExpression">
</asp:ObjectDataSource>
```

صفت `AllowSorting="true"` را به تگ شروع کنترل `GridView` اضافه کنید. کد ساختیافته برای این تگ به صورت زیر خواهد بود:

```
<asp:GridView ID="DepartmentsGridView" runat="server" AutoGenerateColumns="False"
  DataSourceID="DepartmentsObjectDataSource" DataKeyNames="DepartmentID"
  OnRowUpdating="DepartmentsGridView_RowUpdating"
  AllowSorting="true">
```

در *Departments.aspx.cs*، ترتیب مرتب سازی پیش فرض را با فراخوانی متد `Sort` کنترل `GridView` در متد `Page_Load` تنظیم کنید.

```
protected void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        DepartmentsGridView.Sort("Name", SortDirection.Ascending);
    }
}
```

کد مرتب سازی یا فیلتر کردن را می توانید در هر یک از کلاس های منطق تجاری یا مخزن اضافه کنید. اگر در کلاس منطق تجاری اضافه کنید، کار مرتب سازی یا فیلتر کردن، بعد از اینکه داده ها از دیتابیس برگردانده شوند، انجام می شود، چون لایه منطق تجاری با شیء **IEnumerable** که توسط مخزن برگردانده شده کار می کند. اگر کد مرتب سازی و فیلتر کردن را در کلاس مخزن اضافه کنید، و این کار را قبل از عبارات LINQ یا قبل از اینکه شیء کوئری¹ به یک شیء **IEnumerable** تبدیل شده باشد انجام دهید، دستورات شما برای پردازش به سمت دیتابیس عبور داده می شود، که این روش نوعاً کارآمد تر است. در این بخش شما مرتب سازی و فیلتر کردن را پیاده سازی خواهید کرد، به روشی که باعث می شود، پردازش، توسط دیتابیس که در مخزن است، انجام گیرد.

برای اضافه کردن قابلیت مرتب سازی، باید متد جدیدی به کلاس واسط مخزن و کلاس مخزن و همچنین کلاس منطق تجاری اضافه کنید. در فایل *ISchoolRepository.cs* یک متد جدید **GetDepartments** اضافه کنید، که پارامتر **sortExpression** را که برای مرتب سازی لیست دانشکده های برگشتی استفاده خواهد شد، می گیرد:

```
IEnumerable<Department> GetDepartments(string sortExpression);
```

پارامتر **sortExpression** ستونی را که باید مرتب شود و همچنین جهت مرتب سازی را مشخص خواهد کرد.

برای متد جدید، کد زیر را در فایل *SchoolRepository.cs* اضافه کنید:

```
public IEnumerable<Department> GetDepartments(string sortExpression)
{
    if (String.IsNullOrEmpty(sortExpression))
    {
        sortExpression = "Name";
    }
    return context.Department.Include("Person").OrderBy("it." + sortExpression).ToList();
}
```

متد بدون پارامتر **GetDepartments** را به منظور فراخوانی متد جدید، تغییر دهید:

```
public IEnumerable<Department>GetDepartments()
{
    returnGetDepartments("");
}
```

¹ object query

در پروژه تست، متد جدید زیر را به *MockSchoolRepository.cs* اضافه کنید:

```
public IEnumerable<Department> GetDepartments(string sortExpression)
{
    return departments;
}
```

اگر قصد ایجاد هر تست واحدی را دارید که وابسته به این متدی است که یک لیست مرتب شده را بر می گرداند، نیاز دارید قبل از برگرداندن لیست، آن را مرتب کنید. این گونه تست ها را در این بخش نیاز نیست ایجاد کنید، چون متد ایجاد شده فقط لیست نامرتبی از دانشکده ها را بر می گرداند.

در فایل *SchoolBL.cs* کد زیر را به کلاس منطق تجاری اضافه کنید:

```
public IEnumerable<Department> GetDepartments(string sortExpression)
{
    return schoolRepository.GetDepartments(sortExpression);
}
```

این کد پارامتر مرتب سازی را به مخزن می فرستد.

صفحه *Departments.aspx* را اجرا کنید.

DEPARTMENTS				
	Name	Budget	Start Date	Administrator
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	History	\$1,000,000.00	1/10/2011	Zheng, Roger
Edit Delete	Mathematics	\$250,000.00	9/1/2007	Justice, Peggy
Edit Delete	New Department	\$100,000.00	1/1/2011	Harui, Roger

اکنون می توانید روی هر کدام از عناوین ستون هایی که می خواهید کلیک کنید، تا بر اساس آن ستون مرتب شود.

اگر در حال حاضر، ستون مرتب شده است، با کلیک روی عنوان آن، جهت مرتب سازی برعکس خواهد شد.

اضافه کردن جعبه متنی جستجو

در این قسمت شما یک جعبه متنی جستجو اضافه خواهید کرد، و با استفاده از یک پارامتر کنترل، آن را به کنترل `ObjectDataSource` پیوند خواهید داد، و یک متد به کلاس منطق تجاری برای پشتیبانی از عملیات فیلتر کردن اضافه خواهید کرد.

صفحه `Departments.aspx` را باز کرده، و کد ساختیافته زیر را بین عنوان و اولین کنترل `ObjectDataSource` اضافه کنید:

```
Enter any part of the name or leave the box blank to see all names:  
<asp:TextBox ID="SearchTextBox" runat="server" AutoPostBack="true"></asp:TextBox>  
<asp:Button ID="SearchButton" runat="server" Text="Search"/>
```

در کنترل `ObjectDataSource` با نام `DepartmentsObjectDataSource` اعمال زیر را انجام دهید:

- یک عنصر `SelectParameters` برای پارامتری با نام `nameSearchString` که مقدار وارد شده در کنترل `SearchTextBox` را بر می گرداند اضافه کنید.
- مقدار صفت `SelectMethod` را به `GetDepartmentsByName` تغییر دهید. (این متد را بعداً ایجاد خواهید کرد).

کد ساختیافته کنترل `ObjectDataSource` اکنون شبیه مثال زیر خواهد بود:

```
<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"  
  TypeName="ContosoUniversity.BLL.SchoolBL"  
  DataObjectName="ContosoUniversity.DAL.Department"  
  SelectMethod="GetDepartmentsByName" DeleteMethod="DeleteDepartment"  
  UpdateMethod="UpdateDepartment"  
  ConflictDetection="CompareAllValues"  
  OldValuesParameterFormatString="orig{0}"  
  OnUpdated="DepartmentsObjectDataSource_Updated"  
  SortParameterName="sortExpression">  
  <SelectParameters>  
  <asp:ControlParameter ControlID="SearchTextBox" Name="nameSearchString"  
    PropertyName="Text"  
    Type="String"/>  
  </SelectParameters>  
</asp:ObjectDataSource>
```

در *ISchoolRepository.cs* متد `GetDepartmentsByName` را که پارامترهای `sortExpression` و `nameSearchString` را دریافت می کند، اضافه کنید:

```
IEnumerable<Department> GetDepartmentsByName(string sortExpression, string nameSearchString);
```

در *SchoolRepository.cs* کد زیر را اضافه کنید:

```
public IEnumerable<Department> GetDepartmentsByName(string sortExpression, string nameSearchString)
{
    if (String.IsNullOrEmpty(sortExpression))
    {
        sortExpression = "Name";
    }
    if (String.IsNullOrEmpty(nameSearchString))
    {
        nameSearchString = "";
    }
    return context.Department.Include("Person").OrderBy("it." + sortExpression).Where(d => d.Name.Contains(nameSearchString)).ToList();
}
```

این کد از متد `Where` برای انتخاب رکوردهایی که حاوی رشته جستجو هستند، استفاده می کند. اگر رشته جستجو خالی باشد، همه رکوردها انتخاب می شوند. توجه کنید وقتی فراخوانی چند متد را در یک دستور انجام می دهید به طور مثال (`include` و `OrderBy` و `Where`) باید همیشه متد `Where` آخر بیاید.

متد `GetDepartments` را که یک پارامتر `sortExpression`، دریافت می کند، برای فراخوانی متد جدید تغییر دهید:

```
public IEnumerable<Department> GetDepartments(string sortExpression)
{
    return GetDepartmentsByName(sortExpression, "");
}
```

در *MockSchoolRepository.cs*، در پروژه تست، متد جدید زیر را اضافه کنید:

```
public IEnumerable<Department> GetDepartmentsByName(string sortExpression, string
    nameSearchString)
{
    return departments;
}
```

در *SchoolBL.cs* متد جدید زیر را اضافه کنید:

```
public IEnumerable<Department> GetDepartmentsByName(string sortExpression, string
    nameSearchString)
{
    return schoolRepository.GetDepartmentsByName(sortExpression,
nameSearchString);
}
```

صفحه *Departments.aspx* را اجرا کنید و رشته جستجو ای را برای اطمینان از کار منطق انتخاب، وارد کنید. جعبه متنی جستجو را خالی گذاشته و جستجویی را به منظور اطمینان از اینکه همه رکوردها برگردانده می شوند انجام دهید.

DEPARTMENTS

Enter any part of the name or leave blank to see all

	Name	Budget	Start Date	Administrator
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan
Edit Delete	New Department	\$100,000.00	1/1/2011	Harui, Roger

افزودن ستون Details برای هر Grid Row

در مرحله بعد شما می خواهید، همه دوره های مربوط به هر کدام از دانشکده های نشان داده شده در سمت راست *gridview* را ببینید. برای انجام این کار، از یک کنترل *GridView* تودرتو استفاده خواهید کرد و آن را به داده های خصوصیت *Courses navigation* موجودیت *Department* مقید خواهید کرد.

Departments.aspx را باز کرده، در کد ساختیافته کنترل **GridView**، اداره کننده ای برای رخداد **RowDataBound** تعیین کنید. کد ساختیافته تگ شروع این کنترل شبیه مثال زیر خواهد بود:

```
<asp:GridView ID="DepartmentsGridView" runat="server" AutoGenerateColumns="False"
DataSourceID="DepartmentsObjectDataSource" DataKeyNames="DepartmentID"
OnRowUpdating="DepartmentsGridView_RowUpdating"
OnRowDataBound="DepartmentsGridView_RowDataBound"
AllowSorting="true">
```

بعد از فیلد الگوی Administrator یک عنصر **TemplateField** جدید اضافه کنید:

```
<asp:TemplateField HeaderText="Courses">
<ItemTemplate>
<asp:GridView ID="CoursesGridView" runat="server" AutoGenerateColumns="False">
<Columns>
<asp:BoundField DataField="CourseID" HeaderText="ID"/>
<asp:BoundField DataField="Title" HeaderText="Title"/>
</Columns>
</asp:GridView>
</ItemTemplate>
</asp:TemplateField>
```

این کد ساختیافته یک کنترل **GridView** تودرتو که تعداد دوره ها و عنوان آنها را نشان می دهد، ایجاد خواهد کرد. این کنترل **data source** را مشخص نمی کند چون در اداره کننده رخداد **RowDataBound** آن را **databind** کرده اید. (به داده های مورد نظر مقید شده است).

Departments.aspx.cs را باز کرده و اداره کننده زیر را برای رخداد **RowDataBound** اضافه کنید:

```
protected void DepartmentsGridView_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if(e.Row.RowType==DataControlRowType.DataRow)
    {
        var department = e.Row.DataItem as Department;
        var coursesGridView =(GridView)e.Row.FindControl("CoursesGridView");
        coursesGridView.DataSource= department.Course.ToList();
        coursesGridView.DataBind();
    }
}
```

این کد موجودیت **Department** را از آرگومان های رخداد دریافت می کند، خصوصیت **Courses navigation** را به مجموعه **List** تبدیل می کند، کنترل **GridView** تودرتو را به این مجموعه مقید می کند.

فایل **SchoolRepository.cs** را باز کرده و **eager loading** را برای خصوصیت **Courses navigation** با فراخوانی متد **Include** در شیء پرس و جویی که در متد **GetDepartmentsByName** ایجاد می کنید، تعیین کنید. دستور **return** در متد **GetDepartmentsByName** اکنون شبیه مثال زیر خواهد بود:

```
return context.Department.Include("Person").Include("Course").
    OrderBy("it." + sortExpression).Where(d =>
        d.Name.Contains(nameSearchString)).ToList();
```

صفحه را اجرا کرده، اضافه بر قابلیت های مرتب سازی و فیلتر کردن که اخیراً اضافه کردید، کنترل **GridView**، اکنون برای هر دانشکده جزئیات دوره ها را به صورت تودرتو نشان می دهد.

DEPARTMENTS						
Enter any part of the name or leave blank to see all						Search
	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis	ID	Title
					1050	Chemistry
					1061	Physics
					3000	new course
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan	ID	Title
					2021	Composition
					2030	Poetry
					2042	Literature
Edit Delete	History	\$1,000,000.00	1/10/2011	Zheng, Roger	ID	Title
					1010	US History

در اینجا توضیحات مربوط به مرتب سازی^۱، فیلتر کردن^۲ و سناریوی **master-detail** شرح داده شد. در بخش آینده چگونگی اداره کردن همزمانی را خواهید دید.

¹ sorting

² filtering

اداره کردن همزمانی¹

در این بخش روی برنامه وب Contoso University که در بخش شروع کار با Entity Framework ([Getting Started with the Entity Framework](#)) ایجاد کردید کار خواهد کرد. اگر بخش های قبل را انجام نداده اید، به عنوان نقطه شروع این بخش می توانید برنامه ای را که باید تا اینجا ایجاد می کردید، از [اینجا](#) دانلود کنید. یا می توانید برنامه کامل این مجموعه آموزشی را از [اینجا](#) دانلود کنید. اگر سوالی درباره بخش های گوناگون این کتاب دارید می توانید آن را به [ASP.NET Entity Framework forum](#) ارسال کنید.

در بخش قبل آموختید که چگونه داده ها را با استفاده از کنترل `ObjectDataSource` و `Entity Framework` مرتب و فیلتر کنید. این بخش گزینه هایی را برای اداره کردن همزمانی، در یک ASP.NET web application که از Entity Framework استفاده می کند، نشان می دهد. نیاز دارید یک صفحه وب جدید ایجاد کنید که برای بروزرسانی دفاتر کار اساتید، استفاده می شود. در این صفحه و صفحه Departments که قبلا ایجاد کردید موضوع همزمانی را استفاده خواهید کرد.

OFFICE ASSIGNMENTS

	Instructor	Location
Edit Delete	Abercrombie, Kim	17 Smith
Edit Delete	Fakhouri, Fadi	29 Adams
Edit Delete	Harui, Roger	37 Williams
Edit Delete	Kapoor, Candace	57 Adams
Edit Delete	Serrano, Stacy	271 Williams
Edit Delete	Stewart, Jasmine	131 Smith
Edit Delete	Van Houten, Roger	213 Smith
Edit Delete	Xu, Kristen	203 Williams
Edit Delete	Zheng, Roger	143 Smith

¹ Handling Concurrency

DEPARTMENTS

Enter any part of the name or leave blank to see all

 Search

	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative
Edit Delete	Engineering	\$350,000.00	9/1/2007	Barzdukas, Gytis	ID	Title
					1050	Chemistry
					1061	Physics
					3000	new course
Edit Delete	English	\$120,000.00	9/1/2007	Li, Yan	ID	Title
					2021	Composition
					2030	Poetry
					2042	Literature
Edit Delete	History	\$1,000,000.00	1/10/2011	Zheng, Roger	ID	Title
					1010	US History

برخوردهای همزمانی^۱

یک برخورد همزمانی هنگامی به وقوع می پیوندد که یک کاربر یک رکورد را ویرایش کند و کاربر دیگری همان رکورد را قبل از ذخیره سازی تغییرات کاربر اول در دیتابیس، ویرایش کند. اگر Entity Framework را برای تشخیص اینگونه برخوردها تنظیم نکنید، آخرین نفری که دیتابیس را بروزرسانی می کند، تغییرات خود را روی تغییرات کاربران دیگر می نویسد. در بسیاری نرم افزارها، این خطر پذیرفته شده است و مجبور نیستید برنامه را برای اداره کردن برخوردهای همزمانی احتمالی پیکربندی کنید. (اگر تعداد کاربران کمی وجود دارند یا تعداد بروزرسانی ها کم است، یا اینکه اگر تغییرات کاربران دوباره نویسی^۲ شوند واقعا مهم و حیاتی نیست، ممکن است هزینه برنامه نویسی برای همزمانی، بیشتر از مزایای آن باشد.) اگر نیاز به توجه به برخوردهای همزمانی ندارید، می توانید از این بخش عبور کنید. دو بخش باقی مانده، در این کتاب، وابسته به چیزهایی که در این بخش می سازید نیستند.

^۱ Concurrency Conflicts

^۲ overwrite

همزمانی بدبینانه^۱

اگر برنامه شما در سناریوی همزمانی، نیاز به جلوگیری، نسبت به از دست رفتن تصادفی داده ها، دارد. یک راه برای انجام آن استفاده از قفل های دیتابیس است. این شیوه، *pessimistic concurrency* نامیده می شود. به طور مثال، قبل از اینکه سطری را از دیتابیس بخوانید، یک قفل برای دستیابی به صورت فقط خواندنی یا بروزرسانی درخواست می دهید. اگر سطری را برای دستیابی بروزرسانی قفل کنید، هیچ کاربری اجازه قفل کردن این سطر را چه برای دسترسی فقط خواندنی یا بروزرسانی ندارد، چون یک کپی از داده ها را دریافت می کنند که در حال تغییر هستند. اگر یک سطر را برای دسترسی فقط خواندنی قفل کنید، دیگر کاربران نیز می توانند برای دسترسی فقط خواندنی آن را قفل کنند، اما برای بروزرسانی نمی توانند.

مدیریت قفل ها دارای معایبی است. ممکن است برنامه نویسی را پیچیده کند. منابع مدیریت دیتابیس زیادی لازم دارد، و ممکن است هنگامی که کاربران افزایش پیدا می کنند، سبب مشکلات کارایی شود. به این دلایل، همه سیستم های مدیریت دیتابیس از همزمانی بدبینانه پیشتیبانی نمی کنند. Entity Framework امکانی برای پشتیبانی آن ارائه نمی دهد. در این بخش به پیاده سازی آن نمی پردازیم.

همزمانی خوشبینانه^۲

جایگزین همزمانی بدبینانه، همزمانی خوشبینانه است. همزمانی خوشبینانه به این معنی است که اجازه داده می شود که برخوردهای همزمانی اتفاق بیفتند، و سپس در صورت رخداد، عکس العمل مناسبی انجام دهد. (به طور مثال john صفحه *Department.aspx* را اجرا کرده، روی لینک **Edit** دانشکده تاریخ کلیک کرده و سپس **Budget** را از \$1,000,000.00 به \$125,000.00 کاهش داده است.)

DEPARTMENTS		
Enter any part of the name or leave blank to see all history		Search
Name	Budget	Start Date
Update Cancel History	125000.00	1/10/2011

^۱ Pessimistic Concurrency (Locking)

^۲ optimistic concurrency

قبل از اینکه John روی **Update** کلیک کند، Jane همان صفحه را اجرا کرده و روی لینک **Edit** دانشکده تاریخ کلیک می کند و سپس فیلد **Start Date** را از 1/10/2011 به 1/1/1999 تغییر می دهد. (Jane دانشکده تاریخ را مدیریت می کند و می خواهد سابقه آن را بیشتر کند).

DEPARTMENTS

Enter any part of the name or leave blank to see all

	Name	Budget	Start Date
Update Cancel	History	1000000.0000	1/1/1999

ابتدا John روی **Update** کلیک می کند، سپس **Jane** روی **Update** کلیک می کند. مرورگر Jane در حال حاضر مبلغ **Budget** را به صورت \$1,000,000.00 لیست می کند که ناصحیح است چون این مبلغ توسط John به \$125,000.00 تغییر یافته است.

بعضی اقدامات که در این سناریو می توانید انجام دهید شامل موارد زیر است:

- می توانید پیگیری کنید چه خصوصیتی توسط یک کاربر دستکاری شده است و تنها ستون های متناظر را در دیتابیس بروزرسانی کنید. در این مثال، در صورت استفاده از این روش، هیچ داده ای از دست نمی رود چون خصوصیات متفاوتی توسط دو کاربر بروزرسانی می شوند. بعد از این هر کس دانشکده تاریخ را جستجو کند 1/1/999 و \$125,000.00 را مشاهده خواهد کرد.

این رفتار پیش فرض Entity Framework است و اساسا می تواند برخوردهایی را که منجر به از دست رفتن داده ها شوند، کاهش دهد. هر چند این رفتار در صورتی که رقابت های همزمان برای تغییر یک خصوصیت یکسان از یک موجودیت باشد، از برخورد جلوگیری نخواهد کرد. به علاوه این رفتار همیشه امکان پذیر نیست. هنگامی که یک روبه ذخیره شده را به یک نوع موجودیت نگاشت می کنید، در صورت هرگونه تغییر این موجودیت در دیتابیس، همه خصوصیت های این موجودیت بروزرسانی می شوند.

- می توانید اجازه دهید تغییرات Jane روی تغییرات John نوشته شود. بعد از کلیک Jane روی **Update**، مقدار **Budget** به \$1,000,000.00 بر می گردد. این کار سناریوی *Client Wins* یا *Last in Wins* نامیده می شود. (مقادیر کلاینت نسبت به آنچه در مخزن داده وجود دارد الویت خواهند داشت).

- می توانید از اعمال تغییرات Jane در دیتابیس جلوگیری کنید. به طور نمونه، شما پیغام خطایی را نشان می دهید، که به او وضعیت کنونی داده ها را نشان می دهد، و به او اجازه می دهد در صورتی که هنوز خواستار اعمال تغییرات است، این کار را دوباره انجام دهد. شما می توانید بیشتر این فرآیند را خودکار کنید، به این صورت که مقدار وارد شده توسط او را


ذخیره کرده و به او فرصت دوباره اعمال کردن آن را بدون وارد کردن مجدد آن، بدهید. این سناریوی *Store Wins* نامیده می شود. (مقادیر مخزن داده (data-store) یا پایگاه داده نسبت به مقادیر ارسالی از طرف کلاینت اولیت داده می شوند.)

تشخیص برخوردهای همزمانی

در Entity Framework، می توانید مشکل برخوردها را با اداره کردن استثنای `OptimisticConcurrencyException` که توسط Entity Framework پرتاب می شود، حل کنید. به منظور دانستن اینکه چه زمانی این استثناء باید پرتاب شوند، Entity Framework باید قادر به تشخیص برخوردها باشد. بنابراین، باید دیتابیس و مدل داده را به طور مناسب پیکربندی کنید. بعضی گزینه ها برای فعال سازی تشخیص برخوردها شامل موارد زیر است:

- در دیتابیس، درون جداول، ستونی گنجانده شود، که می تواند به منظور مشخص کردن زمانی تغییر هر کدام از سطرهای جدول، به کار برده شود. سپس می توانید Entity Framework را برای گنجاندن آن ستون در عبارت `Where` دستورات `SQL Update` یا `SQL Delete`، پیکربندی کنید.

منظور از ستون `Timestamp` در جدول `OfficeAssignment` به این صورت است:

OfficeAssignment	
	InstructorID
	Location
	Timestamp

نوع داده ستون `Timestamp` نیز `Timestamp` نامیده شده. گرچه، این ستون واقعا حاوی مقدار تاریخ یا زمان نیست. در عوض یک عدد متوالی است که هر بار سطر بروزرسانی می شود یک واحد اضافه می شود. در دستور `Update` یا `Delete`، عبارت `Where` مقدار اصلی `Timestamp` را در بر می گیرد. اگر سطر که در حال بروزرسانی است، توسط کاربر دیگری تغییر کرده باشد، مقدار `Timestamp` با مقدار اصلی متفاوت خواهد بود، بنابراین عبارت `Where` هیچ سطر با برای بروزرسانی بر نمی گرداند. وقتی Entity Framework متوجه شد که هیچ سطر توسط دستور `Update` یا `Delete` جاری بروزرسانی نشده، (این زمانی است، که تعداد سطرهای متاثر صفر است) آن را به عنوان یک برخورد همزمانی تفسیر می کند.

- Entity Framework را، به منظور در بر گرفتن مقادیر اصلی هر ستون جدول، در عبارت **Where** دستورات **Delete** و **Update** تنظیم کنید.

در اولین گزینه، اگر ستونی از یک سطر نسبت به اولین باری که آن سطر از دیتابیس خوانده شده است، تغییر کرد، عبارت **Where** هیچ سطر را به عنوان نتیجه بروزسانی، بر نمی گرداند، در نتیجه Entity Framework برخورد همزمانی را تشخیص نمی دهد. این روش به فیلد **Timestamp** وابسته است، اما می تواند ناکارآمد باشد. برای جداول دیتابیزی که ستون های زیادی دارند، می تواند منجر به عبارات **Where** بسیار طولانی شود، و در یک برنامه وب ممکن است نیاز باشد شما مقادیر زیادی از حالات¹ را نگه داری کنید. نگه داری میزان زیادی از حالات ممکن است کارآمدی برنامه را تحت تاثیر قرار دهد چون یا به منابع سرور (به طور مثال **session state**) نیاز دارد یا باید درون خود صفحه گنجانده شود (برای مثال **view state**).

در این بخش، اداره کننده خطا را برای همزمانی خوشبینانه در مورد موجودیتی که خصوصیت **Tracking** (همگام بودن با دیتابیس) ندارد (موجودیت **Department**) و برای موجودیتی که خصوصیت **Tracking** دارد (موجودیت **OfficeAssignment**) اضافه خواهیم کرد.

اداره کردن همزمانی خوشبینانه بدون خصوصیت tracking

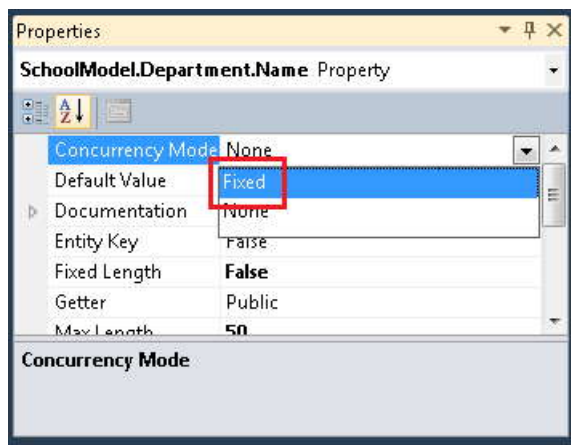
برای پیاده سازی همزمانی خوشبینانه برای موجودیت **Department** که خصوصیت **tracking (Timestamp)** ندارد، وظایف زیر را انجام خواهیم داد:

- مدل داده را برای فعال سازی **tracking** همزمانی برای موجودیت **Department** تغییر دهید.
- در کلاس **SchoolRepository**، استثناهای همزمانی را در متد **SaveChanges** اداره کنید.
- در صفحه **Departments.aspx** استثناهای همزمانی را با نمایش پیغامی به کاربر مبنی بر اینکه تغییرات مورد نظر با موفقیت انجام شد، اداره کنید.

¹ state

فعال سازی Tracking همزمانی در مدل داده

در Visual Studio، برنامه وب Contoso University را که در بخش های قبل با آن کار می کردید، باز کنید. در *SchoolModel.edmx* را باز کرده، و در محیط طراحی روی خصوصیت **Name** موجودیت **Department** راست کلیک و سپس روی **Properties** کلیک کنید. در پنجره **Properties** خصوصیت **ConcurrencyMode** را به **Fixed** تغییر دهید.



این کار را برای بقیه خصوصیت های کمیتی¹ غیر کلید انجام دهید (**StartDate**، **Budget** و **Administrator**). (این کار را برای خصوصیت های **navigation** انجام ندهید.) این کار مشخص می کند که هر زمان Entity Framework دستور (SQL) **Update** یا **Delete** را برای بروزرسانی موجودیت **Department** موجود در دیتابیس تولید می کند، این ستون ها (با مقادیر اصلی شان) باید در عبارت **Where** گنجانده شوند. اگر در زمانی که دستور **Update** یا **Delete** اجرا می شود، هیچ سطری، پیدا نشد، Entity Framework یک استثنای همزمانی خوشبینانه پرتاب می کند.

مدل داده را ذخیره کرده و ببندید.

¹ Scalar

اداره کردن استثنای همزمانی در لایه دسترسی به داده (DAL)

در *SchoolRepository.cs* دستور `using` زیر را برای فضای نام `System.Data` اضافه کنید:

```
using System.Data;
```

متد جدید `SaveChanges` را که استثنای همزمانی را اداره می کند، اضافه کنید:

```
public void SaveChanges()
{
    try
    {
        context.SaveChanges();
    }
    catch(OptimisticConcurrencyException ocex)
    {
        context.Refresh(RefreshMode.StoreWins, ocex.StateEntries[0].Entity);
        throw ocex;
    }
}
```

اگر زمانی که این متد فراخوانی می شود، خطای همزمانی رخ دهد، مقادیر خصوصیت های موجودیتی که درون حافظه است با مقادیر موجود در دیتابیس جایگزین می شود. و استثنای همزمانی دوباره پرتاب می شود تا صفحه وب بتواند آن را اداره کند.

در متدهای `DeleteDepartment` و `UpdateDepartment`، به منظور فراخوانی متد جدید، فراخوانی `context.SaveChanges()` را با `SaveChanges()` جایگزین کنید.

اداره کردن استثنای همزمانی در لایه ارائه¹

را `Departments.aspx` را باز کرده و صفت `OnDeleted="DepartmentsObjectDataSource_Deleted"`

به کنترل `DepartmentsObjectDataSource` اضافه کنید. تگ شروع این کنترل اکنون شبیه مثال زیر خواهد بود:

¹ Presentation Layer

```

<asp:ObjectDataSource ID="DepartmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.BLL.SchoolBL"
  DataObjectTypeName="ContosoUniversity.DAL.Department"
  SelectMethod="GetDepartmentsByName" DeleteMethod="DeleteDepartment"
  UpdateMethod="UpdateDepartment"
  ConflictDetection="CompareAllValues"
  OldValuesParameterFormatString="orig{0}"
  OnUpdated="DepartmentsObjectDataSource_Updated"
  SortParameterName="sortExpression"
  OnDeleted="DepartmentsObjectDataSource_Deleted">

```

در کنترل `DepartmentsGridView`، صفت `DataKeyNames` همان طور که در مثال زیر نشان داده شده، همه ستون های جدول را مشخص می کند. توجه کنید این کار فیلدهای `view state` بزرگی می سازد، این یکی از دلایلی است که معمولاً برای پیگیری برخوردهای همزمانی، استفاده از فیلد `tracking` ترجیح داده می شود.

```

<asp:GridView ID="DepartmentsGridView" runat="server" AutoGenerateColumns="False"
  DataSourceID="DepartmentsObjectDataSource"
  DataKeyNames="DepartmentID,Name,Budget,StartDate,Administrator"
  OnRowUpdating="DepartmentsGridView_RowUpdating"
  OnRowDataBound="DepartmentsGridView_RowDataBound"
  AllowSorting="true">

```

`Departments.aspx.cs` را باز کرده و دستور `using` زیر را برای فضای نام `System.Data` اضافه کنید:

```
using System.Data;
```

متد جدید زیر را اضافه کنید. این متد توسط اداره کننده های رخدادهای `Updated` و `Deleted` متعلق به کنترل منبع داده¹، برای اداره کردن استثنای همزمانی فراخوانی می شود.

¹ Data source

```

private void CheckForOptimisticConcurrencyException(ObjectDataSourceStatusEventArgs
e,string function)
{
    if(e.Exception.InnerException is OptimisticConcurrencyException)
    {
        var concurrencyExceptionValidator =new CustomValidator();
        concurrencyExceptionValidator.IsValid=false;
        concurrencyExceptionValidator.ErrorMessage=
        "The record you attempted to edit or delete was modified by another
"+
        "user after you got the original value. The edit or delete operation
was canceled "+
        "and the other user's values have been displayed so you can "+
        "determine whether you still want to edit or delete this record.";
        Page.Validators.Add(concurrencyExceptionValidator);
        e.ExceptionHandled=true;
    }
}

```

این کد نوع استثنا را بررسی می کند، اگر استثنای همزمانی باشد، این کد به طور خودکار یک کنترل CustomValidator ایجاد می کند که به طور صحیح یک پیغام در کنترل ValidationSummary نشان می دهد.

متد جدید را از اداره کننده رخداد Updated که قبلا ایجاد کرده اید فراخوانی کنید. به علاوه یک اداره کننده برای رخداد Deleted که همان متد را فراخوانی می کند ایجاد کنید (البته فعلا هیچ کار دیگری انجام نمی دهد).

```

protected void DepartmentsObjectDataSource_Updated(object
sender,ObjectDataSourceStatusEventArgs e)
{
    if(e.Exception!=null)
    {
        CheckForOptimisticConcurrencyException(e,"update");
        // ...
    }
}
protected void DepartmentsObjectDataSource_Deleted(object
sender,ObjectDataSourceStatusEventArgs e)
{
    if(e.Exception!=null)
    {
        CheckForOptimisticConcurrencyException(e,"delete");
    }
}

```

آزمودن همزمانی خوشبینانه در صفحه Departments

صفحه Departments.aspx را اجرا کنید.

DEPARTMENTS
Enter any part of the name or leave blank to see all

	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative

روی **Edit** در یک سطر کلیک کرده و مقدار **Budget** را تغییر دهید. (به خاطر داشته باشید تنها رکوردهایی را که در این بخش ایجاد کرده اید می توانید ویرایش کنید، چون دیتابیس School حاوی تعدادی داده های نامعتبر است. رکورد دانشکده اقتصاد (Economics) برای آزمایش ایمن است.)

DEPARTMENTS
Enter any part of the name or leave blank to see all

	Name	Budget	Start Date
Update Cancel	Economics	0.00	9/1/2007

یک پنجره مرورگر جدید باز کنید، و صفحه را دوباره اجرا کنید، (URL را از جعبه آدرس پنجره اول به جعبه آدرس پنجره دوم کپی کنید)

DEPARTMENTS
Enter any part of the name or leave blank to see all

	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$200,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative

روی **Edit** همان سطر که قبلا ویرایش کردید کلیک کنید و مقدار **Budget** را به چیزی دیگری تغییر دهید.

DEPARTMENTS

Enter any part of the name or leave blank to see all

	Name	Budget	Start Date
Update Cancel	Economics	999000.00	9/1/2007

در پنجره مرورگر دوم، روی **Update** کلیک کنید. مشاهده می کنید که مقدار **Budget** با موفقیت به مقدار جدید تغییر کرده است.

DEPARTMENTS

Enter any part of the name or leave blank to see all

	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$999,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative

در پنجره مرورگر اول، روی **Update** کلیک کنید. بروزرسانی شکست می خورد. مقدار **Budget** با همان مقداری که در پنجره مرورگر دوم تنظیم کردید نشان داده می شود، و شما یک پیغام خطا مشاهده خواهید کرد.

DEPARTMENTS

Enter any part of the name or leave the box blank to see all names:

- The record you attempted to edit or delete was modified by another user after you got the original value. The edit or delete operation was canceled and the other user's values have been displayed so you can determine whether you still want to edit or delete this record.

	Name	Budget	Start Date	Administrator	Courses	
Edit Delete	Economics	\$999,000.00	9/1/2007	Fakhouri, Fadi	ID	Title
					4022	Microeconomics
					4041	Macroeconomics
					4061	Quantitative

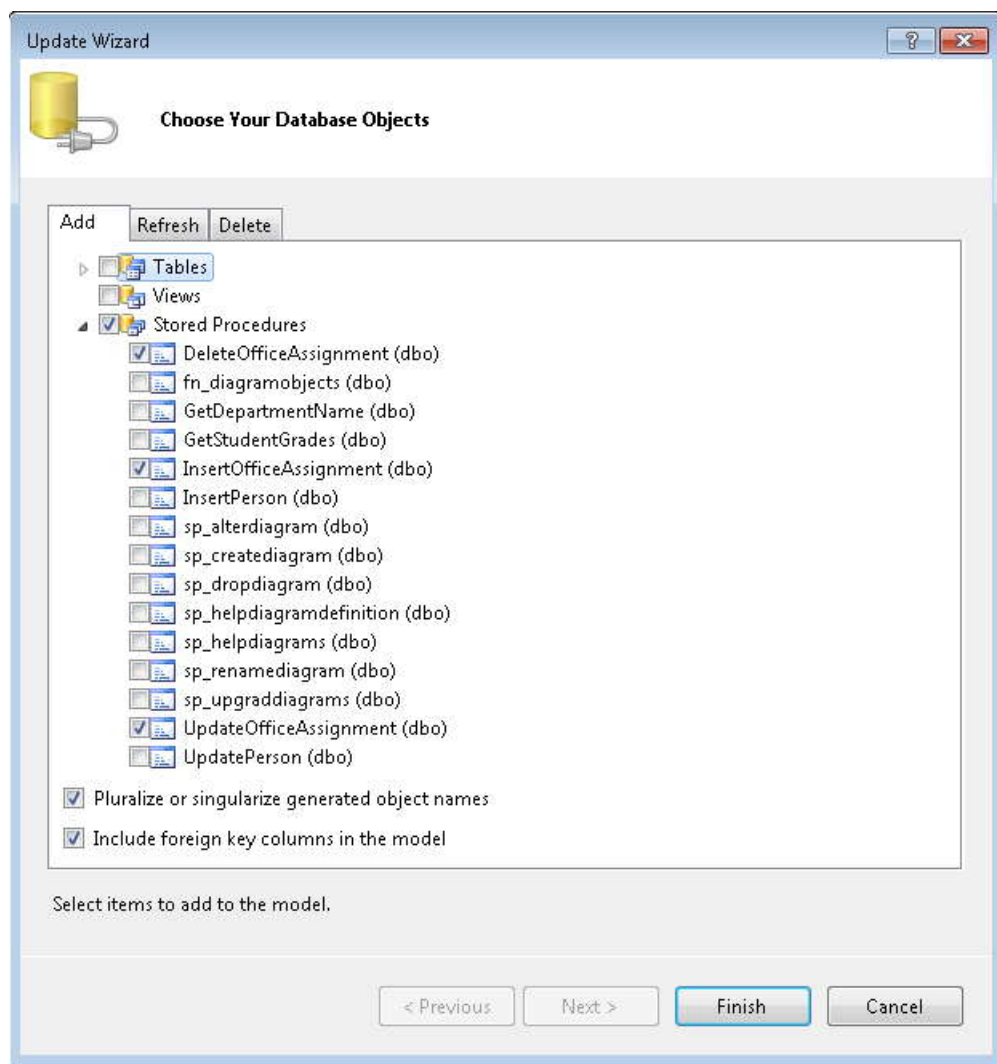
اداره کردن همزمانی خوشبینانه با استفاده از خصوصیت tracking

برای اداره کردن همزمانی خوشبینانه برای یک موجودیت که دارای خصوصیت tracking است، کارهای زیر را انجام دهید:

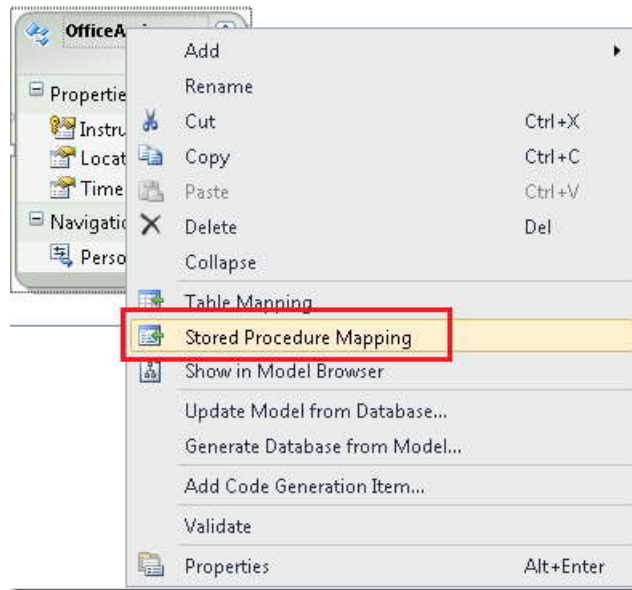
- رویه های ذخیره شده را برای مدیریت موجودیت های **OfficeAssignment** به مدل داده اضافه کنید. (خصوصیت **Tracking** و رویه های ذخیره شده لازم نیست با هم به کار برده شوند، در اینجا فقط جهت تشریح با هم آورده شده اند.)
- متدهای **CRUD** را که حاوی کد اداره کننده استثنای همزمانی خوشبینانه در **DAL** هستند را به **DAL** و **BLL** موجودیت های **OfficeAssignment** اضافه کنید.
- صفحه دفتر کار را اضافه کنید.
- همزمانی خوشبینانه را در صفحه جدید امتحان کنید.

افزودن رویه ذخیره شده **OfficeAssignment** به مدل داده

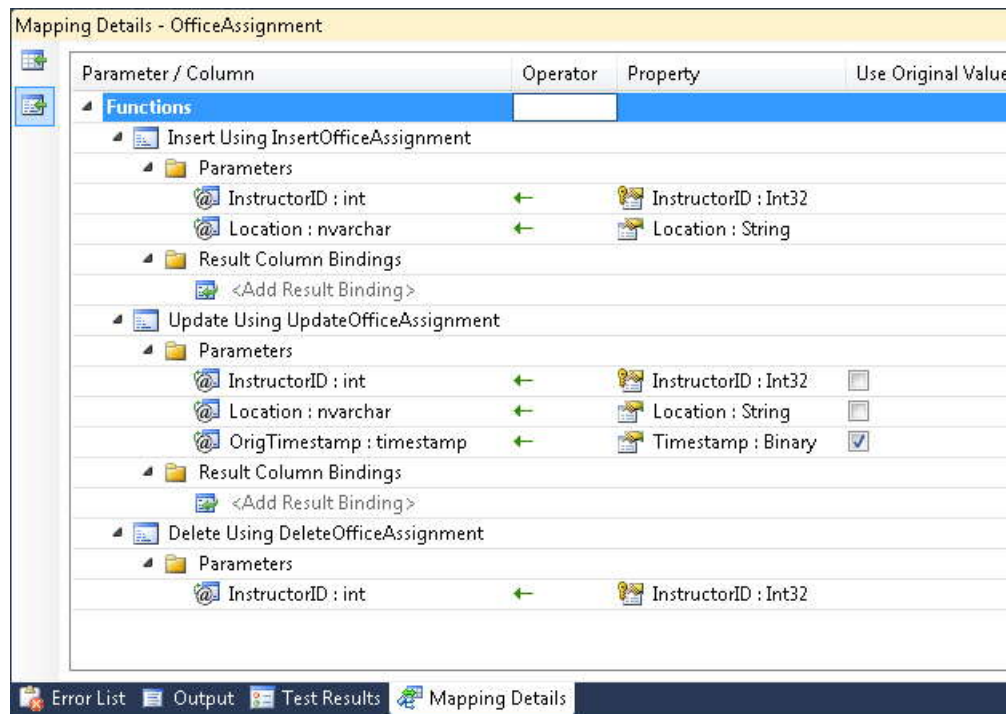
فایل *SchoolModel.edmx* را باز کرده و در محیط طراحی در جای خالی راست کلیک کنید و روی **Update Model from Database** کلیک کنید. در برگه **Add** کادر محاوره ای **Choose Your Database Objects**، گزینه **Stored Procedures** را گسترش داده و سه رویه ذخیره شده **OfficeAssignment** را انتخاب کنید (تصویر زیر را مشاهده کنید)، و روی **Finish** کلیک کنید. (رویه های ذخیره شده در حال حاضر در دیتابیس دانلود شده موجود)



روی موجودیت OfficeAssignment کلیک کرده و **Stored Procedure Mapping** را انتخاب کنید.



توابع **Delete** و **Update** و **Insert** را برای رویه های ذخیره شده متناظر تنظیم کنید. برای پارامتر **OrigTimestamp** تابع **Update** مقدار **Property** را برابر **Timestamp** قرار داده و گزینه **Use Original Value** را انتخاب کنید.



هنگامی که Entity Framework رویه ذخیره شده UpdateOfficeAssignment را فراخوانی می کند، مقدار اصلی ستون Timestamp را در پارامتر OrigTimestamp ارسال می کند. این رویه ذخیره شده این پارامتر را در عبارت Where خود استفاده می کند.

```
ALTER PROCEDURE [dbo].[UpdateOfficeAssignment]
@InstructorIDint,
@Location nvarchar(50),
@OrigTimestamp timestamp
AS
UPDATE OfficeAssignment SET Location=@Location
WHERE InstructorID=@InstructorID AND [Timestamp]=@OrigTimestamp;
IF @@ROWCOUNT>0
BEGIN
SELECT [Timestamp] FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
END
```

این رویه ذخیره شده، همچنین مقدار جدید ستون Timestamp را بعد از بروزرسانی انتخاب می کند تا Entity Framework بتواند موجودیت OfficeAssignment را که در حافظه است با سطر متناظر در دیتابیس همگام نگه دارد.

(توجه کنید رویه ذخیره شده برای حذف یک دفتر کار، دارای پارامتر OrigTimestamp نیست. به همین خاطر، Entity Framework نمی تواند بررسی کند که یک موجودیت قبل از حذف تغییر کرده است یا خیر.)

مدل داده را ذخیره کرده و ببندید.

افزودن متدهای OfficeAssignment به DAL

ISchoolRepository.cs را باز کنید، و متدهای CRUD زیر را برای مجموعه موجودیت OfficeAssignment اضافه

کنید:

```

IEnumerable<OfficeAssignment>GetOfficeAssignments(string sortExpression);
void InsertOfficeAssignment(OfficeAssignment OfficeAssignment);
void DeleteOfficeAssignment(OfficeAssignment OfficeAssignment);
void UpdateOfficeAssignment(OfficeAssignment OfficeAssignment, OfficeAssignment
origOfficeAssignment);

```

متدهای جدید زیر را به *SchoolRepository.cs* اضافه کنید. در متد *UpdateOfficeAssignment* متد *SaveChanges* را به جای متد *context.SaveChanges* فراخوانی کنید:

```

public IEnumerable<OfficeAssignment>GetOfficeAssignments(string sortExpression)
{
    return new ObjectQuery<OfficeAssignment>("SELECT VALUE o FROM
OfficeAssignment AS o",
    context).Include("Person").OrderBy("it."+ sortExpression).ToList();
}
public void InsertOfficeAssignment(OfficeAssignment officeAssignment)
{
    context.OfficeAssignment.AddObject(officeAssignment);
    context.SaveChanges();
}
public void DeleteOfficeAssignment(OfficeAssignment officeAssignment)
{
    context.OfficeAssignment.Attach(officeAssignment);
    context.OfficeAssignment.DeleteObject(officeAssignment);
    context.SaveChanges();
}
public void UpdateOfficeAssignment(OfficeAssignment
officeAssignment, OfficeAssignment
origOfficeAssignment)
{
    context.OfficeAssignment.Attach(origOfficeAssignment);
    context.ApplyCurrentValues("OfficeAssignment", officeAssignment);
    SaveChanges();
}

```

در پروژه تست، *MockSchoolRepository.cs* را باز کرده و مجموعه *OfficeAssignment* و متدهای *CRUD* را به آن اضافه کنید. (مخزن mock باید واسط مخزن را پیاده سازی کند، در غیر این صورت *solution* کامپایل نمی شود).

```

List<OfficeAssignment> officeAssignments =new List<OfficeAssignment>();
public IEnumerable<OfficeAssignment>GetOfficeAssignments(string
sortExpression)
{
return officeAssignments;
}
public void InsertOfficeAssignment(OfficeAssignment officeAssignment)
{
officeAssignments.Add(officeAssignment);
}
public void DeleteOfficeAssignment(OfficeAssignment officeAssignment)
{
officeAssignments.Remove(officeAssignment);
}
public void UpdateOfficeAssignment(OfficeAssignment
officeAssignment,OfficeAssignment
origOfficeAssignment)
{
officeAssignments.Remove(origOfficeAssignment);
officeAssignments.Add(officeAssignment);
}

```

افزودن متدهای OfficeAssignment به BLL

در پروژه اصلی، *SchoolBL.cs* را باز کرده و متدهای CRUD را برای مجموعه موجودیت **OfficeAssignment** به آن

اضافه کنید.

```

public IEnumerable<OfficeAssignment>GetOfficeAssignments(string sortExpression)
{
    if(string.IsNullOrEmpty(sortExpression)) sortExpression = "Person.LastName";
    return schoolRepository.GetOfficeAssignments(sortExpression);
}
public void InsertOfficeAssignment(OfficeAssignment officeAssignment)
{
    try
    {
        schoolRepository.InsertOfficeAssignment(officeAssignment);
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}
public void DeleteOfficeAssignment(OfficeAssignment officeAssignment)
{
    try
    {
        schoolRepository.DeleteOfficeAssignment(officeAssignment);
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}
public void UpdateOfficeAssignment(OfficeAssignment officeAssignment,OfficeAssignment
origOfficeAssignment)
{
    try
    {
        schoolRepository.UpdateOfficeAssignment(officeAssignment,
origOfficeAssignment);
    }
    catch(Exception ex)
    {
        //Include catch blocks for specific exceptions first,
        //and handle or log the error as appropriate in each.
        //Include a generic catch block like this one last.
        throw ex;
    }
}
}

```

ایجاد صفحه وب OfficeAssignments

صفحه وبی با نام *OfficeAssignments.aspx* ایجاد کنید که از *Site.Master* استفاده می کند. کد ساختیافته زیر را به

کنترل *Content* به نام *Content2* اضافه کنید:

```
<h2>Office Assignments</h2>
<asp:ObjectDataSource ID="OfficeAssignmentsObjectDataSource" runat="server"
  TypeName="ContosoUniversity.BLL.SchoolBL"
  DataObjectTypeName="ContosoUniversity.DAL.OfficeAssignment"
  SelectMethod="GetOfficeAssignments"
  DeleteMethod="DeleteOfficeAssignment"
  UpdateMethod="UpdateOfficeAssignment"
  ConflictDetection="CompareAllValues"
  OldValuesParameterFormatString="orig{0}"
  SortParameterName="sortExpression"
  OnUpdated="OfficeAssignmentsObjectDataSource_Updated">
</asp:ObjectDataSource>
<asp:ValidationSummary ID="OfficeAssignmentsValidationSummary" runat="server"
  ShowSummary="true" DisplayMode="BulletList" Style="color:Red; width:40em;"/>
<asp:GridView ID="OfficeAssignmentsGridView" runat="server"
  AutoGenerateColumns="False"
  DataSourceID="OfficeAssignmentsObjectDataSource"
  DataKeyNames="InstructorID,Timestamp"
  AllowSorting="True">
  <Columns>
  <asp:CommandField ShowEditButton="True" ShowDeleteButton="True"
    ItemStyle-VerticalAlign="Top">
  <ItemStyle VerticalAlign="Top">
  </ItemStyle>
  </asp:CommandField>
  <asp:TemplateField HeaderText="Instructor" SortExpression="Person.LastName">
  <ItemTemplate>
  <asp:Label ID="InstructorLastNameLabel" runat="server" Text='<%=
    Eval("Person.LastName") %>'></asp:Label>,
  <asp:Label ID="InstructorFirstNameLabel" runat="server" Text='<%=
    Eval("Person.FirstMidName") %>'></asp:Label>
  </ItemTemplate>
  </asp:TemplateField>
  <asp:DynamicField DataField="Location" HeaderText="Location"
    SortExpression="Location"/>
  </Columns>
  <SelectedRowStyle BackColor="LightGray"></SelectedRowStyle>
</asp:GridView>
```

توجه کنید در صفت `DataKeyNames`، خصوصیت `Timestamp` و همچنین کلید رکورد را مشخص شده اند (`InstructorID`). مشخص کردن خصوصیت ها در صفت `DataKeyNames` باعث می شود که کنترل مربوطه، این خصوصیت ها را درون وضعیت کنترل¹ ذخیره کند (شبيه `view state`) تا مقادیر اصلی در طول فرایند `postback` قابل دسترس باشند.

اگر مقدار `Timestamp` را ذخیره نکنید، `Entity Framework` این مقدار را درون دستور `Update` عبارت `Where` نخواهد داشت. بنابراین هیچ چیز برای بروزرسانی یافت نخواهد شد. در نتیجه `Entity Framework` هر زمان که موجودیت `OfficeAssignment` بروزرسانی شود یک استثنا پرتاب خواهد کرد.

`OfficeAssignments.aspx.cs` را باز کرده و دستور `using` زیر را برای لایه دسترسی به داده اضافه کنید:

```
using ContosoUniversity.DAL;
```

متد `Page_Init` زیر که `Dynamic Data functionality` را فعال می کند، اضافه کنید. اداره کننده زیر را برای رخداد `Updated` کنترل `ObjectDataSource` به منظور بررسی خطاهای همزمانی اضافه کنید:

```
protected void Page_Init(object sender, EventArgs e)
{
    OfficeAssignmentsGridView.EnableDynamicData(typeof(OfficeAssignment));
}
protected void OfficeAssignmentsObjectDataSource_Updated(object
sender, ObjectDataSourceStatusEventArgs e)
{
    if(e.Exception!=null)
    {
        var concurrencyExceptionValidator =new CustomValidator();
        concurrencyExceptionValidator.IsValid=false;
        concurrencyExceptionValidator.ErrorMessage="The record you attempted to "+
"update has been modified by another user since you last visited this page.
"+
"Your update was canceled to allow you to review the other user's "+
"changes and determine if you still want to update this record.";
        Page.Validators.Add(concurrencyExceptionValidator);
        e.ExceptionHandled=true;
    }
}
```

¹ Control state

آزمودن همزمانی خوشبینانه در صفحه OfficeAssignments

صفحه *OfficeAssignments.aspx* را اجرا کنید.

OFFICE ASSIGNMENTS		
	Instructor	Location
Edit Delete	Abercrombie, Kim	17 Smith

روی **Edit** یک سطر کلیک کرده و مقدار ستون **Location** را تغییر دهید.

OFFICE ASSIGNMENTS		
	Instructor	Location
Update Cancel	Abercrombie, Kim	Harry Potter Closet

صفحه مرورگر جدیدی را باز کرده و صفحه را دوباره اجرا کنید. (URL را از پنجره مرورگر اول به پنجره مرورگر دوم کپی کنید)

OFFICE ASSIGNMENTS		
	Instructor	Location
Edit Delete	Abercrombie, Kim	17 Smith

روی **Edit** سطری که اخیراً ویرایش کردید کلیک کرده و مقدار **Location** را به چیز دیگری تغییر دهید.

OFFICE ASSIGNMENTS		
	Instructor	Location
Update Cancel	Abercrombie, Kim	17 Mordor Tower

در پنجره مرورگر دوم روی **Update** کلیک کنید.

OFFICE ASSIGNMENTS

	Instructor	Location
Edit Delete	Abercrombie, Kim	17 Mordor Tower

به پنجره مرورگر اول برگشته و روی **Update** کلیک کنید.

OFFICE ASSIGNMENTS

- The record you attempted to update has been modified by another user since you last visited this page. Your update was canceled to allow you to review the other user's changes and determine if you still want to update this record.

	Instructor	Location
Edit Delete	Abercrombie, Kim	17 Mordor Tower

یک پیغام خطا و مقداری که در پنجره مرورگر دوم تغییر دادید را مشاهده خواهید کرد.

اداره کردن همزمانی با کنترل EntityDataSource

کنترل EntityDataSource دارای منطق ساختاری است که تنظیمات همزمانی را در مدل داده تشخیص داده و عملیات های بروزرسانی و حذف را با توجه به نوع عملیات اداره می کند. گرچه، مانند همه استثناها، باید استثناهای OptimisticConcurrencyException را به منظور ارائه پیغام خطای کاربر پسند¹، خودتان اداره کنید.

در گام بعد، صفحه *Courses.aspx* (که از یک کنترل EntityDataSource استفاده می کند) را به منظور اجازه دادن برای انجام عملیات های بروزرسانی و حذف و نمایش پیغام های خطا درموقعی که برخوردهای همزمانی رخ می دهد، تنظیم خواهید کرد. موجودیت Course دارای ستون پیگیری همزمانی (tracking) نیست، بنابراین شما از متدی مانند آنچه در موجودیت Department به کار بردید، استفاده خواهید کرد. مقادیر خصوصیت های غیر کلید را پیگیری (track) کنید.

¹ user-friendly

فایل *SchoolModel.edmx* را باز کنید. برای خصوصیت های غیر کلید موجودیت **Course** (Title, Credits) و **DepartmentID** (Concurrency Mode) خصوصیت را برابر **Fixed** قرار دهید. سپس مدل داده را ذخیره کرده و ببندید.

صفحه *Courses.aspx* را باز کرده و تغییرات زیر را اعمال کنید:

- در کنترل **CoursesEntityDataSource** صفات **EnableUpdate="true"** و **EnableDelete="true"** را اضافه کنید. تک شروع این کنترل اکنون شبیه مثال زیر خواهد بود:

```
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
    AutoGenerateWhereClause="True"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
    EntitySetName="Course"
    EnableUpdate="true" EnableDelete="true">
```

در کنترل **CoursesGridView** مقدار صفت **DataKeyNames** را به **"CourseID,Title,Credits,DepartmentID"** تغییر دهید. یک عنصر **CommandField** به عنصر **Columns** اضافه کنید، که دکمه های **Edit** و **Delete** را نشان دهد.

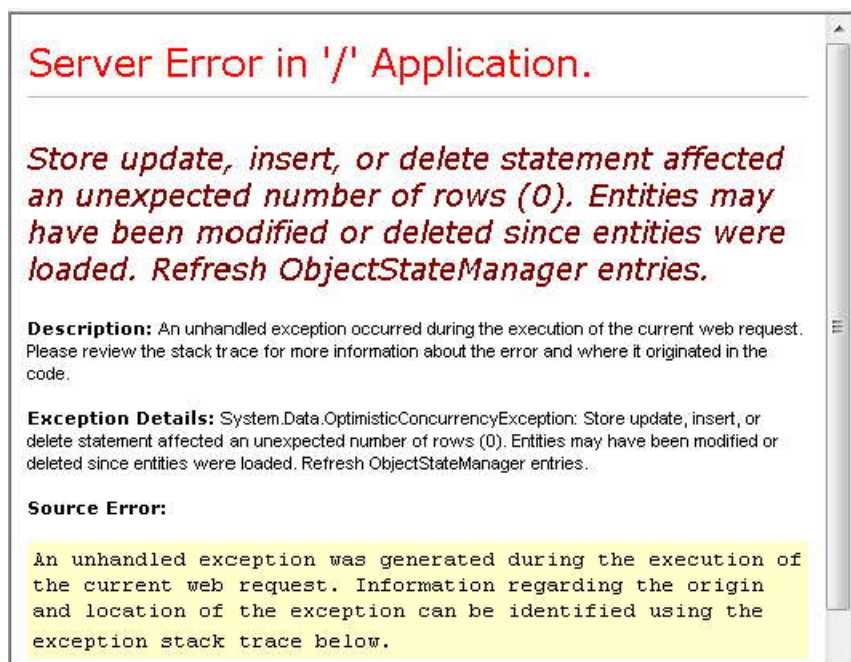
```
( <asp:CommandField ShowEditButton="True" ShowDeleteButton="True" /> )
```

کنترل **GridView** اکنون شبیه مثال زیر خواهد بود:

```
<asp:GridView ID="CoursesGridView" runat="server" AutoGenerateColumns="False"
    DataKeyNames="CourseID,Title,Credits,DepartmentID"
    DataSourceID="CoursesEntityDataSource">
    <Columns>
    <asp:CommandField ShowEditButton="True" ShowDeleteButton="True" />
    <asp:BoundField DataField="CourseID" HeaderText="CourseID" ReadOnly="True"
        SortExpression="CourseID" />
    <asp:BoundField DataField="Title" HeaderText="Title"
        SortExpression="Title" />
    <asp:BoundField DataField="Credits" HeaderText="Credits"
        SortExpression="Credits" />
    </Columns>
</asp:GridView>
```

صفحه را اجرا کرده و یک وضعیت برخورد همانند آنچه قبلا در صفحه **Departments** انجام دادید، ایجاد کنید. این صفحه را در دو پنجره مرورگر باز کرده و روی **Edit** یک سطر متناظر در دو پنجره کلیک کرده و در هر کدام تغییری ایجاد کرده و روی

Update در یکی از پنجره ها کلیک کرده و سپس روی **Update** در پنجره دیگر کلیک کنید. زمانی که برای بار دوم روی **Update** کلیک کنید، یک صفحه خطا مشاهده خواهید کرد که در اثر یک برخورد همزمانی اداره نشده¹ است.



این خطا را به روشی مشابه آنچه در مورد کنترل `ObjectDataSource` انجام دادید اداره کنید. صفحه `Courses.aspx` را باز کرده، در کنترل `CoursesEntityDataSource`، اداره کننده هایی برای رخدادهای `Deleted` و `Updated` تعیین کنید. تگ شروع این کنترل اکنون مانند مثال زیر خواهد بود:

```
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
    AutoGenerateWhereClause="True"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="Course"
    EnableUpdate="true" EnableDelete="true"
    OnDeleted="CoursesEntityDataSource_Deleted"
    OnUpdated="CoursesEntityDataSource_Updated">
```

قبل از کنترل `CoursesGridView` کنترل `ValidationSummary` زیر را اضافه کنید:

¹ unhandled

```
<asp:ValidationSummary ID="CoursesValidationSummary" runat="server"
    ShowSummary="true" DisplayMode="BulletList"/>
```

در *Courses.aspx.cs* دستور `using` را برای استفاده از فضای نام `System.Data` اضافه کنید، یک متد برای بررسی استثنای همزمانی اضافه کنید و اداره کننده هایی برای رخدادهای `Updated` و `Deleted` کنترل `EntityDataSource` اضافه کنید. کد مورد نظر اکنون شبیه کد زیر خواهد بود:

```
protected void CoursesEntityDataSource_Updated(object
    sender, EntityDataSourceChangedEventArgs e)
    {
        CheckForOptimisticConcurrencyException(e, "update");
    }
protected void CoursesEntityDataSource_Deleted(object
    sender, EntityDataSourceChangedEventArgs e)
    {
        CheckForOptimisticConcurrencyException(e, "delete");
    }
private void
CheckForOptimisticConcurrencyException(EntityDataSourceChangedEventArgs
    e, string function)
    {
        if(e.Exception!=null && e.Exception is OptimisticConcurrencyException)
        {
            var concurrencyExceptionValidator =new CustomValidator();
            concurrencyExceptionValidator.IsValid=false;
            concurrencyExceptionValidator.ErrorMessage=
                "The record you attempted to edit or delete was modified by another "+
                "user after you got the original value. The edit or delete operation was
                canceled "+
                "and the other user's values have been displayed so you can "+
                "determine whether you still want to edit or delete this record.";
            Page.Validators.Add(concurrencyExceptionValidator);
            e.ExceptionHandled=true;
        }
    }
}
```

تنها تفاوت این کد با آنچه شما در مورد کنترل `ObjectDataSource` انجام دادید این است که در این مورد، استثنای همزمانی، به جای خصوصیت `InnerException` آن استثنا، در خصوصیت `Exception` شی آرگومان های رخداد¹ قرار گرفته است.

صفحه را اجرا کرده و بار دیگر یک برخورد همزمانی ایجاد کنید. این بار یک پیغام خطا را مشاهده خواهید کرد.

¹ event arguments object

COURSES BY DEPARTMENT

Select a department:

- The record you attempted to edit or delete was modified by another user after you got the original value. The edit or delete operation was canceled and the other user's values have been displayed so you can determine whether you still want to edit or delete this record.

	CourseID	Title	Credits
Edit Delete	1050	Chemistry	4
Edit Delete	1061	Physics	4
Edit Delete	4062	New Engineering Course	5

در این بخش بحث در مورد برخوردهای همزمانی تمام می شود. بخش بعدی دستورالعملی در مورد چگونگی بهبود کارایی برنامه های تحت وب که از Entity Framework استفاده می کنند، ارائه می دهد.

بالا بردن کارایی

در بخش قبل چگونگی اداره کردن برخوردهای همزمانی را مشاهده کردید. این بخش به شما گزینه هایی را برای بهبود کارایی یک برنامه تحت وب ASP.NET که از Entity Framework استفاده می کند، نشان می دهد. در این بخش متدهای (شیوه های) متعددی برای بالا بردن کارایی و یا رفع مشکلات کارایی خواهید آموخت.

اطلاعاتی که در بخش های زیر ارائه شده است، احتمالاً در سناریوهای مختلف زیادی مفید خواهد بود.

- بارگذاری موثر داده های مرتبط^۱
- مدیریت وضعیت دید^۲

اطلاعات ارائه شده در بخش های زیر، ممکن است در صورتی که پرس و جویهای جداگانه ای داشته باشید که باعث ایجاد مشکلات کارایی می شوند، مفید باشند:

- استفاده از گزینه NoTracking
- پردازش پرس و جویهای LINQ قبل از کامپایل برنامه (Pre-compile LINQ queries)
- بررسی دستورات پرس و جویی که به دیتابیس فرستاده می شوند.

اطلاعات ارائه شده در بخش زیر ممکن است برای برنامه هایی که دارای مدل های داده بسیار بزرگ هستند مفید باشد:

- Pre-generate views

توجه: کارایی برنامه کاربردی تحت وب، متأثر از عوامل بسیاری است، شامل چیزهایی مانند اندازه داده های درخواست (request) و پاسخ (response)، سرعت پرس و جویهای دیتابیس، چه تعداد درخواست را دیتابیس می تواند در صف قرار دهد و با چه سرعتی می تواند به آنها سرویس دهد، و حتی تاثیر کتابخانه های اسکریپت کلاینت^۳ که ممکن است از آنها استفاده کنید. اگر کارایی در برنامه شما امری حیاتی است، یا اگر تست ها و آزمایش ها نشان می دهند که کارایی برنامه رضایت بخش نیست، باید از یک پروتکل معمولی برای تنظیم کارایی استفاده کنید. بررسی کنید کجا تگنهای کارایی روی می دهند و سپس به سراغ ناحیه ای بروید که بیشترین اثر را روی کارایی برنامه دارد.

¹ Efficiently load related data

² Manage view state

³ client-script libraries



این بخش روی شیوه‌هایی متمرکز می‌شود که به طور بالقوه می‌توانید با آنها کارایی را بهبود ببخشید، مخصوصاً در مورد Entity Framework در ASP.NET. پیشنهادهایی که اینجا ذکر می‌شود در صورتی که متوجه شده‌اید که دسترسی به داده یکی از تنگناهای کارایی در برنامه شما است مفید خواهد بود. توجه فرمائید متدهایی که این جا توضیح داده می‌شوند نباید در همه جا به عنوان بهترین شیوه‌ها دانسته شوند. بسیاری از آنها تنها در موارد استثنایی، مناسب هستند یا برای شناسایی محل انواع بسیار خاص تنگناهای کارایی.

برای شروع این بخش، Visual Studio را باز کرده و برنامه وب Contoso University را که در بخش‌های قبل با آن کار می‌کردید باز کنید.

بارگذاری موثر داده‌های مرتبط^۱

راه‌های متعددی وجود دارد که Entity Framework می‌تواند داده‌های مرتبط را در خصوصیت‌های navigation یک موجودیت بارگذاری کند:

- *Lazy loading*. هنگامی که یک موجودیت برای اولین بار خوانده می‌شود، داده‌های مرتبط برگردانده نمی‌شوند. هر چند در بار نخست که تلاش می‌کنید به یک خصوصیت navigation دسترسی پیدا کنید، داده‌های مورد نیاز برای آن خصوصیت navigation به طور خودکار برگردانده می‌شوند. نتیجه این کار پرس و جوهای متعددی است که به دیتابیس فرستاده می‌شوند- هر زمان که داده‌های مرتبط برای آن موجودیت باید برگردانده شوند. به ازای هر کدام از خصوصیت‌های navigation یک اتصال با دیتابیس ایجاد می‌شود.

```
departments = context.Departments.ToList();  Read Department rows
foreach (Department d in departments)
{
    if (d.Name == "History")
    {
        adminLastName = d.Person.LastName;  Read one Person row
    }
}
```

^۱ Efficiently Loading Related Data

- *Eager loading*. هنگامی که یک موجودیت خوانده می شود، داده های مرتبط با او برگردانده می شوند. معمولاً برای این کار یک پرس و جوی `join` همه داده های مورد نیاز را برمی گرداند. `eager loading` را با استفاده از متد `Include` می توانید مشخص کنید. همانگونه که در بخش های قبل دیدید.

```

departments = context.Departments.Include("Person").ToList();
foreach (Department d in departments)
{
    if (d.Name == "History")
    {
        adminLastName = d.Person.LastName;
    }
}

```

Read Department rows and all related Person rows

- *Explicit loading*. شبیه `lazy loading` است به جز اینکه شما صریحاً داده های مرتبط را در کد برنامه نویسی بر می گردانی. به طور خودکار هنگام دستیابی به خصوصیت های `navigation` انجام نمی شود. شما با استفاده از متد `Load` خصوصیت `navigation`. به طور دستی داده های مرتبط را برای مجموعه ها بارگذاری می کنید. یا از متد `Load` خصوصیت ارجاع¹ برای خصوصیت هایی که یک شیء را نگه داری می کنند استفاده کنید. (برای مثال شما متد `PersonReference.Load` را برای بارگذاری خصوصیت `Person navigation` موجودیت `Department` فراخوانی می کنید)

```

departments = context.Departments.ToList();
foreach (Department d in departments)
{
    if (d.Name == "History")
    {
        d.PersonReference.Load();
        adminLastName = d.Person.LastName;
    }
}

```

Read Department rows

Read one Person row

چون `explicit loading` و `lazy loading` فوراً مقدار خصوصیت ها را بر نمی گردانند هر دو به عنوان بارگذاری معوق (`deferred loading`) شناخته می شوند.

`Lazy loading` حالت پیش فرض شیء `context` تولید شده توسط `designer` (طراح مدل داده) است. اگر فایل `SchoolModel.Designer.cs` که کلاس `context` را تعریف کرده است باز کنید، سه متد سازنده را خواهید یافت که هر کدام از آنها حاوی دستور زیر است:

¹ Reference


```
this.ContextOptions.LazyLoadingEnabled=true;
```

به طور کلی، اگر می دانید برای هر موجودیتی که برگردانده می شود به داده های مرتبط نیاز دارید، eager loading بهترین کارایی را ارائه می دهد، چون یک پرس و جو را به دیتابیس ارسال می کند نوعا کارآمد تر از پرس و جوهای جداگانه است که برای هر موجودیت برگردانده می شوند. از طرف دیگر، اگر به ندرت نیاز به دسترسی به خصوصیت های navigation یک موجودیت دارید یا برای مجموعه های کوچک موجودیت ها، lazy loading یا explicit loading ممکن است کارآمد تر باشند، چون eager loading داده هایی بیشتر از آنچه نیاز دارید، بر می گرداند.

در یک برنامه وب lazy loading ممکن است از ارزش نسبتا کمی برخوردار باشد، چون اقدامات کاربر که منجر به نیاز برای داده های مرتبط می شود در مرورگری انجام می گیرد، که هیچ اتصالی به شیء Context ای که صفحه را ارائه می دهد ندارد. از طرف دیگر، وقتی یک کنترل را مقید (databind) می کنید، معمولا می دانید که چه داده هایی را نیاز دارید، و بنابراین معمولا eager loading یا deferred loading بسته به هر سناریو بهترین انتخاب است.

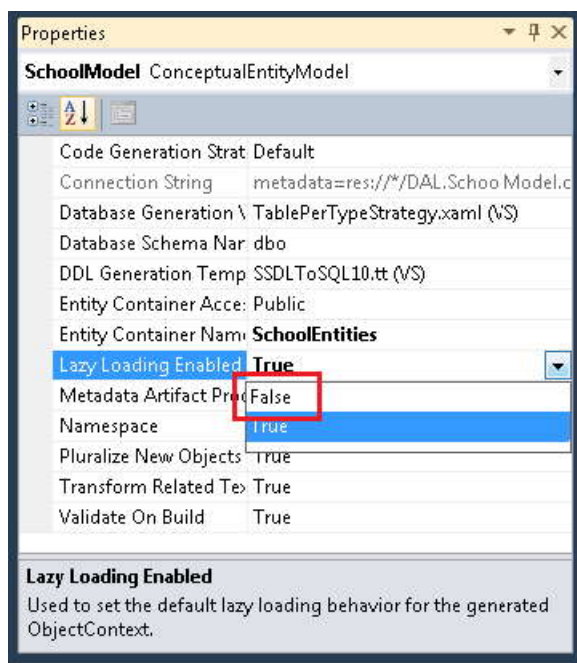
به علاوه یک کنترل databound ممکن است بعد از از بین رفتن شیء Context، از یک شیء موجودیت استفاده کند. در این صورت، تلاش برای lazy-load یک خصوصیت navigation شکست خواهد خورد. پیغام خطایی که شما دریافت می کنید به این صورت است:

```
"The  
ObjectContext instance has been disposed and can no longer be used for operations  
that require a connection."
```

کنترل EntityDataSource به طور پیش فرض lazy loading را غیر فعال می کند. برای کنترل ObjectDataSource که در این بخش استفاده می کنید (یا اگر از طریق کد صفحه به شیء Context دسترسی پیدا می کنید)، راه های متعددی وجود دارد که می توانید به طور پیش فرض lazy loading را غیر فعال کنید. می توانید هنگامی که یک شیء Context را نمونه سازی می کنید، lazy loading را غیر فعال کنید. برای مثال، می توانید خط زیر را به متد سازنده کلاس SchoolRepository اضافه کنید:

```
context.ContextOptions.LazyLoadingEnabled=false;
```

برای برنامه Contoso University، کاری می‌کنید که شیء context به طور خودکار lazy loading را غیر فعال کند، بنابراین نیاز نیست هر بار که Context نمونه‌سازی می‌شود، این خصوصیت تنظیم شود. مدل داده SchoolModel.edmx را باز کرده، روی سطح خالی آن کلیک کنید و مقدار Lazy Loading Enabled را برابر False قرار دهید. مدل را ذخیره کرده و ببندید.



مدیریت وضعیت دید¹

به منظور ارائه عملکرد بروزرسانی، یک برنامه وب ASP.NET باید مقادیر خصوصیت‌های اصلی یک موجودیت را هنگامی که صفحه ارائه می‌شود، ذخیره کند. در طول فرایند Postback این صفحه می‌تواند، وضعیت اصلی آن موجودیت را دوباره ایجاد کرده و متد Attach موجودیت را قبل از اعمال تغییرات و فراخوانی متد SaveChanges دوباره ایجاد کند. به طور پیش فرض، کنترل‌های داده ASP.NET Web Form از وضعیت دید برای ذخیره کردن مقادیر اصلی استفاده می‌کنند. گرچه، وضعیت دید می‌تواند کارایی را تحت تاثیر قرار دهد، چون در فیلدهای مخفی ذخیره می‌شود ه می‌تواند اساساً اندازه صفحه‌ای که به و از مرورگر ارسال می‌شود را افزایش دهند.

¹ View State

تکنیک های مدیریت وضعیت دید، یا جایگزین هایی مانند وضعیت نشست¹، منحصر به Entity Framework نیستند، بنابراین این بخش به طور مفصل وارد این موضوع نمی شود. برای اطلاعات بیشتر لینک های موجود در انتهای این بخش را مشاهده کنید.

به هر حال، نسخه ASP.NET 4 راه جدیدی برای کار با وضعیت دید ارائه می دهد که هر توسعه دهنده برنامه های ASP.NET web form باید از آن آگاه باشند: خصوصیت `ViewStateMode`. این خصوصیت جدید می تواند در سطح یک صفحه یا یک کنترل تنظیم شود، و شما را قادر می سازد که بتوانید وضعیت دید را به طور پیش فرض برای یک صفحه غیر فعال کنید و آن را در صورت نیاز برای کنترل هایی که به آن نیاز دارند فعال کنید.

برای برنامه ها، در جایی که کارایی امری حیاتی است، یک شیوه خوب این هست که همیشه وضعیت دید در سطح صفحه غیر فعال باشد و تنها برای کنترل هایی که به آن نیاز دارند فعال باشد. اندازه وضعیت دید در صفحات `Contoso University` اساساً به کارگیری این شیوه افزایش پیدا نخواهد کرد، اما برای اینکه ببینید چگونه کار می کند، این شیوه را در مورد صفحه `Instructors.aspx` به کار بگیرید. این صفحه دارای کنترل های زیادی، از جمله یک کنترل `Label` که وضعیت دید آن غیر فعال شده است می باشد. هیچ کدام از کنترل های این صفحه واقعاً به وضعیت دید فعال نیاز ندارند. (خصوصیت `DataKeyNames` کنترل `GridView` وضعیتی را مشخص می کند که باید بین دو `PostBack` نگه داری شود، اما این مقادیر در وضعیت کنترل `(control state)` نگه داری می شوند، که متأثر از خصوصیت `ViewStateMode` نیست).

کد ساختیافته راهنمای صفحه² و کنترل `Label` اکنون شبیه مثال زیر خواهد بود:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"
CodeBehind="Instructors.aspx.cs" Inherits="ContosoUniversity.Instructors" %>
...
<asp:Label ID="ErrorMessageLabel" runat="server" Text="" Visible="false"
ViewStateMode="Disabled">
</asp:Label>
...
```

تغییرات زیر را اعمال کنید:

- `ViewStateMode="Disabled"` را به راهنمای صفحه اضافه کنید.
- `ViewStateMode="Disabled"` را از کنترل `Label` حذف کنید.

¹ session state
² page directive

کد ساختیافته اکنون شبیه مثال زیر خواهد بود:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true"
CodeBehind="Instructors.aspx.cs" Inherits="ContosoUniversity.Instructors"
ViewStateMode="Disabled" %>
...
<asp:Label ID="ErrorMessageLabel" runat="server" Text="" Visible="false">
</asp:Label>
```

اکنون وضعیت دید برای همه کنترل ها غیر فعال شده است. اگر بعدا کنترلی اضافه کردید که نیاز به استفاده از وضعیت دید داشت، همه کاری که باید انجام دهید این است که صفت `ViewStateMode="Enabled"` را در آن کنترل قرار دهید.

استفاده از گزینه NoTracking

هنگامی که یک شیء Context سطرهای دیتابیس را دریافت کرد و اشیای موجودیتی را که ارائه می دهد، ایجاد کرد، به طور پیش فرض آن اشیای موجودیت را با استفاده از مدیر وضعیت شیء، Track می کند. این پیگیری داده ها مانند یک Cache عمل می کند و هنگامی که که یک موجودیت را بروزرسانی می کنید، استفاده می شود. چون یک برنامه وب معمولا نمونه های شیء Context با عمر کوتاه دارند، پرس و جو ها اغلب داده هایی را بر می گردانند که نیاز به پیگیری (Track) ندارند، چون شیء context ای که آنها را می خواند قبل از اینکه هر یک از موجودیت های خوانده شده توسط او، دوباره استفاده یا بروزرسانی شوند، از بین می رود.

در Entity Framework، شما می توانید مشخص کنید که آیا شیء Context اشیای موجودیت را با استفاده از merge option پیگیری کند یا خیر. می توانید merge option را برای پرس و جوهای تکی یا برای مجموعه های موجودیت (entity sets) تنظیم کنید. اگر شما آن را برای یک مجموعه موجودیت تنظیم کنید، به این معنی است که شما default merge option را برای همه پرس و جو هایی که برای آن مجموعه موجودیت ایجاد شده اند، تنظیم کرده اید.

برای برنامه Contoso University، پیگیری برای مجموعه موجودیت هایی که شما از طریق مخزن به آنها دسترسی پیدا می کنید، نیاز نیست، بنابراین می توانید برای آن مجموعه موجودیت ها، هنگامی که شیء Context را در کلاس مخزن نمونه سازی

می کنید، merge option را برابر **NoTracking** قرار دهید. (توجه کنید در این بخش، تنظیم merge option اثر مورد توجهی روی کارایی برنامه ندارد. به نظر می رسد گزینه NoTracking فقط در سناریوهای high-data-volume برای قابل مشاهده کردن بهبود کارایی است.)

در پوشه DAL، فایل *SchoolRepository.cs* را باز کرده و متد سازنده ای را اضافه کنید که merge option را برای مجموعه موجودیت هایی که مخزن، به آنها دسترسی دارد تنظیم می کند:

```
public SchoolRepository()
{
    context.Department.MergeOption=MergeOption.NoTracking;
    context.InstructorNames.MergeOption=MergeOption.NoTracking;
    context.OfficeAssignment.MergeOption=MergeOption.NoTracking;
}
```

پردازش پرس و جوهای LINQ قبل از کامپایل

در بار نخستی که Entity Framework یک پرس و جوی SQL Entity را در طول عمر نمونهObjectContext داده شده، اجرا می کند، مقداری زمان می برد تا پرس و جو کامپایل شود. نتیجه کامپایل ذخیره شود، و این به این معنی است که اجراهای بعدی این پرس و جو بسیار سریع تر خواهد بود. پرس و جو های LINQ الگوی مشابهی را دنبال می کنند، به جز اینکه بعضی کارهایی که نیاز به کامپایل کوئری دارند هر بار که این کوئری اجرا می شود باید این کامپایل صورت گیرد. به عبارت دیگر، به طور پیش فرض، برای پرس و جو های LINQ، همه نتایج عملیات کامپایل، ذخیره نمی شوند.

اگر یک پرس و جوی LINQ دارید، که انتظار دارید مکرر در مدت عمر یک شی Context اجرا شود، می توانید کدی بنویسید که باعث شود همه نتایج کامپایل در اولین اجرای پرس و جوی LINQ ذخیره شود.

به طور مثال، شما می خواهید این کار را برای دو متد Get موجود در کلاس SchoolRepository انجام دهید، یکی از آنها هیچ پارامتری نمی گیرد (متد GetInstructorNames)، و دیگری نیاز به یک پارامتر دارد (متد GetDepartmentsByAdministrator). این متدها در حال حاضر واقعا نیازی به کامپایل شدن ندارند، چون پرس و جوهای LINQ نیستند:

```

public IEnumerable<InstructorName> GetInstructorNames()
{
    return context.InstructorNames.OrderBy("it.FullName").ToList();
}

public IEnumerable<Department> GetDepartmentsByAdministrator(Int32 administrator)
{
    return new ObjectQuery<Department>("SELECT VALUE d FROM Department as d",
    context, MergeOption.NoTracking).Include("Person").Where(d =>
d.Administrator == administrator).ToList();
}

```

به هر حال، برای اینکه بتوانید کوئری کامپایل شده را امتحان کنید، همان طور که کوئری های LINQ زیر نوشته شده اند، آنها را بنویسید:

```

public IEnumerable<InstructorName> GetInstructorNames()
{
    return(from i in context.InstructorNames orderby i.FullName select
i).ToList();
}

public IEnumerable<Department> GetDepartmentsByAdministrator(Int32 administrator)
{
    context.Department.MergeOption = MergeOption.NoTracking;
    return(from d in context.Department where d.Administrator==
administrator select d).ToList();
}

```

قبل از ادامه کار، شما می توانید کدهای این متدها را آنگونه که در بالا نشان داده شده تغییر دهید و برنامه را به منظور بررسی عملکرد این متدها اجرا کنید. اما ساختارهای زیر به طور صحیح به ایجاد نسخه های pre-compiled این متدها می پردازد.

یک فایل کلاس در پوشه DAL به نام *SchoolEntities.cs* ایجاد کرده و کد موجود را با کد زیر جایگزین نمایید:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Data.Objects;
namespace ContosoUniversity.DAL
{
    public partial class SchoolEntities
    {
        private static readonly Func<SchoolEntities,IQueryable<InstructorName>>
        compiledInstructorNamesQuery =
        CompiledQuery.Compile((SchoolEntities context)=>from i in
        context.InstructorNames orderby i.FullName select i);
        public IEnumerable<InstructorName>CompiledInstructorNamesQuery()
        {
            return compiledInstructorNamesQuery(this).ToList();
        }
        private static readonly Func<SchoolEntities,Int32,IQueryable<Department>>
        compiledDepartmentsByAdministratorQuery =
        CompiledQuery.Compile((SchoolEntities context,Int32 administrator)=>from d in
        context.Department.Include("Person")where d.Administrator==administrator
select d);
        public IEnumerable<Department>CompiledDepartmentsByAdministratorQuery(Int32
        administrator)
        {
            return compiledDepartmentsByAdministratorQuery(this, administrator).ToList();
        }
    }
}

```

این کد یک کلاس `partial` ایجاد می کند، که کلاس شیء `context` ای که به طور خودکار ایجاد شده را توسعه می دهد. این کلاس `partial` شامل دو پرس و جوی LINQ است که از متد `Compile` کلاس `CompiledQuery` استفاده می کنند. همچنین متدهایی را ایجاد می کند، که می توانید از آنها برای فراخوانی پرس و جوها استفاده کنید. این فایل را ذخیره کرده و ببینید.

سپس، در `SchoolRepository.cs` متدهای `GetInstructorNames` و `GetDepartmentsByAdministrator` موجود در مخزن را تغییر دهید، تا پرس و جوهای کامپایل شده را فراخوانی کنند:

```

public IEnumerable<InstructorName> GetInstructorNames()
{
    return context.CompiledInstructorNamesQuery();
}

public IEnumerable<Department> GetDepartmentsByAdministrator(Int32 administrator)
{
    return context.CompiledDepartmentsByAdministratorQuery(administrator);
}

```

صفحه *Departments.aspx* را، به منظور بررسی عملکرد آن اجرا کنید تا ببینید آیا همانند قبل کار می کند یا خیر. متد *GetInstructorNames* به منظور دستکاری لیست کشویی *administrator* فراخوانی شده است، و متد *GetDepartmentsByAdministrator* هنگامی که روی **Update** کلیک می کنید فراخوانی می شود تا بررسی کند که استادی مدیر بیش از یک دانشکده نیست.

The screenshot shows a web application interface titled "DEPARTMENTS". It features a search bar with the text "Enter any part of the name or leave blank to see all" and a search button. Below the search bar is a table with the following columns: Name, Budget, Start Date, and Administrator. The table contains one row for the "Economics" department with a budget of 200000.0000 and a start date of 9/1/2007. The Administrator column has a dropdown menu open, displaying a list of names: Harui, Roger; Abercrombie, Kim; Fakhouri, Fadi; Harui, Roger (highlighted); Kapoor, Candace; Serrano, Stacy; Stewart, Jasmine; Van Houten, Roger; Xu, Kristen; and Zheng, Roger. There are "Update" and "Cancel" links next to the table row.

شما در برنامه Contoso University دارای پرس و جوهای *pre-compiled* هستید فقط به خاطر اینکه ببینید چگونه کار می کنند نه به خاطر اینکه به طور قابل ملاحظه ای کارایی را بهبود می دهد. کوئری های *Pre-compiling* در LINQ درجه ای از پیچیدگی را به برنامه شما اضافه می کنند، بنابراین مطمئن شوید این کار را تنها برای کوئری هایی انجام می دهید که واقعا باعث ایجاد تنگناهای کارایی در برنامه شما می شوند.

آزمودن پرس و جوهای ارسالی به دیتابیس

هنگامی که به موضوع کارایی می پردازید، گاهی اوقات دانستن دقیق دستورات SQL ای که Entity Framework به دیتابیس ارسال می کند، مفید خواهد بود. اگر با شیء *IQueryable* کار می کنید، یک راه برای دانستن این دستورات استفاده از متد *ToTraceString* است.

در *SchoolRepository.cs* کد درون متد *GetDepartmentsByName* را مانند مثال زیر تغییر دهید:

```
public IEnumerable<Department> GetDepartmentsByName(string sortExpression, string
nameSearchString)
{
    ...
}
```


متغیر departments باید به نوع ObjectQuery تبدیل شود، فقط به خاطر اینکه متد Where در انتهای خط قبلی یک شیء IQueryable ایجاد می کند. بدون متد Where این تبدیل نوع، ضروری نخواهد بود.

روی خط return یک نقطه انفصال¹ ایجاد کنید، و به صفحه Departments.aspx در debugger برگردید. هنگامی که به نقطه انفصال برخورد کردید، متغیر commandText در پنجره Locals را بررسی کنید و از text visualize (ذره بین در ستون Value) برای نمایش مقدار آن در پنجره Text Visualizer استفاده کنید. می توانید تمام دستور SQL ای که نتیجه این کد است را ببینید:

¹ breakpoint

Text Visualizer

Expression: commandText

Value:

```

SELECT
[Project1].[PersonID] AS [PersonID],
[Project1].[DepartmentID] AS [DepartmentID],
[Project1].[Name] AS [Name],
[Project1].[Budget] AS [Budget],
[Project1].[StartDate] AS [StartDate],
[Project1].[Administrator] AS [Administrator],
[Project1].[C1] AS [C1],
[Project1].[LastName] AS [LastName],
[Project1].[FirstName] AS [FirstName],
[Project1].[C2] AS [C2],
[Project1].[C3] AS [C3],
[Project1].[C4] AS [C4],
[Project1].[CourseID] AS [CourseID],
[Project1].[Title] AS [Title],
[Project1].[Credits] AS [Credits],
[Project1].[DepartmentID1] AS [DepartmentID1]
FROM ( SELECT
[Extent1].[DepartmentID] AS [DepartmentID],
[Extent1].[Name] AS [Name],
[Extent1].[Budget] AS [Budget],
[Extent1].[StartDate] AS [StartDate],
[Extent1].[Administrator] AS [Administrator],
[Extent2].[PersonID] AS [PersonID],
[Extent2].[LastName] AS [LastName],
[Extent2].[FirstName] AS [FirstName],
CASE WHEN ([Extent2].[PersonID] IS NULL) THEN CAST(NULL AS varchar(1)) WHE
CASE WHEN ([Extent2].[PersonID] IS NULL) THEN CAST(NULL AS datetime2) WHEN
CASE WHEN ([Extent2].[PersonID] IS NULL) THEN CAST(NULL AS datetime2) WHEN
[Extent3].[CourseID] AS [CourseID],
[Extent3].[Title] AS [Title],
[Extent3].[Credits] AS [Credits],
[Extent3].[DepartmentID] AS [DepartmentID1],
CASE WHEN ([Extent3].[CourseID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END
FROM [dbo].[Department] AS [Extent1]
LEFT OUTER JOIN [dbo].[Person] AS [Extent2] ON ([Extent2].[HireDate] IS N
LEFT OUTER JOIN [dbo].[Course] AS [Extent3] ON [Extent1].[DepartmentID] =
) AS [Project1]
ORDER BY [Project1].[Name] ASC, [Project1].[PersonID] ASC, [Project1].[DepartmentI

```

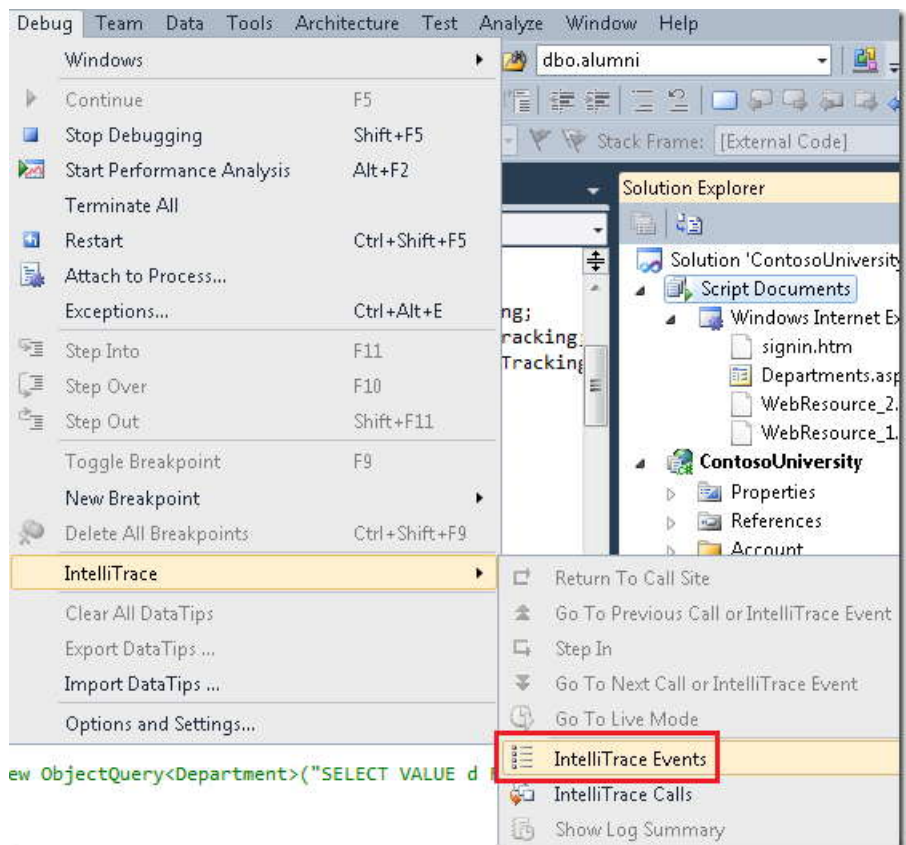
Wrap

Close Help

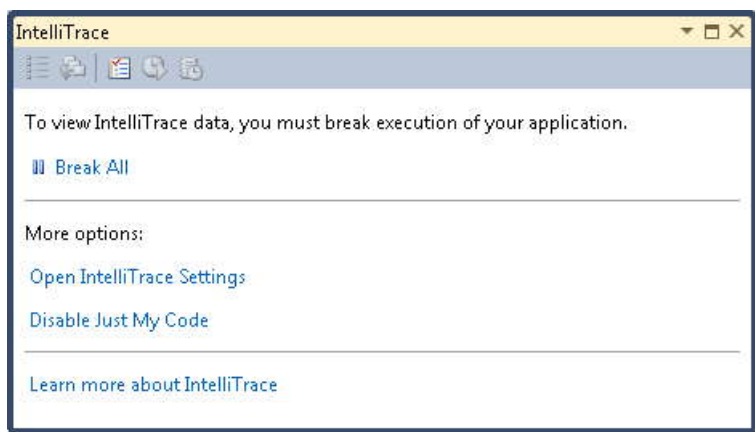
راه دیگر: ویژگی IntelliTrace در Visual Studio Ultimate شیوه ای را برای دیدن دستورات SQL ای که توسط Entity Framework ایجاد شده، ارائه می دهد که شما نیازی به تغییر کد یا ایجاد نقطه انفصال نخواهید داشت.

توجه: تنها در صورت داشتن Visual Studio Ultimate می توانید رویه زیر را دنبال کنید.

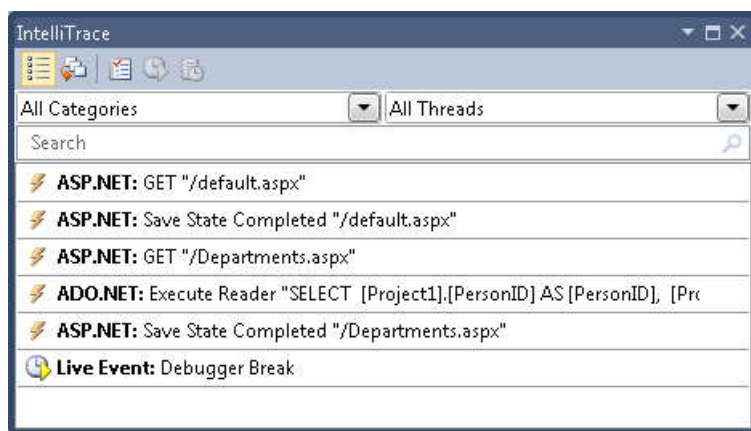
کد اصلی متد **GetDepartmentsByName** را برگردانید و سپس صفحه **Departments.aspx** را در debugger اجرا کنید. در Visual Studio منوی **Debug**، سپس **IntelliTrace** و سپس **IntelliTrace Events** را انتخاب کنید.



در پنجره IntelliTrace، روی **Break All** کلیک کنید.



پنجره IntelliTrace لیستی از رخدادهای اخیر را نشان می دهد:



روی **ADO.NET** کلیک کنید تا گسترش یابد و متن دستور را به شما نشان دهد:

```

ADO.NET: Execute Reader "SELECT [Project1].[PersonID] AS [PersonID], [Project1].[DepartmentID] AS [De
The command text "SELECT
[Project1].[PersonID] AS [PersonID],
[Project1].[DepartmentID] AS [DepartmentID],
[Project1].[Name] AS [Name],
[Project1].[Budget] AS [Budget],
[Project1].[StartDate] AS [StartDate],
[Project1].[Administrator] AS [Administrator],
[Project1].[C1] AS [C1],
[Project1].[LastName] AS [LastName],
[Project1].[FirstName] AS [FirstName],
[Project1].[C2] AS [C2],
[Project1].[C3] AS [C3],
[Project1].[C4] AS [C4],
[Project1].[CourseID] AS [CourseID],
[Project1].[Title] AS [Title],
[Project1].[Credits] AS [Credits],
[Project1].[DepartmentID1] AS [DepartmentID1]
FROM ( SELECT
    [Extent1].[DepartmentID] AS [DepartmentID],
    [Extent1].[Name] AS [Name],
    [Extent1].[Budget] AS [Budget],
    [Extent1].[StartDate] AS [StartDate],
    [Extent1].[Administrator] AS [Administrator],
    [Extent2].[PersonID] AS [PersonID],
    [Extent2].[LastName] AS [LastName],
    [Extent2].[FirstName] AS [FirstName],
    CASE WHEN ([Extent2].[PersonID] IS NULL) THEN CAST(NULL AS varchar(1)) WHEN ([Extent2].
[HireDate] IS NOT NULL) THEN '2X0X' ELSE '2X1X' END AS [C1],
    CASE WHEN ([Extent2].[PersonID] IS NULL) THEN CAST(NULL AS datetime2) WHEN ([Extent2].
[HireDate] IS NOT NULL) THEN [Extent2].[HireDate] END AS [C2],
    CASE WHEN ([Extent2].[PersonID] IS NULL) THEN CAST(NULL AS datetime2) WHEN ([Extent2].
[HireDate] IS NOT NULL) THEN CAST(NULL AS datetime2) ELSE [Extent2].[EnrollmentDate] END AS [C3],
    [Extent3].[CourseID] AS [CourseID],
    [Extent3].[Title] AS [Title],
    [Extent3].[Credits] AS [Credits],
    [Extent3].[DepartmentID] AS [DepartmentID1],
    CASE WHEN ([Extent3].[CourseID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C4]
FROM [dbo].[Department] AS [Extent1]
LEFT OUTER JOIN [dbo].[Person] AS [Extent2] ON (([Extent2].[HireDate] IS NOT NULL) OR
([Extent2].[EnrollmentDate] IS NOT NULL)) AND ([Extent1].[Administrator] = [Extent2].[PersonID])
LEFT OUTER JOIN [dbo].[Course] AS [Extent3] ON [Extent1].[DepartmentID] = [Extent3].
[DepartmentID1]

```

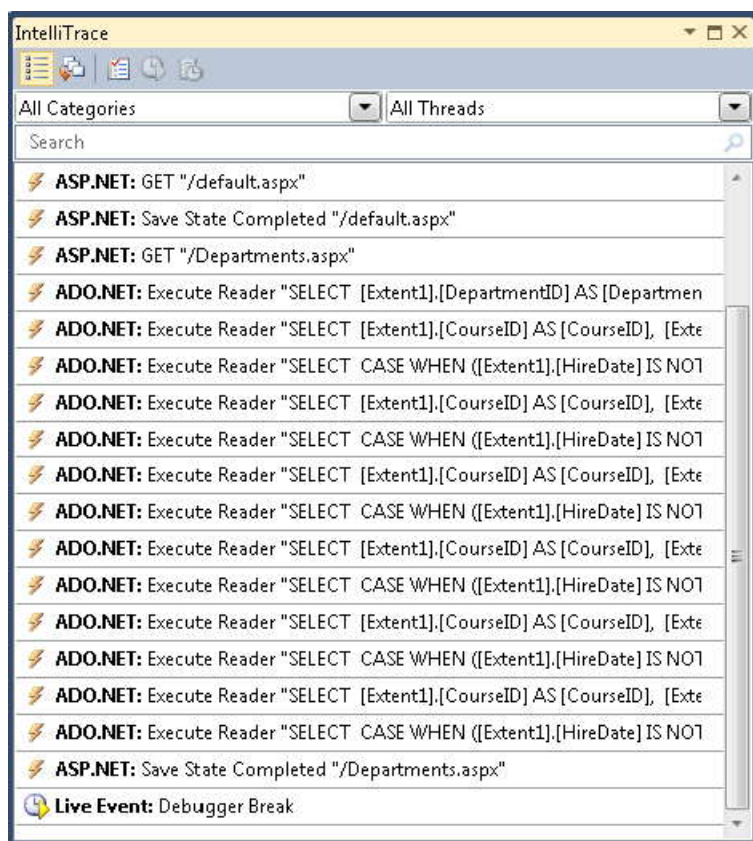
می توانید تمام متن دستور را از پنجره **Locals** به کلیپ برد کپی کنید.

فرض کنید شما با یک دیتابیس با جدول ها، رابطه ها، و ستون های بیشتر از دیتابیس ساده **School** کار می کردید. ممکن بود یک پرس و جویی را بیابید که تمام اطلاعاتی که شما نیاز دارید را در یک دستور **Select** متشکل از چند عبارت **Join** جمع کند، که برای موثر کار کردن پیچیده می شود. در این صورت به منظور ساده سازی پرس و جو، می توانید به جای **eager loading** از **explicit loading** استفاده کنید.

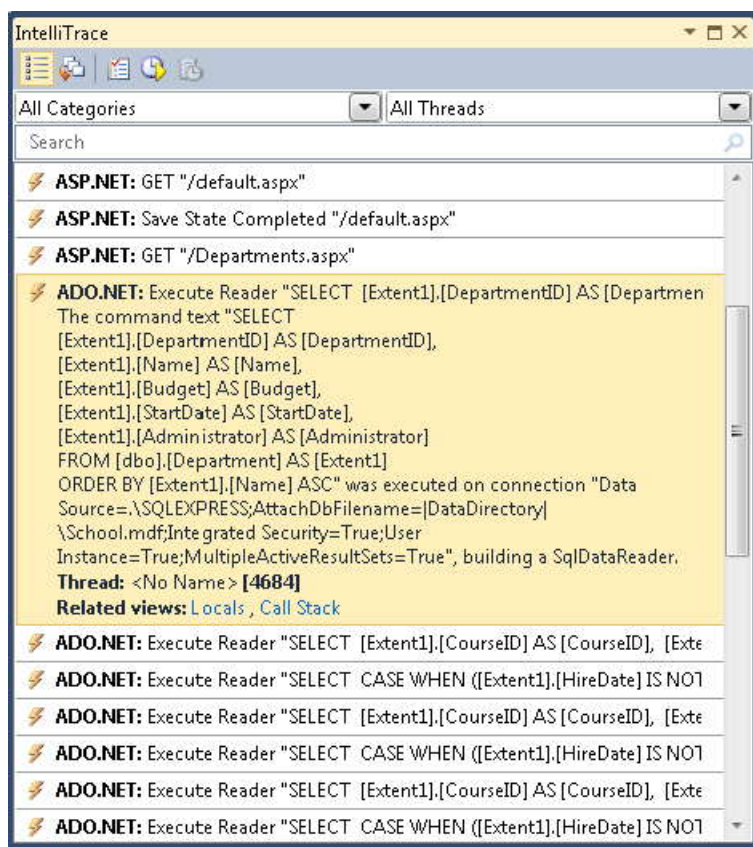
برای مثال، تغییراتی در متد `GetDepartmentsByName` کلاس `SchoolRepository.cs` ایجاد کنید. در حال حاضر در این متد شما یک شیء پرس و جو دارید که دارای متدهای `Include` برای خصوصیت های `Person navigation` و `navigation` `Course` است. دستور `return` را با کدی که `explicit loading` را اجرا می کند جایگزین کنید، همان طور که در مثال زیر نشان داده شده:

```
public IEnumerable<Department> GetDepartmentsByName(string sortExpression, string
    nameSearchString)
{
    ...
    var departments = new ObjectQuery<Department>("SELECT VALUE d FROM
    Department AS d",
    context).OrderBy("it." + sortExpression).Where(d =>
    d.Name.Contains(nameSearchString)).ToList();
    foreach (Department d in departments)
    {
        d.Course.Load();
        d.PersonReference.Load();
    }
    return departments;
}
```

صفحه `Departments.aspx` را در `debugger` اجرا کنید، و پنجره `IntelliTrace` را دوباره همانند قبل بررسی کنید. در جایی که قبلا یک کوئری مشاهده کردید، حالا، یک رشته طولانی کوئری خواهید دید.



روی اولین خط **ADO.NET** کلیک کنید، تا ببینید برای کوئری پیچیده ای که اخیراً مشاهده کردید چه اتفاقی افتاده است.



پرس و جو از دیتابیس، به صورت یک پرس و جوی Select بدون عبارت Join آمده است، اما با پرس و جوهای جداگانه ای که course های مرتبط و یک مدیر را بر می گرداند، دنبال شده است، که برای هر دانشکده برگشتی توسط پرس و جوی اصلی، از دو مجموعه پرس و جو، استفاده شده است.

توجه: اگر lazy loading را فعال بگذارید، الگویی که اینجا مشاهده می کنید، با پرس و جوی یکسانی که بارها تکرار شده است، ممکن است نتیجه lazy loading باشد. الگویی که معمولاً برای اجتناب از این امر نیاز دارید، lazy-loading دادهای مرتبط، برای هر سطر جدول اصلی است. مگر اینکه به این نتیجه رسیده باشید که یک پرس و جوی join تکی، بسیار پیچیده تر از آن است که کارآمد باشد، بنابراین معمولاً در این گونه موارد می توانید کارایی را با تغییر دادن پرس و جوی اصلی برای استفاده از eager loading، بهبود ببخشید.

پیش تولید کردن دیدها^۱

زمانی که `ObjectContext` اولین بار در قلمروی یک برنامه ایجاد می شود، `Entity Framework` یک مجموعه از کلاس ها را که برای دسترسی به داده استفاده می کند، تولید می کند. این کلاس ها `View` نامیده می شوند، و اگر شما یک مدل داده بسیار بزرگ داشته باشید، تولید این `view` ها می تواند در پاسخ وب سایت، در اولین درخواست برای یک صفحه بعد از اینکه قلمروی برنامه مقاردهی اولیه شد، تاخیر ایجاد کند. شما می توانید تاخیر درخواست اول را با ایجاد دیدهایی در زمان کامپایل به جای زمان اجرا کاهش دهید.

توجه: اگر برنامه شما مدل داده بسیار بزرگی ندارد، یا اگر مدل داده بزرگی دارد اما شما در مورد مشکل کارایی که بسیاری از درخواست های صفحه اول را بعد از بازسازی `IIS`، تحت تاثیر قرار می دهد، نگرانی ندارید، می توانید، این بخش را رها کنید. ایجاد دید در هر زمانی که یک شیء `ObjectContext` را نمونه سازی می کنید، صورت نمی گیرد، چون دیدها در قلمروی برنامه نگه داری^۲ می شوند. بنابراین، به غیر از مواقعی که مکرر برنامه تان را در `IIS` بازسازی می کنید، تعداد بسیار کمی از درخواست های صفحه از دیدهای پیش تولید شده^۳ سود می برند.

شما می توانید دیدها را با استفاده از ابزار خط دستور `EdmGen.exe` یا استفاده از یک الگوی `Text Template Transformation Toolkit (T4)` پیش تولید کنید. در این بخش از یک الگوی `T4` استفاده خواهید کرد.

در پوشه `DAL`، با استفاده از الگوی `Text Template` (زیر مجموعه گره `General` در لیست `Installed Templates`) یک فایل اضافه کنید، و نام آن را `SchoolModel.Views.tt` قرار دهید. کد موجود را با کد زیر جایگزین نمایید:

¹ Pre-Generating Views

² Cache

³ pre-generated

```

<#
/*****
Copyright (c) Microsoft Corporation. All rights reserved.
THIS CODE IS PROVIDED *AS IS* WITHOUT WARRANTY OF
ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY
IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR
PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.
*****/
#>
<#
//
// TITLE: T4 template to generate views for an EDMX file in a C# project
//
// DESCRIPTION:
// This is a T4 template to generate views in C# for an EDMX file in C# projects.
// The generated views are automatically compiled into the project's output
assembly.
//
// This template follows a simple file naming convention to determine the EDMX
file to process:
// - It assumes that [edmx-file-name].Views.tt will process and generate views
for [edmx-file-name].EDMX
// - The views are generated in the code behind file [edmx-file-name].Views.cs
//
// USAGE:
// Do the following to generate views for an EDMX file (e.g. Model1.edmx) in a C#
project
// 1. In Solution Explorer, right-click the project node and choose
"Add...Existing...Item" from the context menu
// 2. Browse to and choose this .tt file to include it in the project
// 3. Ensure this .tt file is in the same directory as the EDMX file to process
// 4. In Solution Explorer, rename this .tt file to the form [edmx-filename].
Views.tt (e.g. Model1.Views.tt)
// 5. In Solution Explorer, right-click Model1.Views.tt and choose "Run Custom
Tool" to generate the views
// 6. The views are generated in the code behind file Model1.Views.cs
//
// TIPS:
// If you have multiple EDMX files in your project then make as many copies of
this .tt file and rename appropriately
// to pair each with each EDMX file.
//
// To generate views for all EDMX files in the solution, click the "Transform All
Templates" button in the Solution Explorer toolbar
// (its the rightmost button in the toolbar)
//
#>
<#
//
// T4 template code follows
//

```

```

#>
<#@ template language="C#" hostspecific="true"#>
<#@ include file="EF.Utility.CS.ttinclude"#>
<#@ output extension=".cs" #>
<#
// Find EDMX file to process: Model1.Views.tt generates views for Model1.EDMX
string edmxFilename =
Path.GetFileNameWithoutExtension(this.Host.TemplateFile).ToLowerInvariant().Replace(
.views", "") + ".edmx";
string edmxFilenamePath = Path.Combine(Path.GetDirectoryName(this.Host.TemplateFile),
edmxFilename);
if (File.Exists(edmxFilenamePath))
{
// Call helper class to generate pre-compiled views and write to output
this.WriteLine(GenerateViews(edmxFilenamePath));
}
else
{
this.Error(String.Format("No views were generated. Cannot find file {0}.
Ensure the project has an EDMX file and the file name of the .tt file is of the form
[edmxfilename].Views.tt", edmxFilenamePath));
}
// All done!
#>
<#+
private String GenerateViews(string edmxFilenamePath)
{
MetadataLoader loader = new MetadataLoader(this);
MetadataWorkspace workspace;
if(!loader.TryLoadAllMetadata(edmxFilenamePath, out workspace))
{
this.Error("Error in the metadata");
return String.Empty;
}
String generatedViews = String.Empty;
try
{
using (StreamWriter writer = new StreamWriter(new MemoryStream()))
{
StorageMappingItemCollection mappingItems =
(StorageMappingItemCollection)workspace.GetItemCollection(DataSpace.CSSpace);
// Initialize the view generator to generate views in C#
EntityViewGenerator viewGenerator = new EntityViewGenerator();
viewGenerator.LanguageOption = LanguageOption.GenerateCSharpCode;
IList<EdmSchemaError> errors =
viewGenerator.GenerateViews(mappingItems, writer);
foreach (EdmSchemaError e in errors)
{
// log error
this.Error(e.Message);
}
}
}
}
}

```

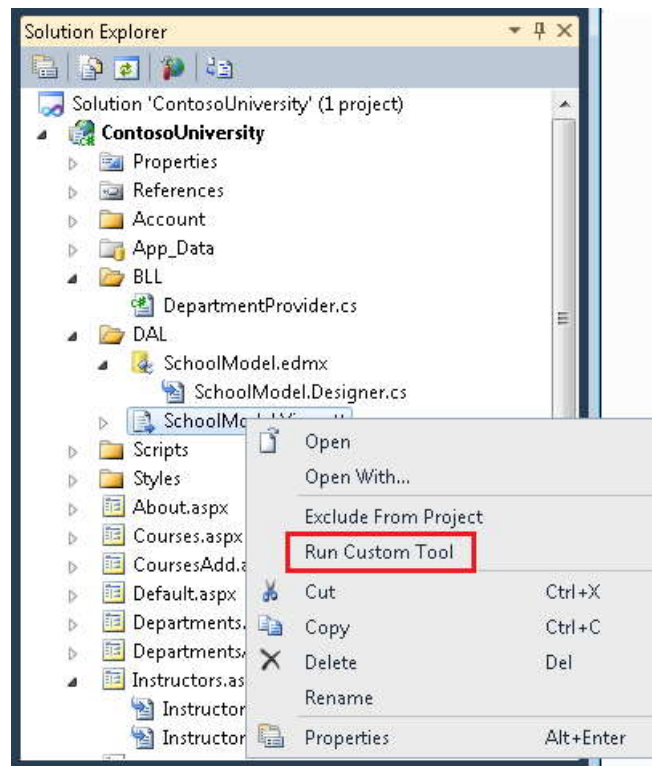
```

}
MemoryStream memStream = writer.BaseStream as MemoryStream;
generatedViews = Encoding.UTF8.GetString(memStream.ToArray());
}
}
catch (Exception ex)
{
// log error
this.Error(ex.ToString());
}
return generatedViews;
}
#>

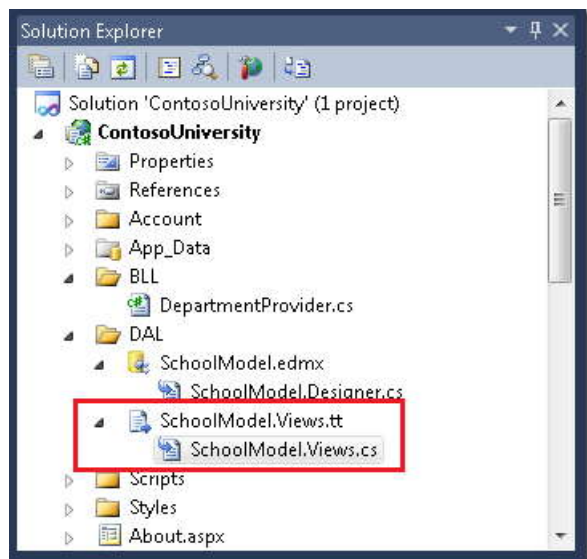
```

این کد، دیدهایی را برای فایل `edmx` که در پوشه محل قرارگیری فایل الگو قرار دارد، ایجاد می کند، و نام آن مشابه نام فایل الگو است. به طور مثال اگر نام فایل الگوی شما `SchoolModel.Views.tt` است، دنبال فایل با نام `SchoolModel.edmx` خواهد گشت.

فایل را ذخیره کرده و در **Solution Explorer** روی آن فایل، راست کلیک کرده و **Run Custom Tool** را انتخاب کنید.



Visual Studio فایل کدی را تولید می کند که آن کد، دیدها را ایجاد می کند. که بر اساس الگو به نام **Run Custom** *SchoolModel.Views.cs* خواهد بود. (شما ممکن است ملاحظه کرده باشید که فایل کد حتی قبل از اینکه **Tool** را انتخاب کنید، به محض ذخیره کردن فایل الگو، ایجاد شده است.)



اکنون می توانید برنامه را اجرا کرده و بررسی کنید که آیا مانند گذشته کار می کند یا خیر. برای اطلاعات بیشتر در مورد دیدهای **pre-generated**، منابع زیر را مشاهده کنید:

- [How to: Pre-Generate Views to Improve Query Performance](#) در سایت MSDN
 - چگونگی استفاده از ابزار خط فرمان **EdmGen.exe** برای **pre-generate** دیدها را شرح می دهد.
 - [Isolating Performance with Precompiled/Pre-generated Views in the Entity Framework 4](#) در بلاگ Windows Server AppFabric Customer Advisory Team
- در این بخش بحث در مورد بهبود کارایی در برنامه های کاربردی وب ASP.NET که از Entity Framework استفاده می کنند به پایان رسید. برای اطلاعات بیشتر، منابع زیر را مشاهده کنید:
- [Performance Considerations \(Entity Framework\)](#) در وب سایت MSDN

[Performance-related posts on the Entity Framework Team blog](#) •

[EF Merge Options and Compiled Queries](#) •

در مورد رفتارهای ناخواسته پرس و جوهای کامپایل شده و merge option ها مانند **NoTracking**. اگر در نظر دارید از پرس و جوهای کامپایل شده استفاده کنید، یا تنظیمات merge option را در برنامه خود تغییر دهید، ابتدا این مطلب را بخوانید.

[Entity Framework-related posts in the Data and Modeling Customer Advisory Team blog.](#) •

شامل مطالبی در مورد پرس و جوهای کامپایل شده و استفاده از Visual Studio 2010 Profiler برای مشخص کردن موضوعات کارایی

[Entity Framework forum thread with advice on improving performance of highly complex queries](#) •

[ASP.NET State Management Recommendations](#) •

[Using the Entity Framework and the ObjectDataSource: Custom Paging](#) •

ارائه مطالبی بر پایه برنامه ContosoUniversity ایجاد شده در این کتاب، و توضیح در مورد چگونگی صفحه بندی در صفحه *Departments.aspx*.

بخش بعد به مرور بعضی از ویژگی های مهم اضافه شده به Entity Framework 4 می پردازد.

چه چیزهای جدیدی در Entity Framework 4 وجود دارد

در بخش قبل روش هایی را برای بالابردن کارایی برنامه وبی که از Entity Framework استفاده می کنند، مشاهده کردید. این بخش مروری بر ویژگی های جدید در نسخه 4 Entity Framework دارد، و منابعی را برای کامل کردن بحث در مورد ویژگی های جدید ارائه می دهد. ویژگی های برجسته این بخش شامل موارد زیر است:

- وابستگی های کلید خارجی
- اجرای دستورات SQL تعریف شده توسط کاربر.
- توسعه ابتدا-مدل (model-first)
- پشتیبانی از POCO

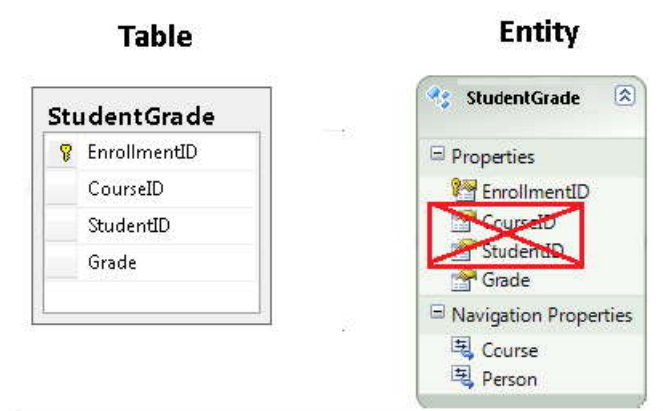
به علاوه، در این بخش، توسعه ابتدا - کد (code-first)، به طور خلاصه را شرح داده خواهد شد.

برای شروع این بخش، Visual Studio و سپس برنامه وب Contoso University که در بخش قبل با آن کار می کردید را باز کنید.

وابستگی های کلید خارجی^۱

نسخه Entity Framework 3.5 دارای خصوصیت های navigation است، اما دارای خصوصیت های کلید خارجی در مدل داده نیست. برای مثال، ستون های CourseID و StudentID جدول StudentGrade از موجودیت StudentGrade ممکن است حذف شوند.

^۱ Foreign-Key Associations



علت این کار این بود، که اکیدا گفته می شود، کلید های خارجی اجزای پیاده سازی فیزیکی هستند و به مدل داده انتزاعی تعلق ندارند. هر چند، در عمل، کار با موجودیت ها در کد، هنگامی که به طور مستقیم به کلیدهای خارجی دسترسی پیدا می کنید، اغلب ساده تر است.

به عنوان مثال برای دانستن اینکه چگونه کلیدهای خارجی در مدل داده می توانند به ساده سازی کد شما کمک کنند، در نظر داشته باشید که بدون آنها چگونه مجبور به کد نویسی صفحه *DepartmentsAdd.aspx* بودید. در موجودیت **Department** خصوصیت **Administrator** کلید خارجی است که برابر **PersonID** در موجودیت **Person** است. به منظور برقراری وابستگی بین یک دانشکده جدید و مدیر آن دانشکده، همه آنچه باید انجام می دادید تنظیم مقدار خصوصیت **PersonID** در اداره کننده رخداد **ItemInserting** کنترل **databound** (منظور کنترلی که داده ها به آن مقید می شوند. در اینجا **DepartmentsDetailsView**) بود:

```
protected void DepartmentsDetailsView_ItemInserting(object sender, DetailsViewInsertEventArgs e)
{
    e.Values["Administrator"]= administratorsDropDownList.SelectedValue;
}
```

بدون کلید خارجی در مدل داده، به منظور گرفتن ارجاعی به خود موجودیت، قبل از اینکه آن موجودیت به مجموعه موجودیت اضافه شود، شما باید رخداد **Inserting** کنترل **data source** را به جای رخداد **ItemInserting** کنترل **databound** اداره می کردید. و هنگامی که آن ارجاع را دریافت کردید، باید با استفاده از کدی مانند مثال زیر وابستگی را برقرار کنید:


```

departmentEntityToBeInserted.PersonReference.EntityKey=newSystem.Data.EntityKey(
"SchoolEntities.Department", "PersonID", Convert.ToInt32(administratorsDropDownList.
Selecte
dValue));

departmentEntityToBeInserted.Person= context.People.Single(p
=>p.PersonID==Convert.ToInt32(administratorsDropDownList.SelectedValue));

```

همان طور که در [Entity Framework team's blog post on Foreign Key associations](#) مشاهده می کنید، موارد دیگری وجود دارند که تفاوت در پیچیدگی کد آنها بسیار بیشتر است. برای برآورده کردن نیازهای افرادی که ترجیح می دهند جزئیات را در مدل داده انتزاعی پیاده سازی کنند. و به خاطر ساده تر شدن کد، Entity Framework گزینه ای شامل کلیدهای خارجی در مدل داده را ارائه می دهد.

در مجموعه اصطلاحات Entity Framework، اگر شما کلیدهای خارجی را در مدل داده بگنجانید، از وابستگی کلیدهای خارجی^۱ استفاده می کنید، و اگر کلیدهای خارجی گنجانده نشود، شما از وابستگی های مستقل^۲ استفاده می کنید.

اجرای دستورات SQL تعریف شده توسط کاربر^۳

در نسخه قبل Entity Framework، شیوه آسانی برای ایجاد دستورات SQL تعریف شده توسط کاربر، در هنگام شروع و اجرا وجود نداشت. چه Entity Framework به طور خودکار دستورات SQL را برای شما تولید کرده باشد، یا شما مجبور بوده باشید به ایجاد یک رویه ذخیره شده و وارد کردن آن به عنوان یک تابع. نسخه ۴ متدهای `ExecuteStoreQuery` و `ExecuteStoreCommand` را به کلاس `ObjectContext` اضافه کرده است، تا ساده تر بتوانید هر پرس و جویی را به طور مستقیم به دیتابیس ارسال کنید.

فرض کنید مدیر دانشگاه Contoso می خواهد قادر به اجرای تغییرات حجیم در دیتابیس باشد بدون اینکه مجبور باشد از طریق فرایند ایجاد `Stored procedure`، و وارد کردن آن به داخل مدل داده این کار را انجام دهد. اولین درخواست آنها برای صفحه ای است که به آنها اجازه تغییر تعدادی از `credit` ها برای همه `course` ها را در دیتابیس بدهد. در صفحه وب، آنها می خواهند قادر به وارد کردن عددی برای استفاده به منظور دستکاری کردن مقدار ستون `Credits` هر سطر `Course` باشند.

¹ foreign key associations

² independent associations

³ User-Defined SQL Commands

صفحه جدیدی ایجاد کنید که از *Site.Master* استفاده کرده و آن را *UpdateCredits.aspx* بنامید. سپس کد ساختیافته زیر را به کنترل *Content* به نام *Content2* اضافه کنید:

```
<h2>Update Credits</h2>
Enter the number to multiply the current number of credits by:
<asp:TextBox ID="CreditsMultiplierTextBox" runat="server"></asp:TextBox>
<br/><br/>
<asp:Button ID="ExecuteButton" runat="server" Text="Execute"
OnClick="ExecuteButton_Click"/><br/><br/>
Rows affected:
<asp:Label ID="RowsAffectedLabel" runat="server" Text="0" ViewStateMode="
Disabled"></asp:Label><br/><br/>
```

این کد ساختیافته یک کنترل *TextBox* ایجاد خواهد کرد که کاربر می تواند مقدار مضرب را درون آن وارد کند، همچنین یک کنترل *Button* به منظور اجرای دستوارت کاربر، و یک کنترل *Label* به منظور نمایش سطرهایی که مورد اثر واقع شده اند.

UpdateCredits.aspx.cs را باز کرده و دستور *using* و اداره کننده رخداد *Click* دکمه را اضافه کنید:

```
using ContosoUniversity.DAL;
...

protected void ExecuteButton_Click(object sender, EventArgs e)
{
    using (SchoolEntities context = new SchoolEntities())
    {
        RowsAffectedLabel.Text = context.ExecuteStoreCommand(
            "UPDATE Course SET Credits = Credits * {0}",
            CreditsMultiplierTextBox.Text).ToString();
    }
}
```

این کد، دستور *SQL Update* را با استفاده از مقدار موجود در جعبه متن اجرا می کند، و از *Label* برای نمایش تعداد سطرهای مورد اثر واقع شده استفاده می کند. قبل از اینکه صفحه را اجرا کنید، صفحه *Courses.aspx* را برای گرفتن یک پیش تصویر از داده ها اجرا کنید.

COURSES BY DEPARTMENT
 Select a Department

ID	Title	Credits
2021	Composition 3	3
2030	Poetry	2
2042	Literature	4

صفحه *UpdateCredits.aspx* را اجرا کرده و مضرب ۱۰ را وارد کرده و روی **Execute** کلیک کنید.

UPDATE CREDITS
 Enter the number to multiply the current number of credits by:

Rows affected: 11

صفحه *Courses.aspx* را برای دیدن داده های تغییر یافته اجرا کنید.

COURSES BY DEPARTMENT
 Select a Department

ID	Title	Credits
2021	Composition 30	30
2030	Poetry	20
2042	Literature	40

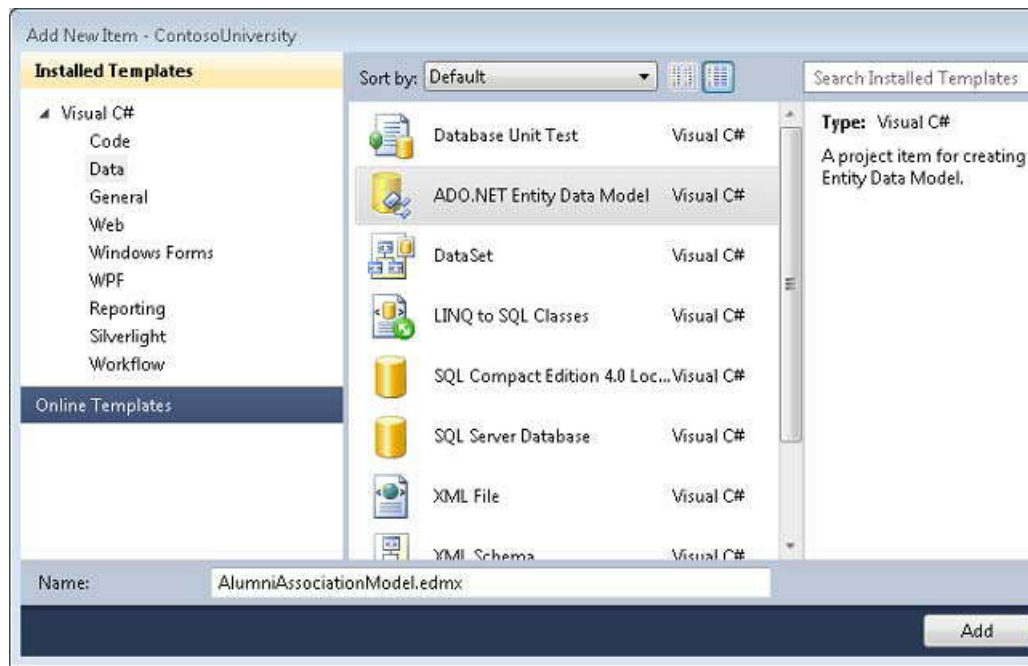
(اگر می خواهید عدد *credits* را به مقدار اصلی بر گردانید، در *UpdateCredits.aspx.cs* عبارت *Credits*{0}* را به *Credits/{0}* تغییر دهید و صفحه را دوباره اجرا کنید، عدد ۱۰ را به عنوان مقسوم علیه وارد کنید).

برای دیدن اطلاعات بیشتر در مورد اجرای پرس و جوهایی که در کد تعریف می کنید، به [How to: Directly Execute](#) مراجعه کنید. [Commands Against the Data Source](#) مراجعه کنید.

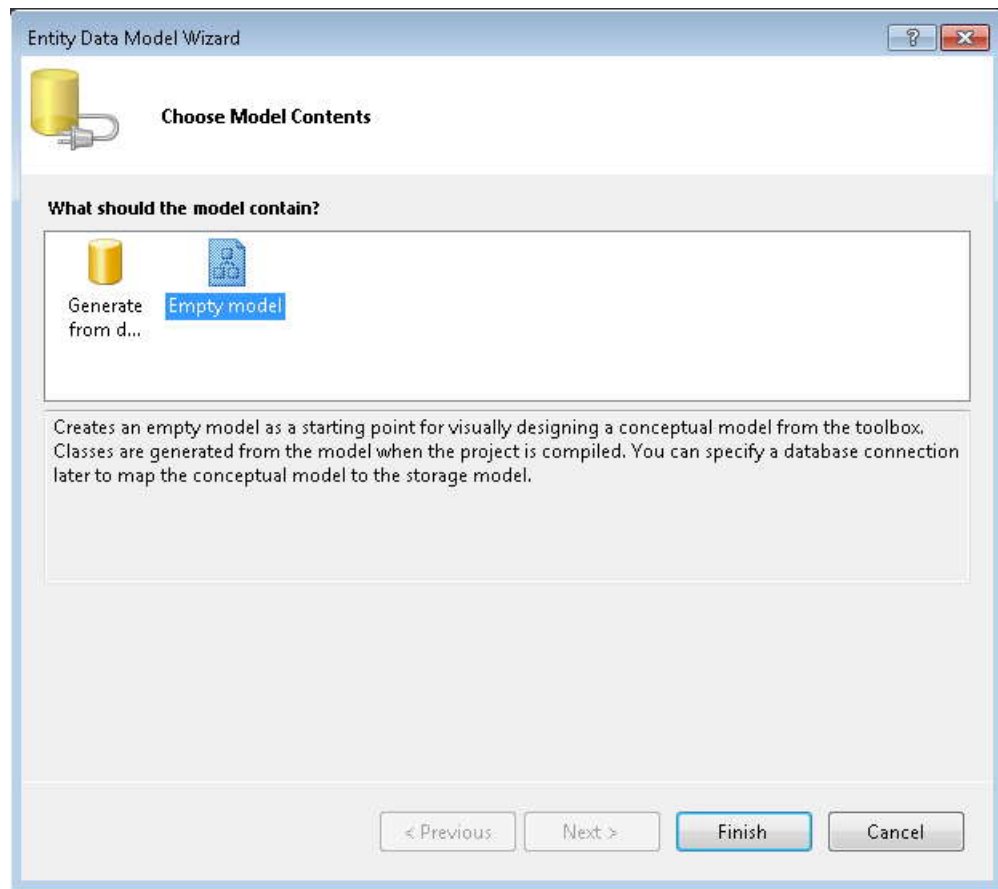
در این گام به گام، شما ابتدا دیتابیس را ایجاد کردید و سپس مدل داده را بر اساس ساختار دیتابیس تولید کردید. در Entity Framework 4 می توانید به جای روش قبل ابتدا با مدل داده شروع کنید و دیتابیس را بر اساس ساختار مدل داده تولید کنید. اگر در حال ساخت برنامه ای هستید که دیتابیس آن فعلا موجود نیست، روش ابتدا-مدل، به شما امکان می دهد تا موجودیت ها و روابط را ایجاد کنید. که باعث درک انتزاعی برای برنامه می شود، بدون نگرانی در مورد جزئیات پیاده سازی فیزیکی. (این روش تنها در مراحل ابتدایی توسعه پا برجا است، به هر حال، دیتابیس به تدریج ایجاد خواهد شد و داده های تولیدی را در درونش خواهد داشت، و دوباره سازی دیتابیس از روی مدل داده بیش از این عملی نخواهد بود. در آن نقطه شما به روش database-first خواهید برگشت).

در این قسمت، یک مدل داده ساده ایجاد خواهید کرد و دیتابیس را از روی آن تولید خواهید کرد.

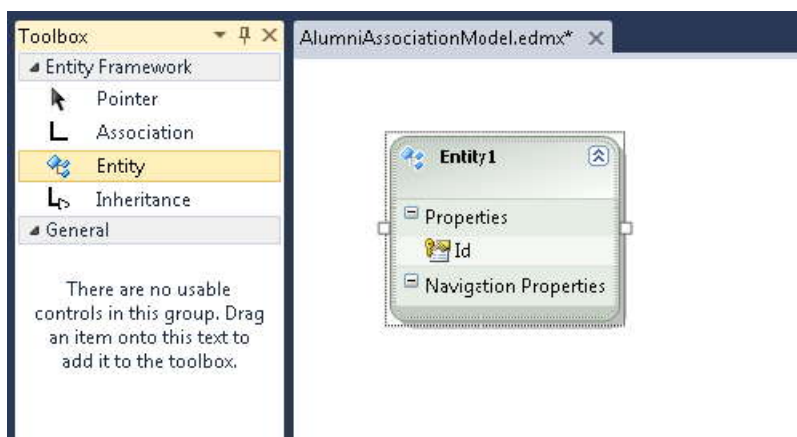
در **Solution Explorer**، روی پوشه **DAL** راست کلیک کرده و **Add New Item** را انتخاب کنید. در کادر محاوره ای **Add New Item**، از زیر مجموعه **Installed Templates** گزینه **Data** و سپس الگوی **ADO.NET Entity Data Model** را انتخاب کنید. نام فایل جدید را **AlumniAssociationModel.edmx** بگذارید و روی **Add** کلیک کنید.



این کار باعث بالا آمدن Entity Data Model Wizard می شود. در مرحله **Choose Model Contents**، گزینه **Empty Model** را انتخاب کرده و سپس روی **Finish** کلیک کنید.

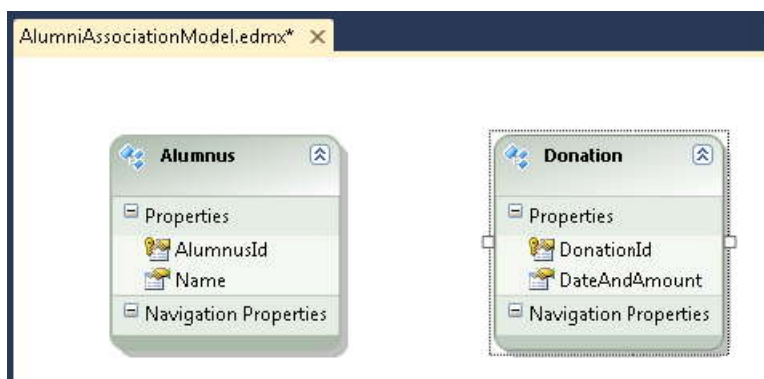


Entity Data Model Designer با یک ناحیه طراحی خالی باز می شود. یک آیتم **Entity** از **Toolbox** را کشیده و روی این سطح بیندازید.

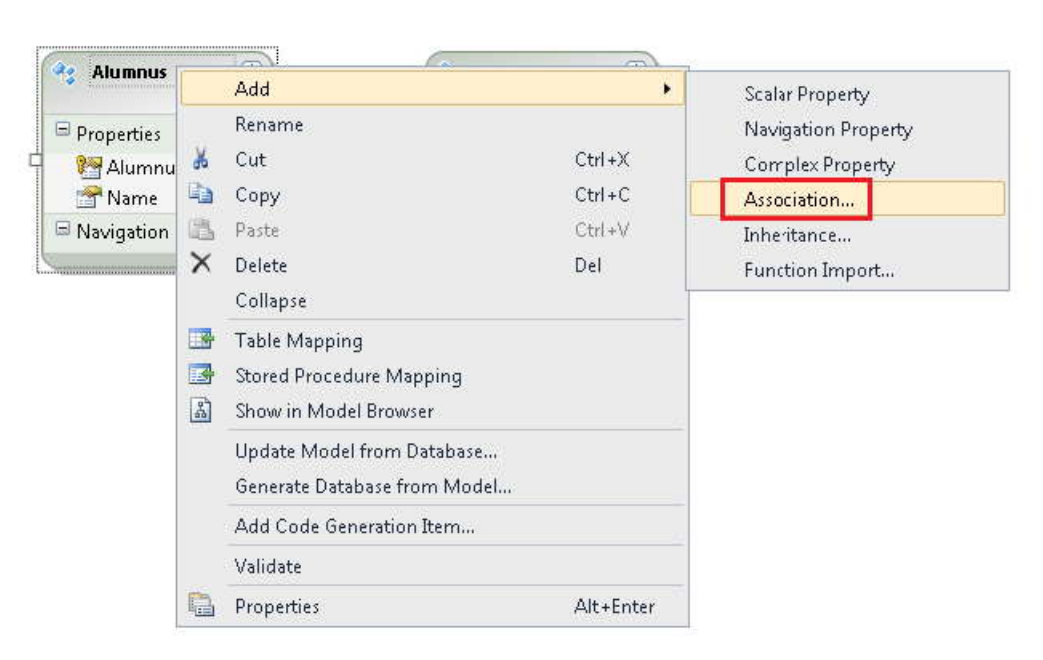


نام موجودیت **Entity1** را به **Alumnus** تغییر دهید، نام خصوصیت **Id** را به **AlumnusId** تغییر دهید، و یک خصوصیت کمیتی (scalar) با نام **Name** اضافه کنید. برای اضافه کردن خصوصیت های جدید می توانید پس از تغییر نام ستون **Id** کلید **Enter** را فشار دهید، یا روی موجودیت کلیک راست کرده و **Add Scalar Property** را انتخاب کنید، نوع پیش فرض برای خصوصیت های جدید **String** است، که برای این مثال مناسب است، اما مطمئنا می توانید چیزهایی مانند نوع داده را در پنجره **Properties** تغییر دهید.

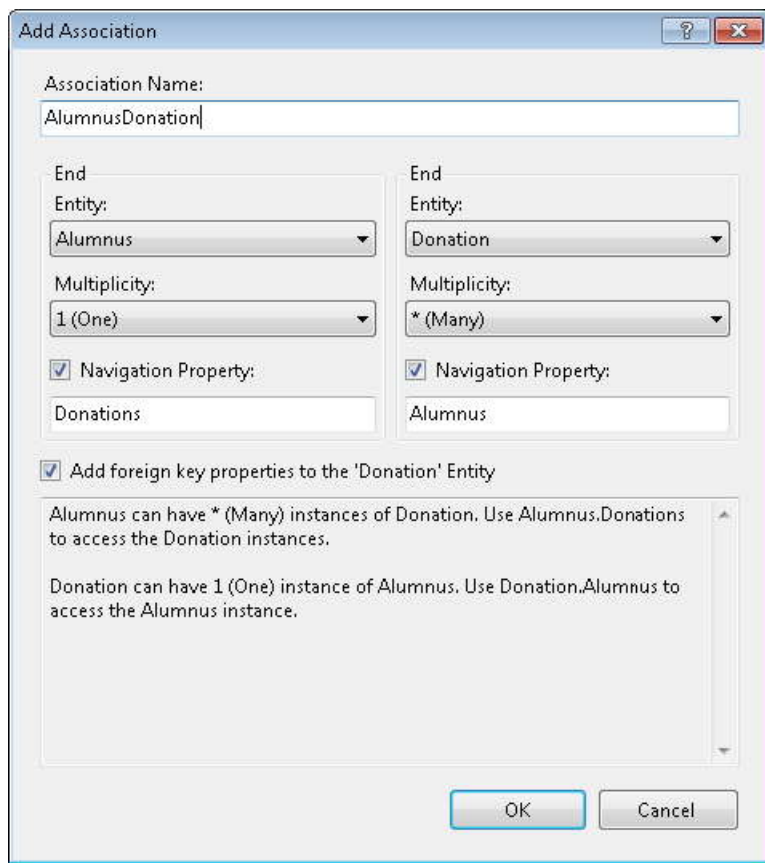
به این روش یک موجودیت دیگر با نام **Donation** ایجاد کنید، خصوصیت **Id** آن را به **DonationId** تغییر داده و یک خصوصیت کمیتی با نام **DateAndAmount** اضافه کنید.



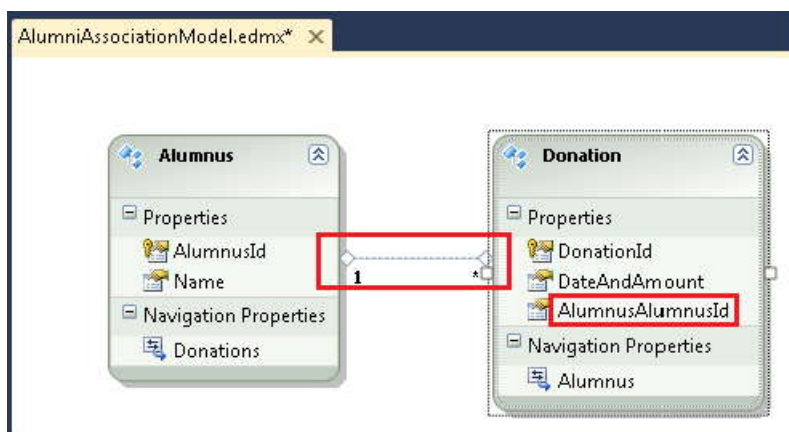
برای اضافه کردن وابستگی بین این دو موجودیت، روی موجودیت **Alumnus** کلیک کرده و **Add** و سپس **Association** را انتخاب کنید.



مقدار پیش فرض در کادر محاوره ای **Add Association** همان چیزی است که در این مثال می خواهیم (یک به چند، شامل خصوصیت های navigation، و کلیدهای خارجی)، بنابراین فقط روی **OK** کلیک کنید.

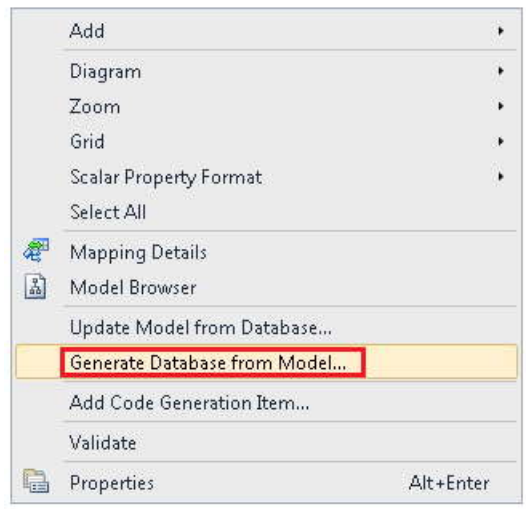


Designer یک خط وابستگی و یک خصوصیت کلید خارجی اضافه می کند.



اکنون شما آماده هستید، تا دیتابیس را ایجاد کنید. روی ناحیه خالی محیط طراحی راست کلیک کرده و

Generate database from model را انتخاب کنید.



با این کار Generate Database Wizard بالا می آید. (اگر خطاری مشاهده کردید که نشان می دهد موجودیت ها نگاشته شده نیستند، می توانید آن خطارها را در آن لحظه نادیده بگیرید.)

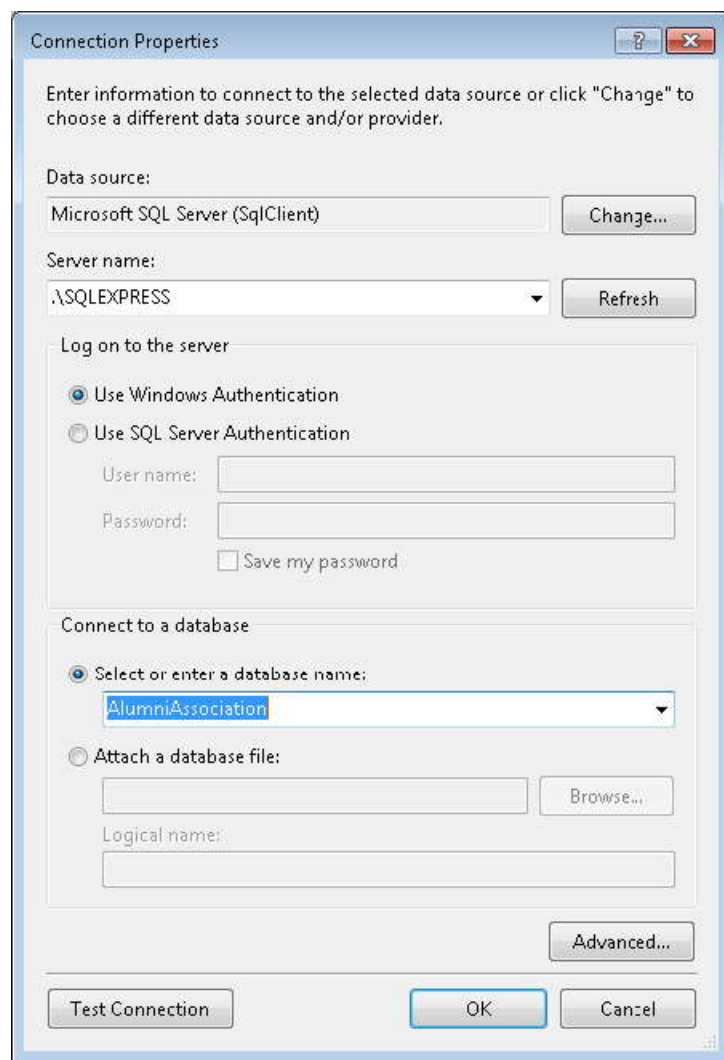
در مرحله **Choose Your Data Connection** روی **New Connection** کلیک کنید.



در کادر محاوره ای **Connection Properties**، نمونه¹ SQL Server Express محلی² را انتخاب کرده و نام دیتابیس را AlumniAssociation قرار دهید.

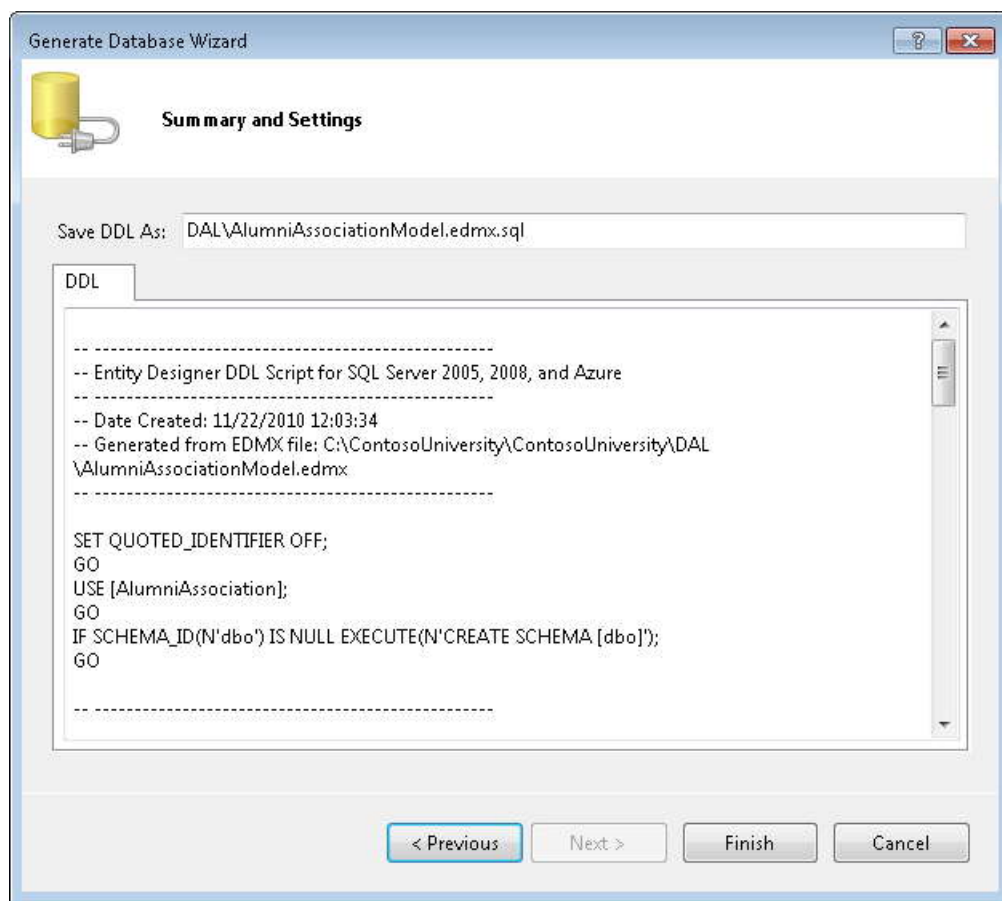
¹ instance

² local



هنگامی که از شما سوال می شود که آیا می خواهید دیتابیس ایجاد شود، روی **Yes** کلیک کنید. هنگامی که که مرحله **Choose Your Data Connection** دوباره نشان داده می شود روی **Next** کلیک کنید.

در مرحله **Summary and Settings** روی **Finish** کلیک کنید.



یک فایل sql. همراه با دستورات زبان تعریف داده (DDL) ایجاد شده است، اما دستورات هنوز اجرا نشده اند.

```

AlumniAssociation...ql - not connected X AlumniAssociationModel.edmx*
-----
-- Entity Designer DDL Script for SQL Server 2005, 2008, and Azure
-----
-- Date Created: 11/22/2010 12:03:34
-- Generated from EDMX file: C:\ContosoUniversity\ContosoUniversity\DAL\AlumniAssociat
-----

SET QUOTED_IDENTIFIER OFF;
GO
USE [AlumniAssociation];
GO
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA [dbo]');
GO

-----
-- Dropping existing FOREIGN KEY constraints
-----
IF OBJECT_ID(N'[dbo].[FK_AlumnusDonation]', 'F') IS NOT NULL
    ALTER TABLE [dbo].[Donations] DROP CONSTRAINT [FK_AlumnusDonation];
GO

-----
-- Dropping existing tables
-----

```

از ابزاری مانند **SQL Server Management Studio** برای اجرای اسکریپت و ایجاد جدول ها استفاده کنید.

شما اکنون می توانید از مدل داده AlumniAssociation در صفحه های وب، همانند شیوه ای که از مدل School استفاده می کردید، استفاده کنید.

با استفاده از، **Server Explorer** سطرهای زیر را به جدول های Alumnus و Donation اضافه کنید.

AlumnusId	Name	DonationId	DateAndAmo...	AlumnusAlumnusId
1	Roger Zheng	1	1/1/2010 \$100	1
2	Candace Kapoor	2	1/2/2010 \$200	1
		3	1/3/2010 \$300	2

صفحه جدیدی با نام *Alumni.aspx* که از *Site.Master* استفاده می کند ایجاد کنید. کد ساخت یافته زیر را به کنترل **Content** به نام **Content2** اضافه کنید:

```

<h2>Alumni</h2>
<asp:EntityDataSource ID="AlumniEntityDataSource" runat="server"
ContextTypeName="ContosoUniversity.DAL.AlumniAssociationModelContainer"
EnableFlattening="False"
EntitySetName="Alumni">
</asp:EntityDataSource>
<asp:GridView ID="AlumniGridView" runat="server"
DataSourceID="AlumniEntityDataSource" AutoGenerateColumns="False"
OnRowDataBound="AlumniGridView_RowDataBound"
DataKeyNames="AlumnusId">
<Columns>
<asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name"/>
<asp:TemplateField HeaderText="Donations">
<ItemTemplate>
<asp:GridView ID="DonationsGridView" runat="server" AutoGenerateColumns="False">
<Columns>
<asp:BoundField DataField="DateAndAmount" HeaderText="Date and Amount"/>
</Columns>
</asp:GridView>
</ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>

```

این کد ساختیافته کنترل های **GridView** تودرتو را ایجاد می کند، کنترل بیرونی برای نمایش نام های فارغ التحصیلان و کنترل درونی برای نمایش تاریخ و میزان هدیه.

Alumni.aspx.cs را باز کرده و یک دستور **using** برای لایه دسترسی به داده و یک اداره کننده برای رخداد **RowDataBound** کنترل **GridView** بیرونی اضافه کنید:

```

using ContosoUniversity.DAL;

....
protected void AlumniGridView_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if(e.Row.RowType==DataControlRowType.DataRow)
    {
        var alumnus = e.Row.DataItem as Alumnus;
        var donationsGridView =(GridView)e.Row.FindControl("DonationsGridView");
        donationsGridView.DataSource= alumnus.Donation.ToList();
        donationsGridView.DataBind();
    }
}

```

این کد با استفاده از خصوصیت Donation navigation سطر جاری موجودیت Alumnus، داده ها را به کنترل GridView درونی مقید می کند.

صفحه را اجرا کنید.

ALUMNI	
Name	Donations
Roger Zheng	Date and Amount
	1/1/2010 \$100
	1/2/2010 \$200
Candace Kapoor	Date and Amount
	1/3/2010 \$300

(توجه: این صفحه در پروژه ای که برای دانلود قرار داده شده، موجود است، اما برای اینکه کار کند، باید دیتابیس را در نمونه SQL Server Express محلی ایجاد کنید. دیتابیس به عنوان فایل *.mdf* در پوشه App_Data گنجانده نشده است.)

برای اطلاعات بیشتر در مورد استفاده از ویژگی ابتدا-مدل¹ Entity Framework، به [Model-First in the Entity Framework 4](#) مراجعه کنید.

پشتیبانی از POCO

هنگام استفاده از شیوه طراحی [domain-driven](#)، کلاس های داده ای طراحی می کنید که داده ها و رفتارهای وابسته به دامنه تجاری را ارائه می دهد. این کلاس ها باید مستقل از هر تکنولوژی خاصی باشند که برای ذخیره (نگه داری) داده ها استفاده شده است. به عبارت دیگر، باید *persistence ignorant* باشند. *persistence ignorant* همچنین می تواند باعث شود که تست واحد یک کلاس آسان تر شود چون پروژه تست واحد می تواند از هر آنچه برای تکنولوژی *persistence* مناسب تر است، برای تست استفاده کند. نسخه های پیشین Entity Framework پشتیبانی کمی از *persistence ignorant* ارائه می دادند چون کلاس های موجودیت باید از کلاس EntityObject ارث می بردند و بنابراین مقدار زیادی عملیات های خاص Entity Framework را در بر می گیرند.

¹ Model first

Entity Framework 4 توانایی استفاده از کلاس های موجودیتی که از کلاس EntityObject ارث نمی برند و بنابراین persistence ignorant هستند را ارائه می دهد. در مفاهیم Entity Framework، این گونه کلاس ها معمولا *plain-old CLR objects* (POCOs یا POCOs) نامیده می شوند. شما می توانید کلاس های POCO را به طور دستی بنویسید، یا به طور خودکار آنها را بر اساس مدل داده موجود با استفاده از قالب های Text Template Transformation Toolkit (T4) که توسط Entity Framework ارائه می شوند، تولید کنید.

برای اطلاعات بیشتر در مورد استفاده از POCOs در Entity Framework، به منابع زیر مراجعه نمایید:

- [Working with POCO Entities](#)

این یک سند MSDN است که مروری بر POCOs دارد، همراه با لینک هایی به دیگر اسناد که اطلاعات جزئی تری در این باره دارند.

- [Walkthrough: POCO Template for the Entity Framework](#)

وبلاگی از تیم توسعه Entity Framework، همراه با لینک هایی به دیگر بلاگ پست ها در مورد POCOs

توسعه Code-first

POCO در Entity Framework 4 هنوز نیاز دارد به اینکه شما یک مدل داده ایجاد کنید و کلاس های موجودیت را به مدل داده پیوند دهید. نسخه های بعد Entity Framework دارای ویژگی است که *code-first development* نامیده می شود. این ویژگی شما را قادر می سازد به استفاده از Entity Framework به همراه کلاس های POCO خودتان، بدون نیاز به استفاده از طراح مدل داده یا فایل XML مدل داده. (بنابراین، این گزینه نیز *code-only* نامیده می شود. *code-first* و *code-only* هر دو به یک ویژگی همسان در Entity Framework اشاره می کنند).

برای اطلاعات بیشتر در مورد شیوه *code-first* برای توسعه، به منابع زیر مراجعه نمایید:

- [Getting Started with Entity Framework Using MVC](#)

- [Code-First Development with Entity Framework 4](#)

این بلاگ پستی از Scott Guthrie است که به معرفی توسعه Code-first می پردازد.

- [Entity Framework Development Team Blog - posts tagged CodeOnly](#)
- [Entity Framework Development Team Blog - posts tagged Code First](#)
- [MVC Music Store tutorial - Part 4: Models and Data Access](#)
- [Getting Started with MVC 3 - Part 4: Entity Framework Code-First Development](#)

اطلاعات بیشتر

به منظور داشتن منابع بیشتر برای استفاده از Entity Framework از سایت های زیر دیدن کنید:

- [Using the Entity Framework 4.0 and the ObjectDataSource Control, Part 1: Getting Started](#)
- [Entity Framework FAQ](#)
- [The Entity Framework Team Blog](#)
- [Entity Framework in the MSDN Library](#)
- [Entity Framework in the MSDN Data Developer Center](#)
- [EntityDataSource Web Server Control Overview in the MSDN Library](#)
- [EntityDataSource control API reference in the MSDN Library](#)
- [Entity Framework Forums on MSDN](#)
- [Julie Lerman's blog](#)
- [What's New in ADO.NET](#) مطالب سایت MSDN در مورد ویژگی های جدید Entity framework 4
- [Announcing the release of Entity Framework 4](#) بلاگ پست تیم توسعه Entity Framework در مورد ویژگی های جدید Entity Framework 4