

# The Entity Framework and ASP.NET – Getting Started

---

The Contoso University sample web application demonstrates how to create ASP.NET Web Forms applications using the Entity Framework. The sample application is a website for a fictional Contoso University. It includes functionality such as student admission, course creation, and instructor assignments.

This tutorial series details the steps taken to build the Contoso University sample application. You can [download the completed application](#) or create it by following the steps in the tutorial. If you have questions about the tutorial, please post them to the [ADO.NET, Entity Framework, LINQ to SQL, NHibernate forum](#) on the ASP.NET website.

This tutorial series uses the ASP.NET Web Forms model. It assumes you know how to work with ASP.NET Web Forms in Visual Studio. If you don't, a good place to start is a [basic ASP.NET Web Forms Tutorial](#). If you prefer to work with the ASP.NET MVC framework, see the [Creating Model Classes with the Entity Framework](#) tutorial.

# Contents

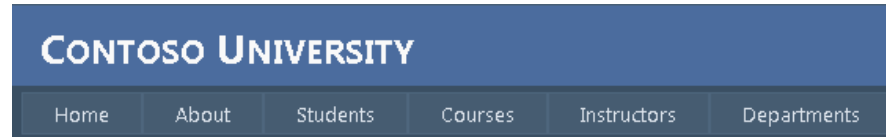
---

<b>Part 1: Overview .....</b>	<b>4</b>
Creating the Web Application .....	5
Creating the Database .....	7
Creating the Entity Framework Data Model .....	10
Exploring the Entity Framework Data Model .....	13
<b>Part 2: The EntityDataSource Control .....</b>	<b>21</b>
Adding and Configuring the EntityDataSource Control .....	21
Configuring Database Rules to Allow Deletion .....	25
Using a GridView Control to Read and Update Entities .....	28
Revising EntityDataSource Control Markup to Improve Performance .....	32
Displaying Data from a Navigation Property .....	34
Using a DetailsView Control to Insert Entities .....	36
Displaying Data in a Drop-Down List .....	37
<b>Part 3: Filtering, Ordering, and Grouping Data .....</b>	<b>40</b>
Using the EntityDataSource "Where" Property to Filter Data .....	41
Using the EntityDataSource "OrderBy" Property to Order Data .....	42
Using a Control Parameter to Set the "Where" Property .....	43
Using the EntityDataSource "GroupBy" Property to Group Data .....	46
Using the QueryExtender Control for Filtering and Ordering .....	47
Using the "Like" Operator to Filter Data .....	50
<b>Part 4: Working with Related Data .....</b>	<b>53</b>
Displaying and Updating Related Entities in a GridView Control .....	53
Displaying Related Entities in a Separate Control .....	58
Using the EntityDataSource "Selected" Event to Display Related Data .....	62
<b>Part 5: Working with Related Data, Continued .....</b>	<b>66</b>
Adding an Entity with a Relationship to an Existing Entity .....	67
Working with Many-to-Many Relationships .....	69
<b>Part 6: Implementing Table-per-Hierarchy Inheritance .....</b>	<b>75</b>
Table-per-Hierarchy versus Table-per-Type Inheritance .....	75
Adding Instructor and Student Entities .....	76
Mapping Instructor and Student Entities to the Person Table .....	80
Using the Instructor and Student Entities .....	81
<b>Part 7: Using Stored Procedures .....</b>	<b>88</b>
Creating Stored Procedures in the Database .....	88

Adding the Stored Procedures to the Data Model .....	91
Mapping the Stored Procedures .....	92
Using Insert, Update, and Delete Stored Procedures .....	96
Using Select Stored Procedures .....	97
<b>Part 8: Using Dynamic Data Functionality to Format and Validate Data.....</b>	<b>99</b>
Using DynamicField and DynamicControl Controls.....	99
Adding Metadata to the Data Model .....	103
<b>Disclaimer .....</b>	<b>107</b>

# Part 1: Overview

The application you'll be building in these tutorials is a simple university website.



**WELCOME TO CONTOSO UNIVERSITY!**

Users can view and update student, course, and instructor information. A few of the screens you'll create are shown below.

## STUDENT LIST

	<a href="#">ID</a>	<a href="#">Name</a>	<a href="#">EnrollmentDate</a>
<a href="#">Edit</a> <a href="#">Delete</a>	3	Peggy Justice	9/1/2001
<a href="#">Edit</a> <a href="#">Delete</a>	6	Yan Li	9/1/2002
<a href="#">Edit</a> <a href="#">Delete</a>	7	Laura Norman	9/1/2003

## ADD NEW STUDENTS

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

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

## COURSES TAUGHT

	ID	Title	Department
<a href="#">Select</a>	2030	Poetry	English

## COURSE DETAILS

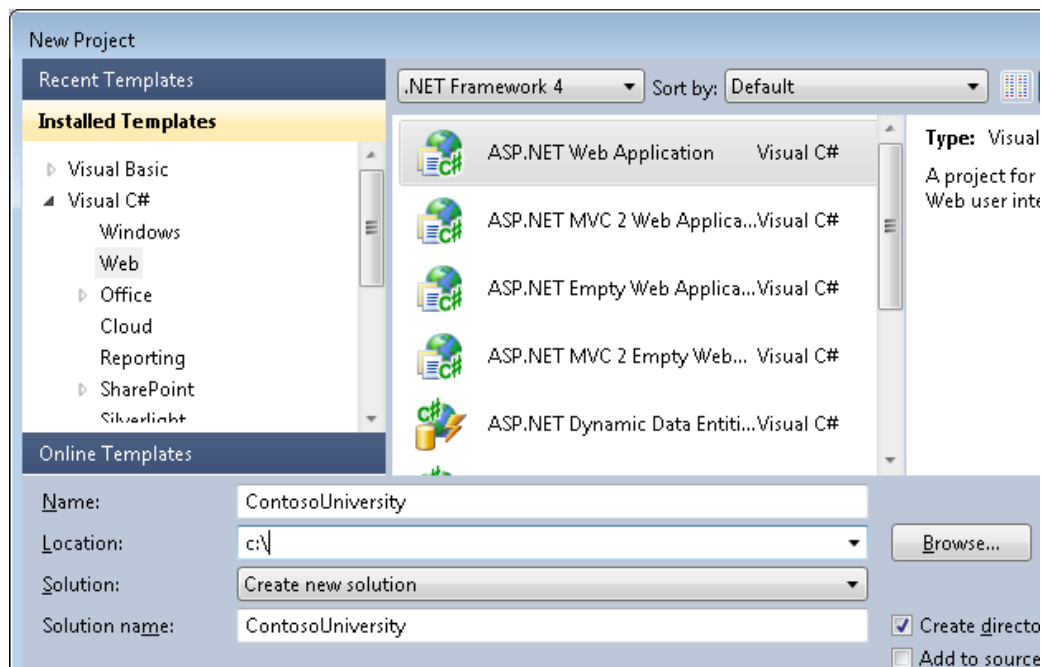
ID	2030
Title	Poetry
Credits	2
Department	English
Location	
URL	<a href="http://www.fineart">http://www.fineart</a>

## STUDENT GRADES

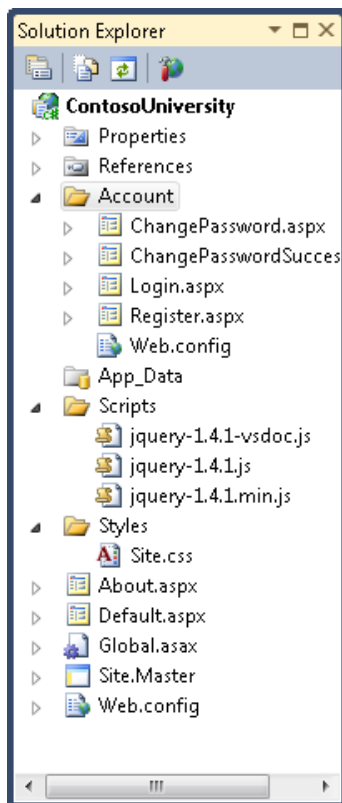
ID	Name	Grade
2	Barzdukas, Gytis	3.50
3	Justice, Peggy	4.00

## Creating the Web Application

To start the tutorial, open Visual Studio and then create a new ASP.NET Web Application Project using the **ASP.NET Web Application** template:



This template creates a web application project that already includes a style sheet and master pages:



Open the *Site.Master* file and change "My ASP.NET Application" to "Contoso University".

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

Find the *Menu* control named **NavigationMenu** and replace it with the following markup, which adds menu items for the pages you'll be creating.

```
<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>
    </Items>
</asp:Menu>
```

```

        <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>

```

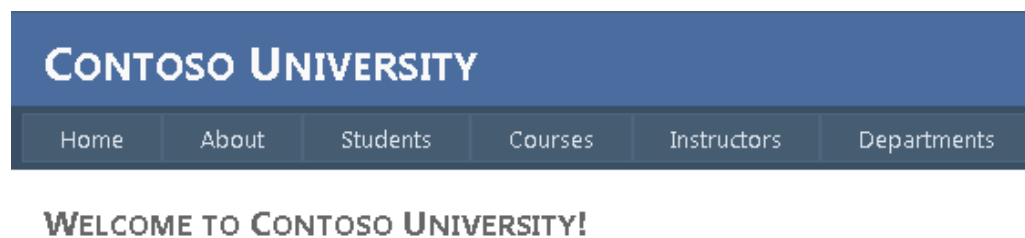
Open the *Default.aspx* page and change the **Content** control named **BodyContent** to this:

```

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

```

You now have a simple home page with links to the various pages that you'll be creating:



## Creating the Database

---

For these tutorials, you'll use the Entity Framework data model designer to automatically create the data model based on an existing database (often called the *database-first* approach). An alternative that's not covered in this tutorial series is to create the data model manually and then have the designer generate scripts that create the database (the *model-first* approach).

For the database-first method used in this tutorial, the next step is to add a database to the site. The easiest way is to first download the project that goes with this tutorial. Then right-click the *App\_Data* folder, select **Add Existing Item**, and select the *School.mdf* database file from the downloaded project.

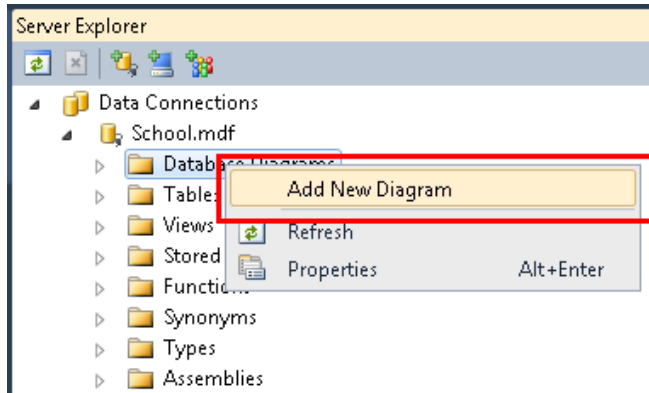
An alternative is to follow the instructions at [Creating the School Sample Database](#). Whether you download the database or create it, copy the *School.mdf* file from the following folder to your application's *App\_Data* folder:

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

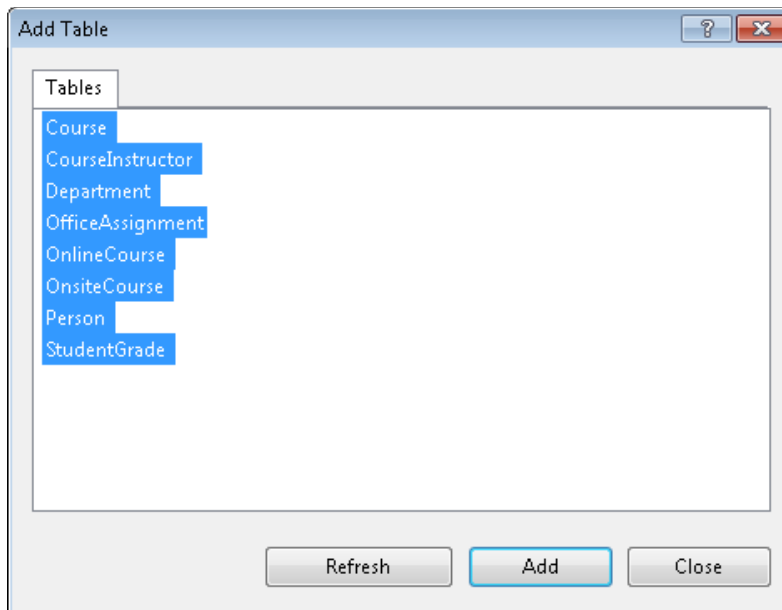
(This location of the *.mdf* file assumes you're using SQL Server 2008 Express.)

If you create the database from a script, perform the following steps to create a database diagram:

1. In **Server Explorer**, expand **Data Connections**, expand *School.mdf*, right-click **Database Diagrams**, and select **Add New Diagram**.



2. Select all of the tables and then click **Add**.

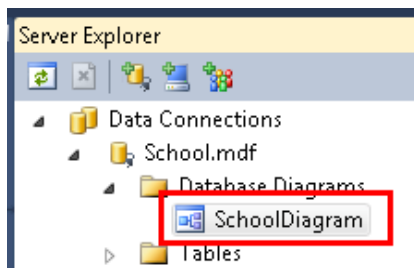


SQL Server creates a database diagram that shows tables, columns in the tables, and relationships between the tables. You can move the tables around to organize them however you like.

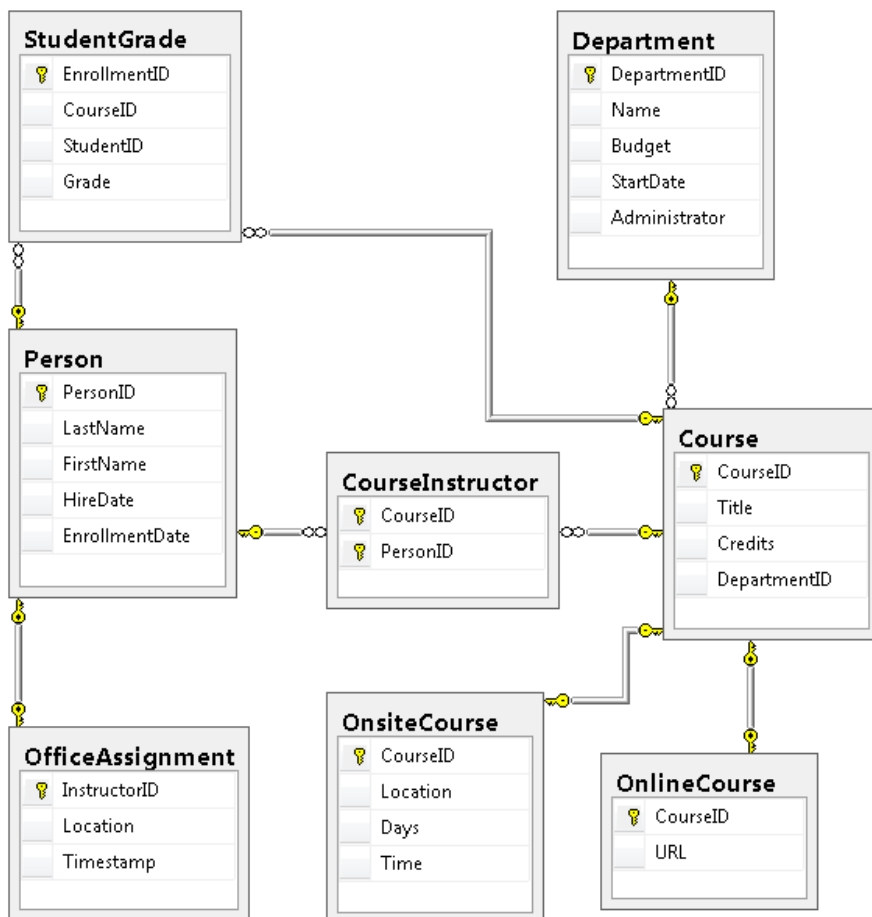
3. Save the diagram as "SchoolDiagram" and close it.



If you download the *School.mdf* file that goes with this tutorial, you can view the database diagram by double-clicking **SchoolDiagram** under **Database Diagrams** in **Server Explorer**.



The diagram looks something like this (the tables might be in different locations from what's shown here):



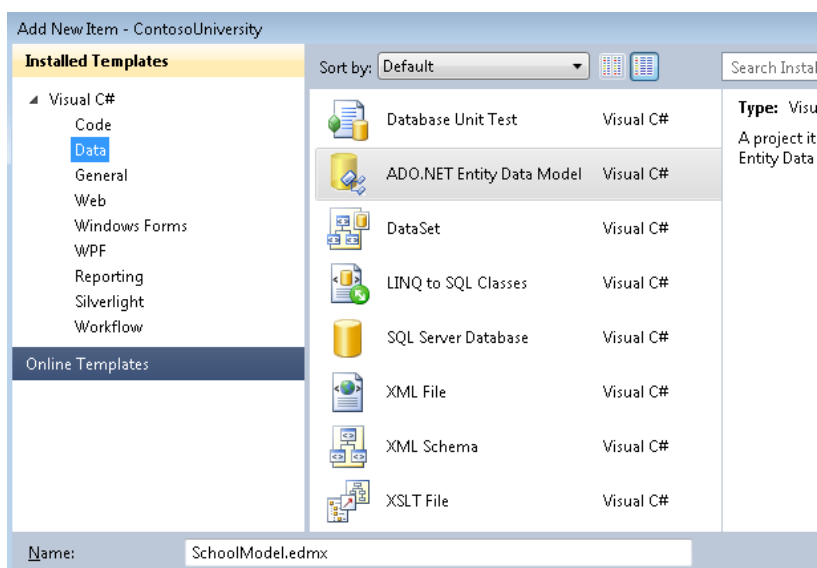
## Creating the Entity Framework Data Model

---

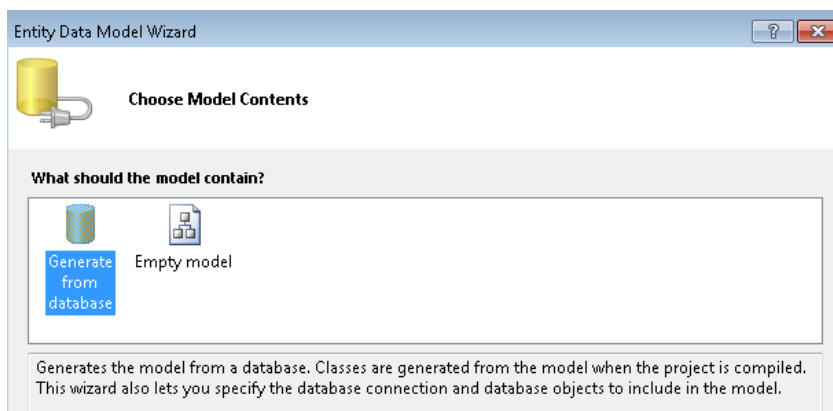
Now you can create an Entity Framework data model from this database. You could create the data model in the root folder of the application, but for this tutorial you'll place it in a folder named *DAL* (for Data Access Layer).

In **Solution Explorer**, add a project folder named *DAL* (make sure it's under the project, not under the solution).

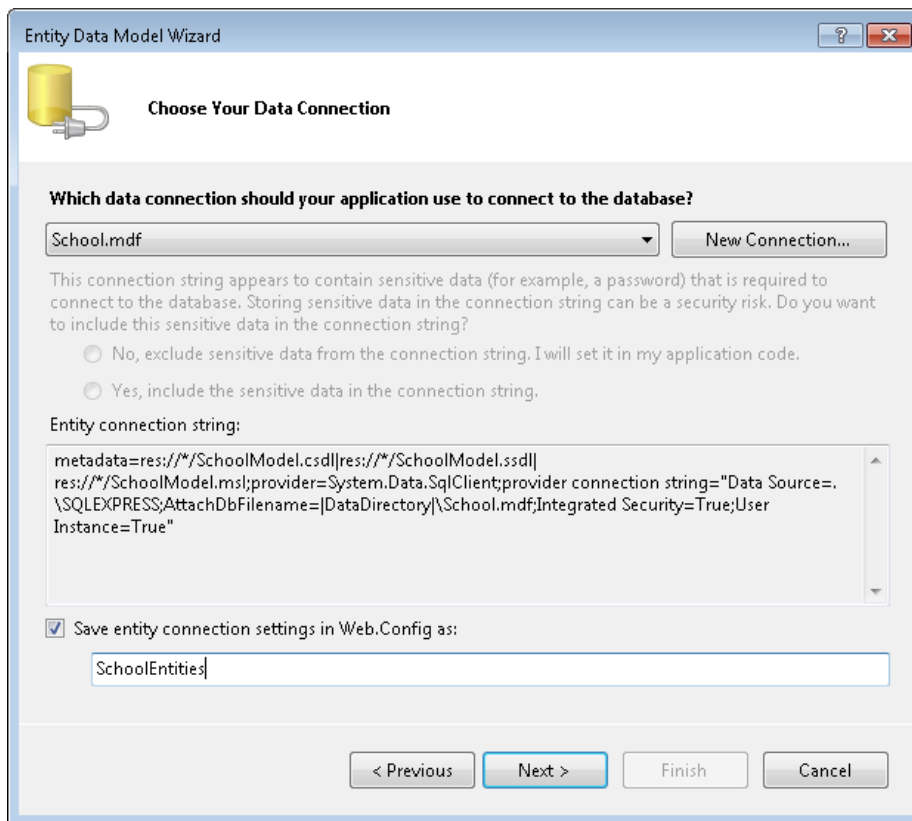
Right-click the *DAL* folder and then select **Add** and **New Item**. Under **Installed Templates**, select **Data**, select the **ADO.NET Entity Data Model** template, name it *SchoolModel.edmx*, and then click **Add**.



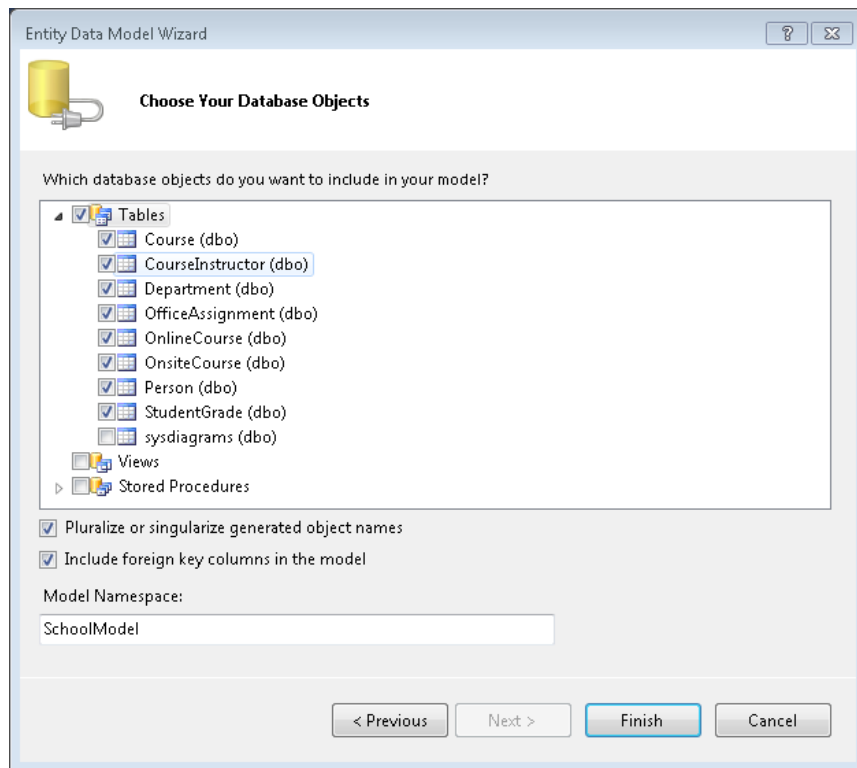
This starts the Entity Data Model Wizard. In the first wizard step, the **Generate from database** option is selected by default. Click **Next**.



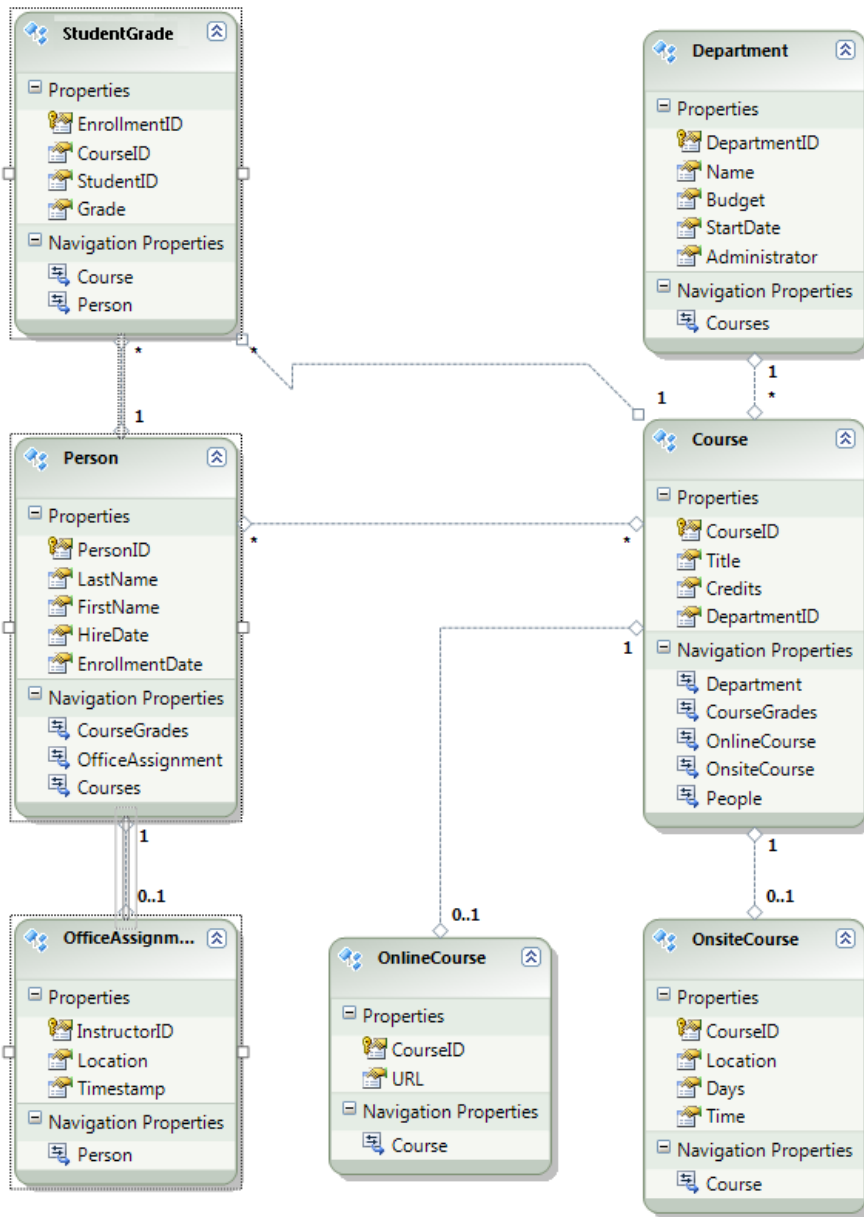
In the **Choose Your Data Connection** step, leave the default values and click **Next**. The School database is selected by default and the connection setting is saved in the *Web.config* file as **SchoolEntities**.



In the **Choose Your Database Objects** wizard step, select all of the tables except **sysdiagrams** (which was created for the diagram you generated earlier) and then click **Finish**.



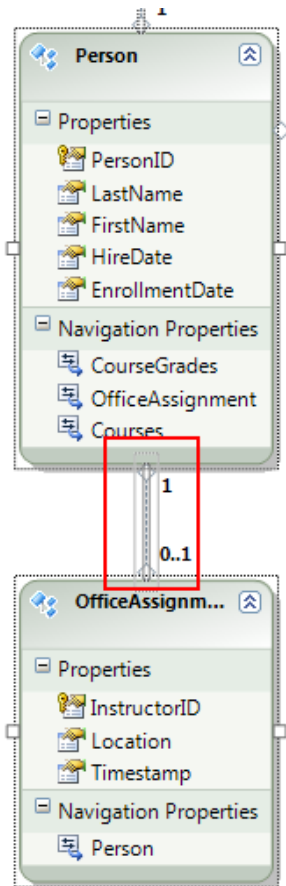
After it's finished creating the model, Visual Studio shows you a graphical representation of the Entity Framework objects (entities) that correspond to your database tables. (As with the database diagram, the location of individual elements might be different from what you see in this illustration. You can drag the elements around to match the illustration if you want.)



## Exploring the Entity Framework Data Model

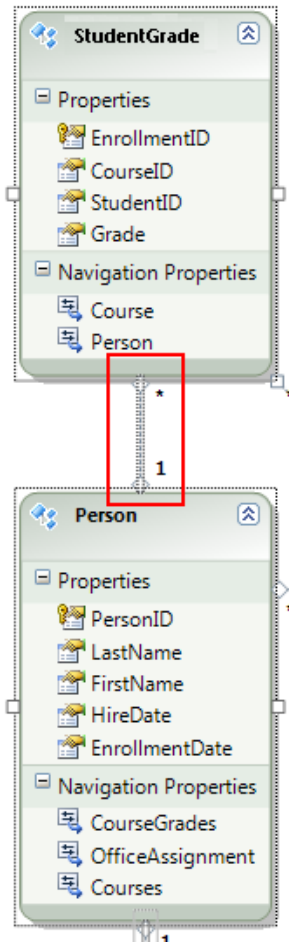
You can see that the entity diagram looks very similar to the database diagram, with a couple of differences. One difference is the addition of symbols at the end of each association that indicate the type of association (table relationships are called entity associations in the data model):

- A one-to-zero-or-one association is represented by "1" and "0..1".



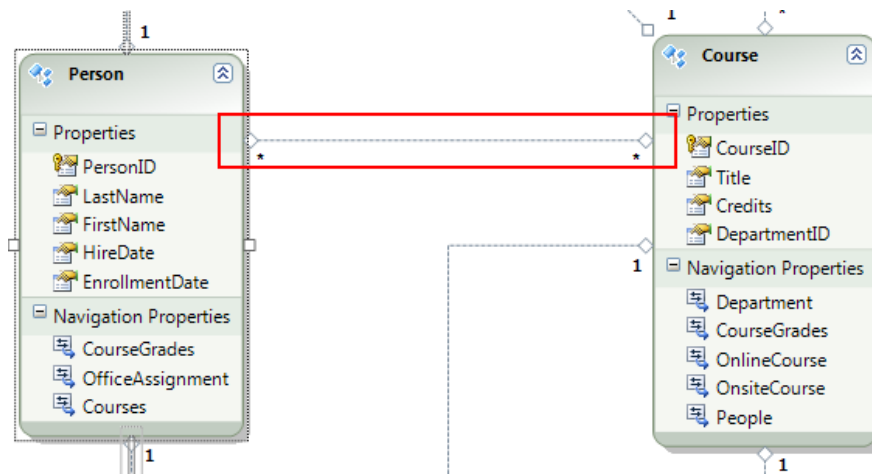
In this case, a **Person** entity may or may not be associated with an **OfficeAssignment** entity. An **OfficeAssignment** entity must be associated with a **Person** entity. In other words, an instructor may or may not be assigned to an office, and any office can be assigned to only one instructor.

- A one-to-many association is represented by "1" and "\*".



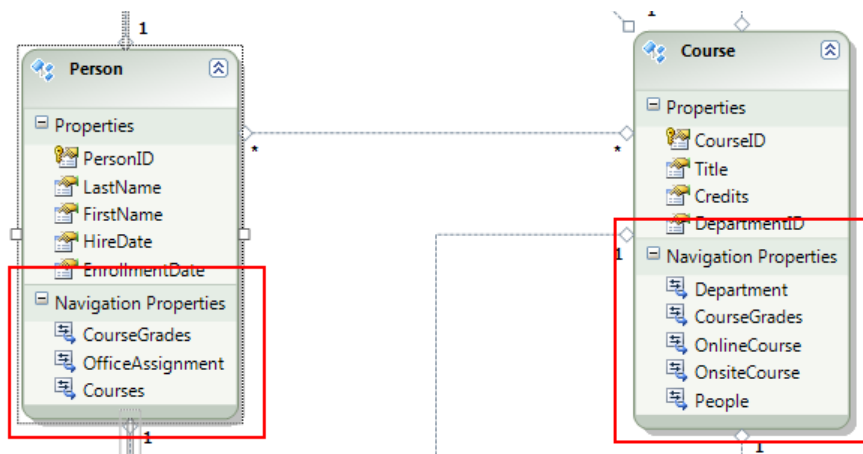
In this case, a **Person** entity may or may not have associated **StudentGrade** entities. A **StudentGrade** entity must be associated with one **Person** entity. **StudentGrade** entities actually represent enrolled courses in this database; if a student is enrolled in a course and there's no grade yet, the **Grade** property is null. In other words, a student may not be enrolled in any courses yet, may be enrolled in one course, or may be enrolled in multiple courses. Each grade in an enrolled course applies to only one student.

- A many-to-many association is represented by "\*" and "\*".



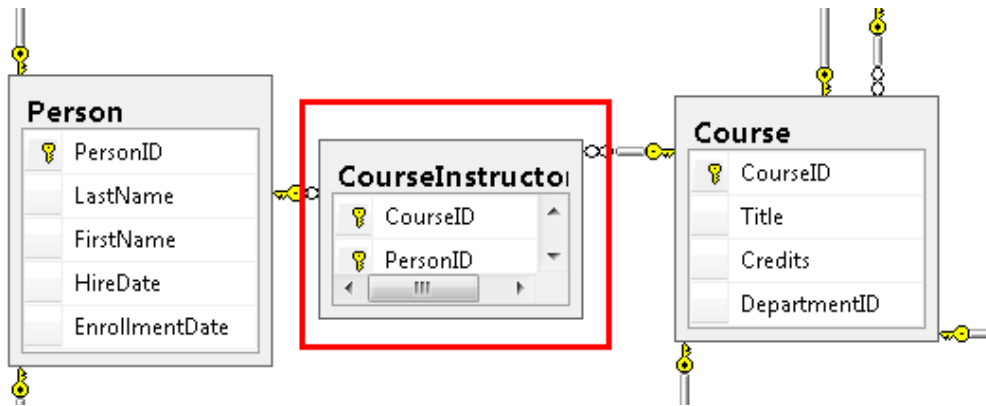
In this case, a **Person** entity may or may not have associated **Course** entities, and the reverse is also true: a **Course** entity may or may not have associated **Person** entities. In other words, an instructor may teach multiple courses, and a course may be taught by multiple instructors. (In this database, this relationship applies only to instructors; it does not link students to courses. Students are linked to courses by the StudentGrades table.)

Another difference between the database diagram and the data model is the additional **Navigation Properties** section for each entity. A navigation property of an entity references related entities. For example, the **Courses** property in a **Person** entity contains a collection of all the **Course** entities that are related to that **Person** entity.



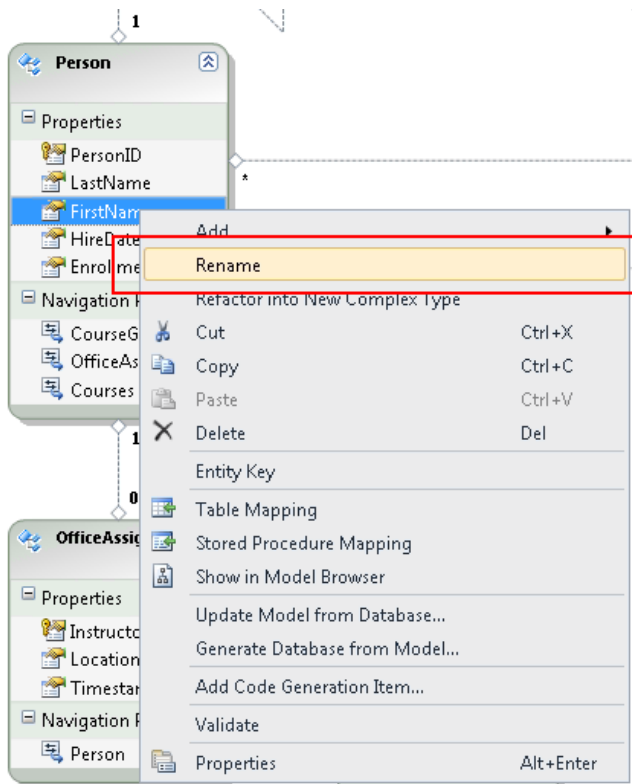
Yet another difference between the database and data model is the absence of the **CourseInstructor** association table that's used in the database to link the **Person** and **Course** tables in a many-to-many relationship. The navigation properties enable you to get related **Course** entities from the **Person** entity and related **Person** entities from the **Course** entity, so there's no need to represent the association table in the data model.



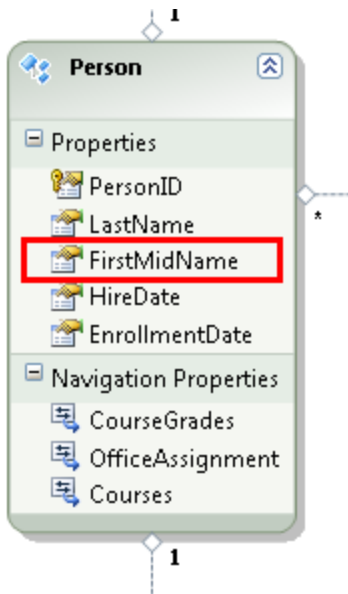


For purposes of this tutorial, suppose the **FirstName** column of the **Person** table actually contains both a person's first name and middle name. You want to change the name of the field to reflect this, but the database administrator (DBA) might not want to change the database. You can change the name of the **FirstName** property in the data model, while leaving its database equivalent unchanged.

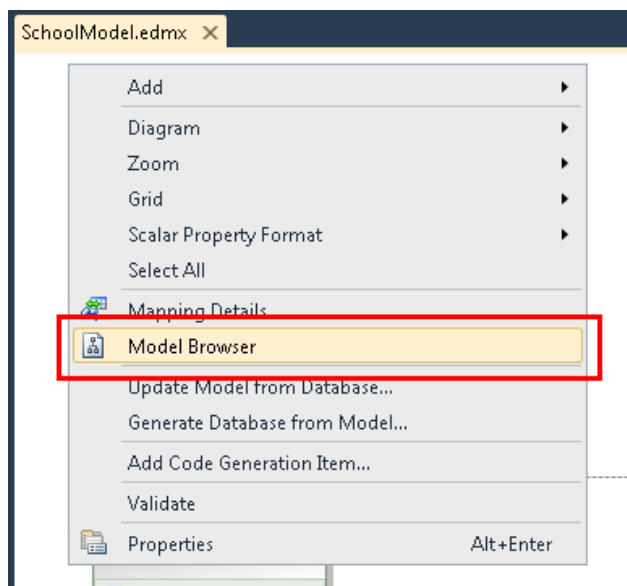
In the designer, right-click **FirstName** in the **Person** entity, and then select **Rename**.



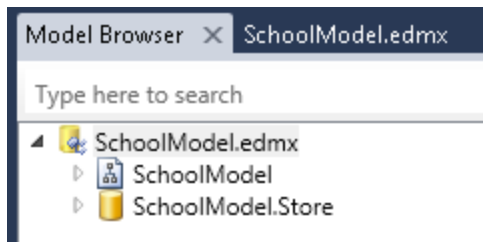
Type in the new name "FirstMidName". This changes the way you refer to the column in code without changing the database.



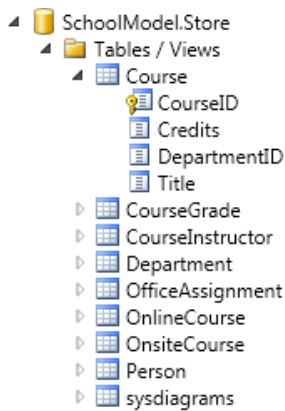
The model browser provides another way to view the database structure, the data model structure, and the mapping between them. To see it, right-click a blank area in the entity designer and then click **Model Browser**.



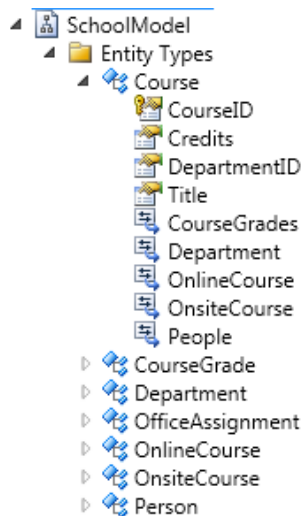
The **Model Browser** pane displays a tree view. (The **Model Browser** pane might be docked with the **Solution Explorer** pane.) The **SchoolModel** node represents the data model structure, and the **SchoolModel.Store** node represents the database structure.



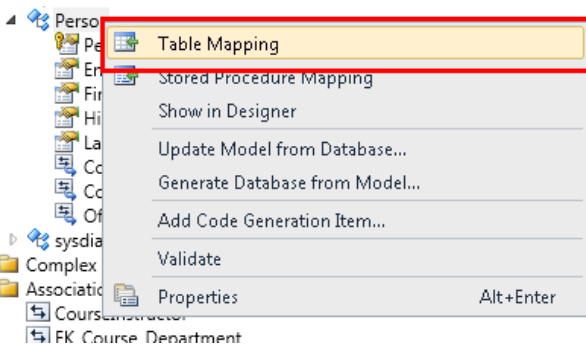
Expand **SchoolModel.Store** to see the tables, expand **Tables / Views** to see tables, and then expand **Course** to see the columns within a table.



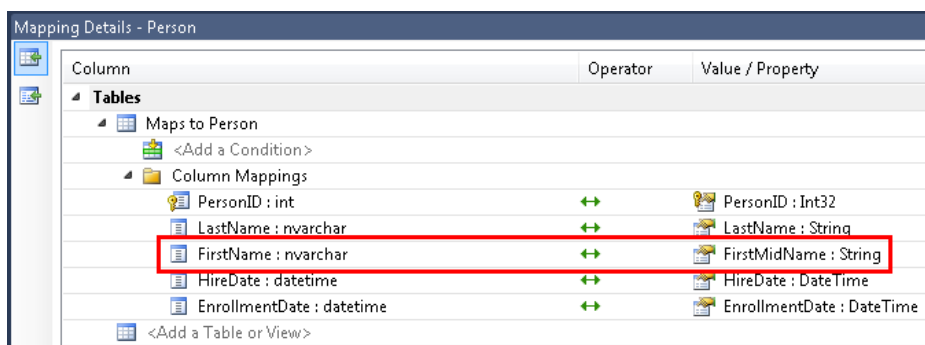
Expand **SchoolModel**, expand **Entity Types**, and then expand the **Course** node to see the entities and the properties within the entities.



In either the designer or the **Model Browser** pane you can see how the Entity Framework relates the objects of the two models. Right-click the **Person** entity and select **Table Mapping**.



This opens the **Mapping Details** window. Notice that this window lets you see that the database column **FirstName** is mapped to **FirstMidName**, which is what you renamed it to in the data model.



The Entity Framework uses XML to store information about the database, the data model, and the mappings between them. The *SchoolModel.edmx* file is actually an XML file that contains this information. The designer renders the information in a graphical format, but you can also view the file as XML by right-clicking the *.edmx* file in **Solution Explorer**, clicking **Open With**, and selecting **XML (Text) Editor**. (The data model designer and an XML editor are just two different ways of opening and working with the same file, so you cannot have the designer open and open the file in an XML editor at the same time.)

You've now created a website, a database, and a data model. In the next walkthrough you'll begin working with data using the data model and the ASP.NET **EntityDataSource** control.

## Part 2: The EntityDataSource Control

In the previous tutorial you created a web site, a database, and a data model. In this tutorial you work with the **EntityDataSource** control that ASP.NET provides in order to make it easy to work with an Entity Framework data model. You'll create a **GridView** control for displaying and editing student data, a **DetailsView** control for adding new students, and a **DropDownList** control for selecting a department (which you'll use later for displaying associated courses).

### STUDENT LIST

	<a href="#">Name</a>	<a href="#">EnrollmentDate</a>	Number of Courses
<a href="#">Edit</a> <a href="#">Delete</a>	Abercrombie, Kim		0
<a href="#">Edit</a> <a href="#">Delete</a>	Barzdukas, Gytis	9/1/2005	2
<a href="#">Edit</a> <a href="#">Delete</a>	Justice, Peggy	9/1/2001	2

### ADD NEW STUDENTS

FirstMidName	<input type="text" value="John"/>
LastName	<input type="text" value="Smith"/>
EnrollmentDate	<input type="text" value="1/1/2011"/>
<a href="#">Insert</a> <a href="#">Cancel</a>	

### COURSES BY DEPARTMENT

Select a department: 

Engineering	▼
Engineering	
English	
Economics	
Mathematics	

Note that in this application you won't be adding input validation to pages that update the database, and some of the error handling will not be as robust as would be required in a production application. That keeps the tutorial focused on the Entity Framework and keeps it from getting too long. For details about how to add these features to your application, see [Validating User Input in ASP.NET Web Pages](#) and [Error Handling in ASP.NET Pages and Applications](#).

## Adding and Configuring the EntityDataSource Control

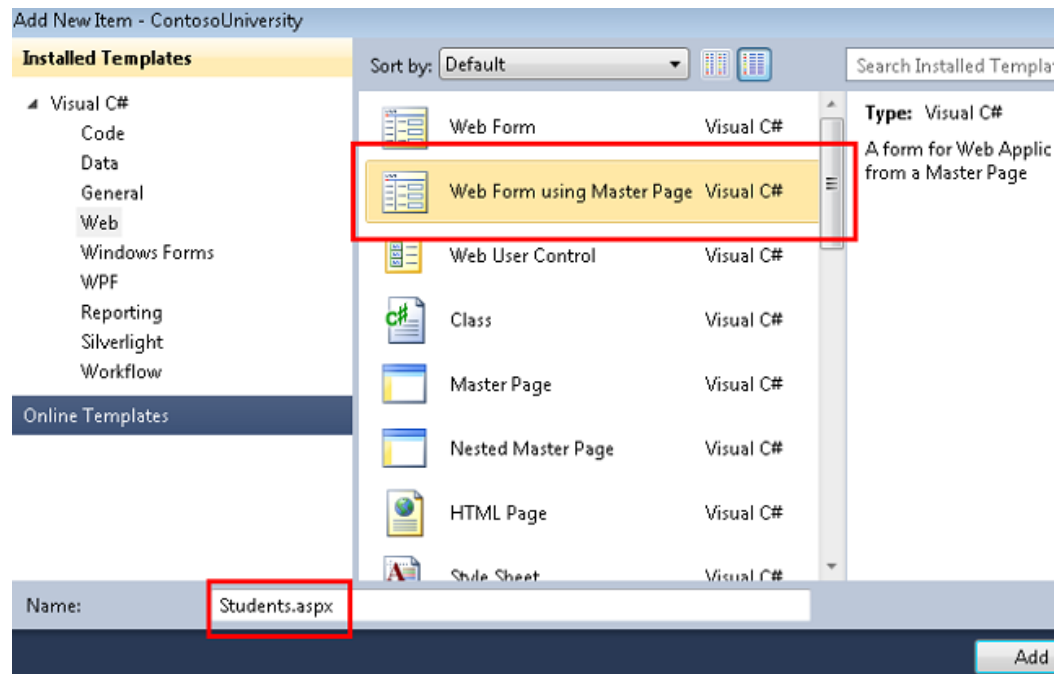
---

You'll begin by configuring an **EntityDataSource** control to read **Person** entities from the **People** entity set.

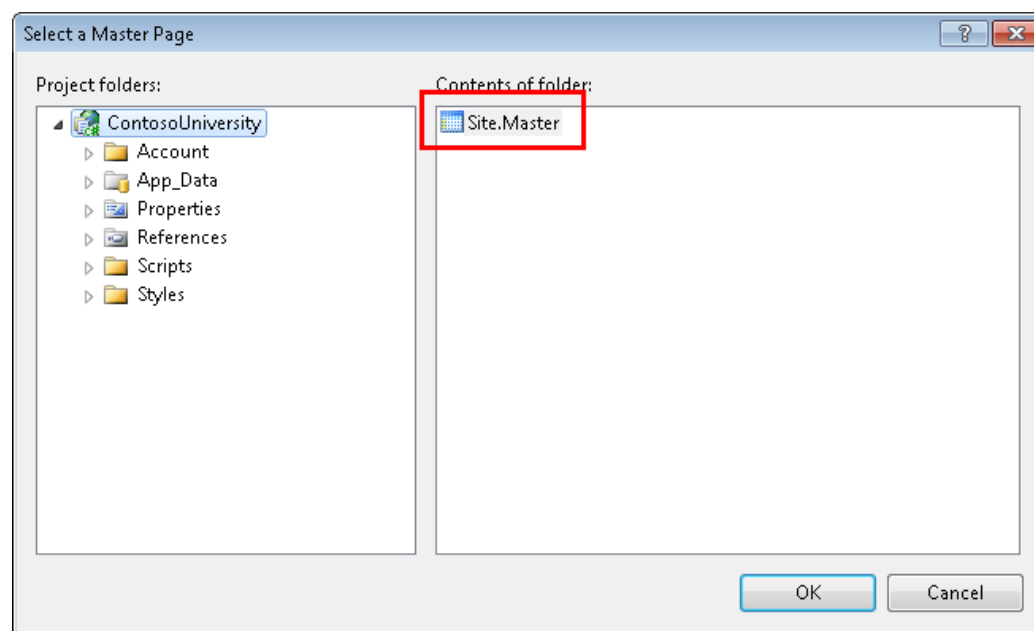
Make sure you have Visual Studio open and that you're working with the project you created in part 1. If you haven't built the project since you created the data model or since the last change you made to it,

build the project now. Changes to the data model are not made available to the designer until the project is built.

Create a new web page using the **Web Form using Master Page** template, and name it *Students.aspx*.



Specify *Site.Master* as the master page. All of the pages you create for these tutorials will use this master page.



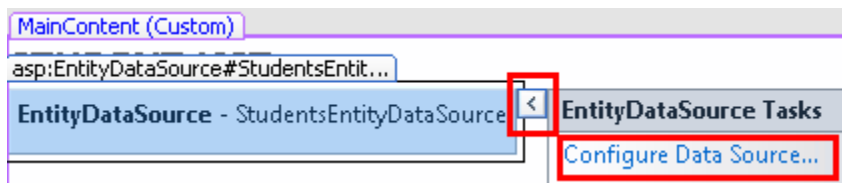
In **Source** view, add an **h2** heading to the **Content** control named **Content2**, as shown in the following example:

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

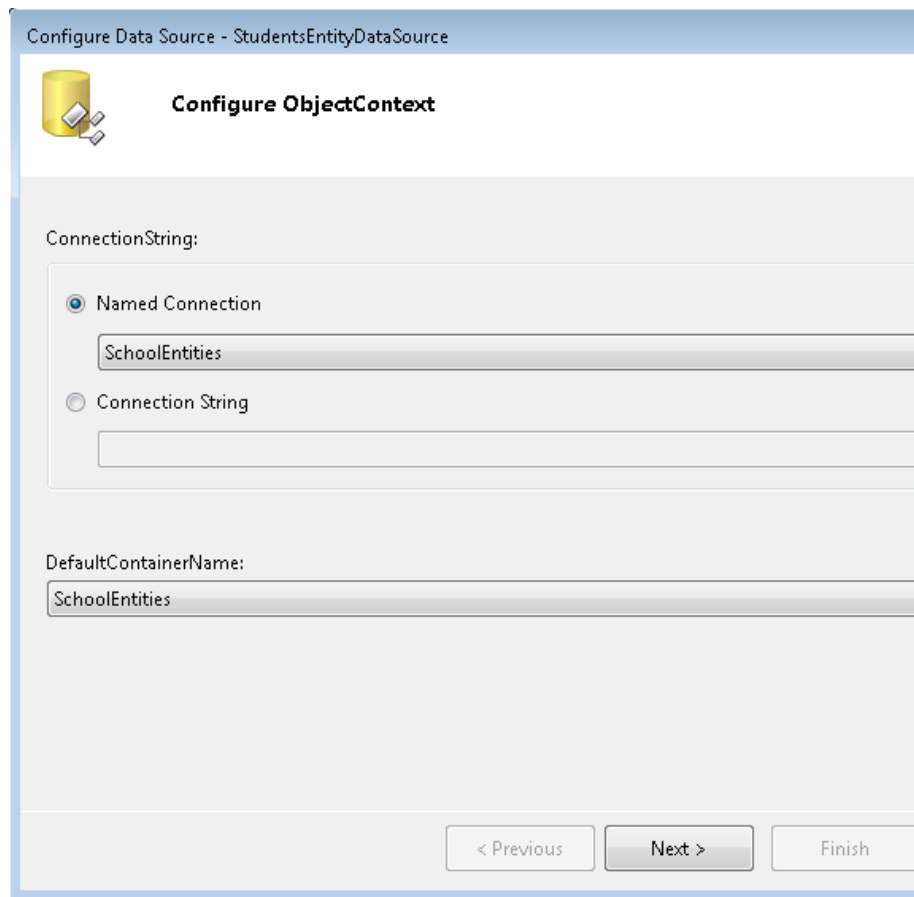
From the **Data** tab of the **Toolbox**, drag an **EntityDataSource** control to the page, drop it below the heading, and change the ID to **StudentsEntityDataSource**:

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

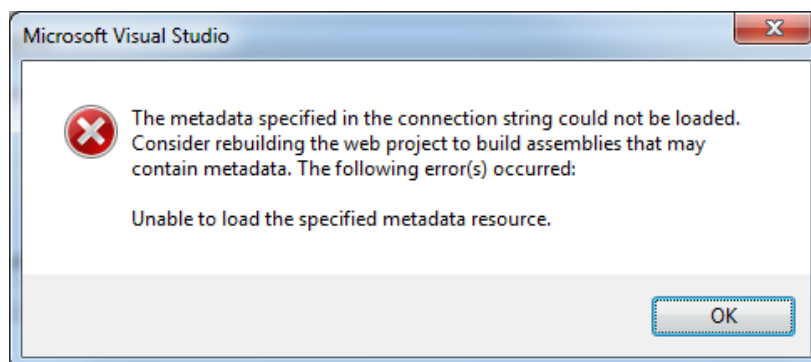
Switch to **Design** view, click the data source control's smart tag, and then click **Configure Data Source** to launch the **Configure Data Source** wizard.



In the **ConfigureObjectContext** wizard step, select **SchoolEntities** as the value for **Named Connection**, and select **SchoolEntities** as the **DefaultContainerName** value. Then click **Next**.




Note: If you get the following dialog box at this point, you have to build the project before proceeding.)



In the **Configure Data Selection** step, select **People** as the value for **EntitySetName**. Under **Select**, make sure the **Select All** check box is selected. Then select the options to enable update and delete. When you're done, click **Finish**.



Configure Data Source - StudentsEntityDataSource

 **Configure Data Selection**

EntitySetName:

EntityTypeFilter:

Select:

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

☐ Enable automatic inserts

☒ Enable automatic updates

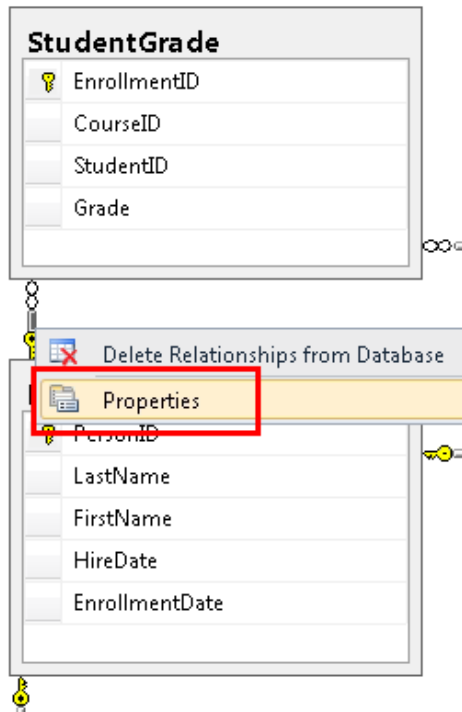
☒ Enable automatic deletes

## Configuring Database Rules to Allow Deletion

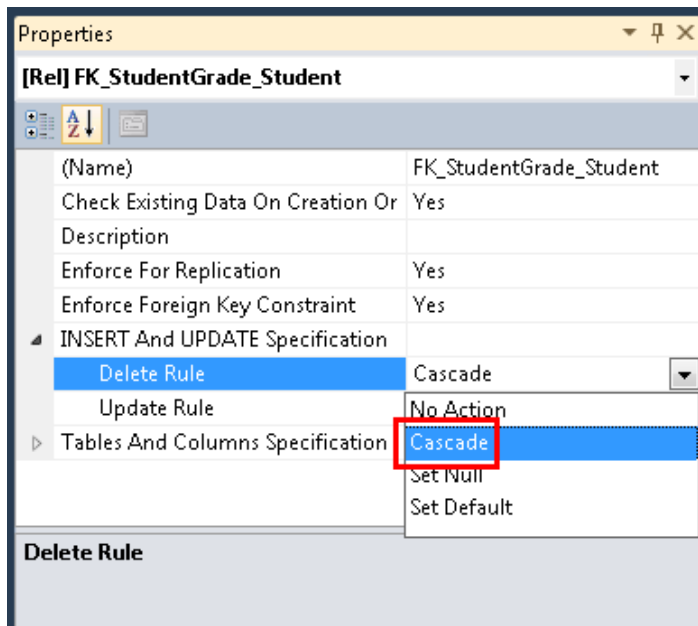
You'll be creating a page that lets users delete students from the **Person** table, which has three relationships with other tables (**Course**, **StudentGrade**, and **OfficeAssignment**). By default, the database will prevent you from deleting a row in **Person** if there are related rows in one of the other tables. You can manually delete the related rows first, or you can configure the database to delete them automatically when you delete a **Person** row. For student records in this tutorial, you'll configure the database to delete the related data automatically. Because students can have related rows only in the **StudentGrade** table, you need to configure only one of the three relationships.

If you're using the *School.mdf* file that you downloaded from the project that goes with this tutorial, you can skip this section because these configuration changes have already been done. If you created the database by running a script, configure the database by performing the following procedures.

In **Server Explorer**, open the database diagram that you created in part 1. Right-click the relationship between **Person** and **StudentGrade** (the line between tables), and then select **Properties**.

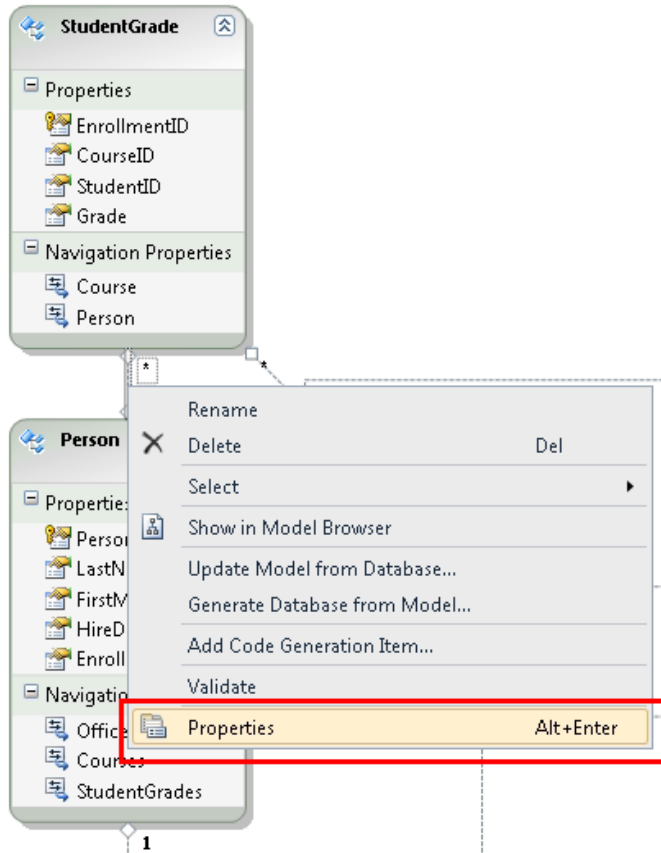


In the **Properties** window, expand **INSERT and UPDATE Specification** and set the **DeleteRule** property to **Cascade**.

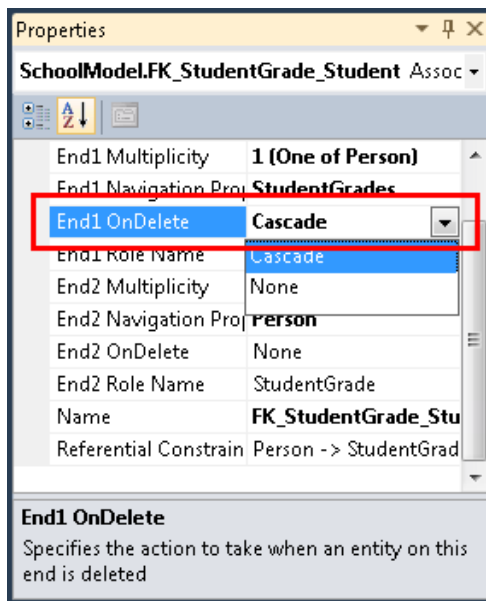


Save and close the diagram. If you're asked whether you want to update the database, click **Yes**.

To make sure that the model keeps entities that are in memory in sync with what the database is doing, you must set corresponding rules in the data model. Open *SchoolModel.edmx*, right-click the association line between **Person** and **StudentGrade**, and then select **Properties**.



In the **Properties** window, set **End1 OnDelete** to **Cascade**.



Save and close the *SchoolModel.edmx* file, and then rebuild the project.

In general, when the database changes, you have several choices for how to sync up the model:

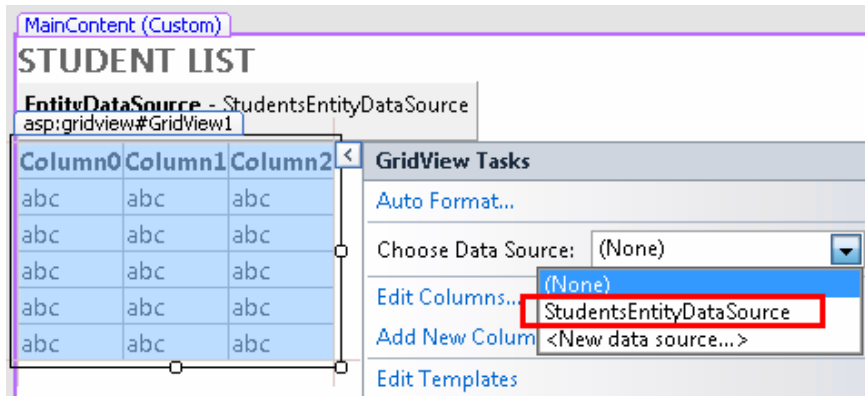
- For certain kinds of changes (such as adding or refreshing tables, views, or stored procedures), right-click in the designer and select **Update Model from Database** to have the designer make the changes automatically.
- Regenerate the data model.
- Make manual updates like this one.

In this case, you could have regenerated the model or refreshed the tables affected by the relationship change, but then you'd have to make the field-name change again (from *FirstName* to *FirstMidName*).

## Using a GridView Control to Read and Update Entities

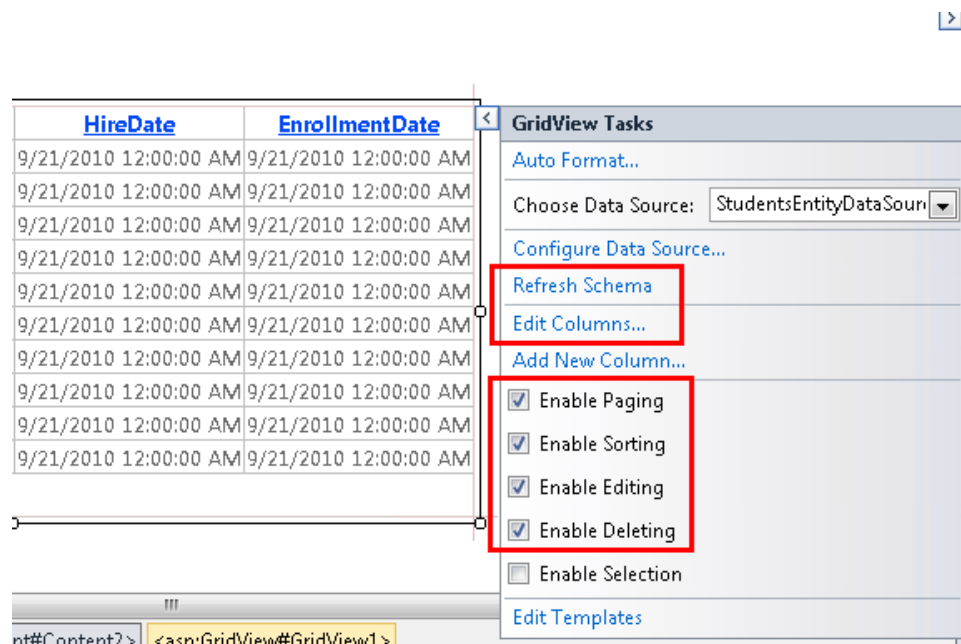
In this section you'll use a *GridView* control to display, update, or delete students.

Open or switch to *Students.aspx* and switch to **Design** view. From the **Data** tab of the **Toolbox**, drag a *GridView* control to the right of the *EntityDataSource* control, name it *StudentsGridView*, click the smart tag, and then select **StudentsEntityDataSource** as the data source.

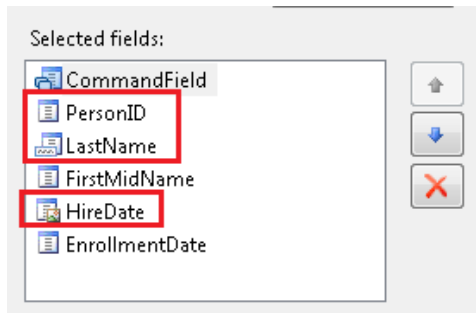


Click **Refresh Schema** (click **Yes** if you're prompted to confirm), then click **Enable Paging**, **Enable Sorting**, **Enable Editing**, and **Enable Deleting**.

Click **Edit Columns**.

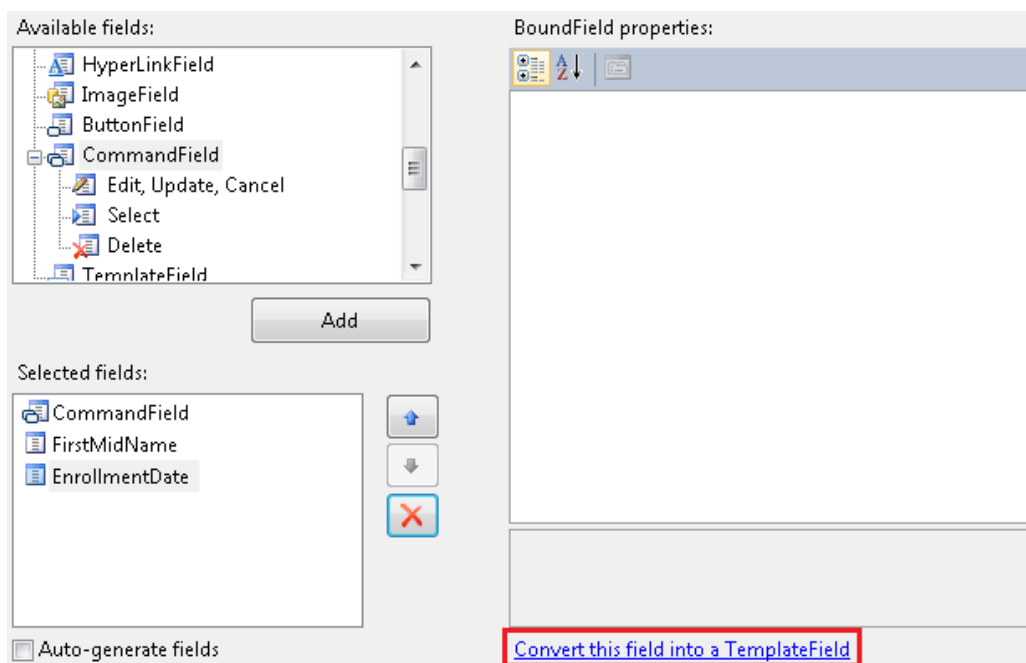


In the **Selected fields** box, delete **PersonID**, **LastName**, and **HireDate**. You typically don't display a record key to users, hire date is not relevant to students, and you'll put both parts of the name in one field, so you only need one of the name fields.)



Select the **FirstMidName** field and then click **Convert this field into a TemplateField**.

Do the same for **EnrollmentDate**.



Click **OK** and then switch to **Source** view. The remaining changes will be easier to do directly in markup. The **GridView** control markup now looks like the following example.

```
<asp:GridView ID="StudentsGridView" runat="server" AllowPaging="True"
    AllowSorting="True" AutoGenerateColumns="False"
    DataKeyNames="PersonID" DataSourceID="StudentsEntityDataSource">
    <Columns>
        <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
        <asp:TemplateField HeaderText="FirstMidName" SortExpression="FirstMidName">
            <EditItemTemplate>
                <asp:TextBox ID="TextBox1" runat="server"
                    Text='<%# Bind("FirstMidName") %>'></asp:TextBox>
            </EditItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

```

        <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>

```

The first column after the command field is a template field that currently displays the first name. Change the markup for this template field to look like the following example:

```

<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>

```

In display mode, two **Label** controls display the first and last name. In edit mode, two text boxes are provided so you can change the first and last name. As with the **Label** controls in display mode, you use **Bind** and **Eval** expressions exactly as you would with ASP.NET data source controls that connect directly to databases. The only difference is that you're specifying entity properties instead of database columns.

The last column is a template field that displays the enrollment date. Change the markup for this field to look like the following example:

```

<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>

```

In both display and edit mode, the format string "{0:d}" causes the date to be displayed in the "short date" format. (Your computer might be configured to display this format differently from the screen images shown in this tutorial.)

Notice that in each of these template fields, the designer used a **Bind** expression by default, but you've changed that to an **Eval** expression in the **ItemTemplate** elements. The **Bind** expression makes the data available in **GridView** control properties in case you need to access the data in code. In this page you don't need to access this data in code, so you can use **Eval**, which is more efficient. For more information, see [Getting your data out of the data controls](#).

## Revising EntityDataSource Control Markup to Improve Performance

---

In the markup for the **EntityDataSource** control, remove the **ConnectionString** and **DefaultContainerName** attributes and replace them with a **ContextTypeName="ContosoUniversity.DAL.SchoolEntities"** attribute. This is a change you should make every time you create an **EntityDataSource** control, unless you need to use a connection that is different from the one that's hard-coded in the object context class. Using the **ContextTypeName** attribute provides the following benefits:

- Better performance. When the **EntityDataSource** control initializes the data model using the **ConnectionString** and **DefaultContainerName** attributes, it performs additional work to load metadata on every request. This isn't necessary if you specify the **ContextTypeName** attribute.
- Lazy loading is turned on by default in generated object context classes (such as **SchoolEntities** in this tutorial) in Entity Framework 4.0. This means that navigation properties are loaded with related data automatically right when you need it. Lazy loading is explained in more detail later in this tutorial.
- Any customizations that you've applied to the object context class (in this case, the **SchoolEntities** class) will be available to controls that use the **EntityDataSource** control. Customizing the object context class is an advanced topic that is not covered in this tutorial series. For more information, see [Extending Entity Framework Generated Types](#).

The markup will now resemble the following example (the order of the properties might be different):



```

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

```

The **EnableFlattening** attribute refers to a feature that was needed in earlier versions of the Entity Framework because foreign key columns were not exposed as entity properties. The current version makes it possible to use *foreign key associations*, which means foreign key properties are exposed for all but many-to-many associations. If your entities have foreign key properties and no [complex types](#), you can leave this attribute set to **False**. Don't remove the attribute from the markup, because the default value is **True**. For more information, see [Flattening Objects \(EntityDataSource\)](#).

Run the page and you see a list of students and employees (you'll filter for just students in the next tutorial). The first name and last name are displayed together.

## STUDENT LIST

	<a href="#">Name</a>	<a href="#">EnrollmentDate</a>
<a href="#">Edit</a> <a href="#">Delete</a>	Abercrombie, Kim	
<a href="#">Edit</a> <a href="#">Delete</a>	Barzdukas, Gytis	9/1/2005
<a href="#">Edit</a> <a href="#">Delete</a>	Justice, Peggy	9/1/2001
<a href="#">Edit</a> <a href="#">Delete</a>	Fakhouri, Fadi	
<a href="#">Edit</a> <a href="#">Delete</a>	Harui, Roger	
<a href="#">Edit</a> <a href="#">Delete</a>	Li, Yan	9/1/2002
<a href="#">Edit</a> <a href="#">Delete</a>	Norman, Laura	9/1/2003
<a href="#">Edit</a> <a href="#">Delete</a>	Olivotto, Nino	9/1/2005
<a href="#">Edit</a> <a href="#">Delete</a>	Tang, Wayne	9/1/2005
<a href="#">Edit</a> <a href="#">Delete</a>	Alonso, Meredith	9/1/2002
<a href="#">1</a> <a href="#">2</a> <a href="#">3</a> <a href="#">4</a>		

To sort the display, click a column name.

Click **Edit** in any row. Text boxes are displayed where you can change the first and last name.

## STUDENT LIST

Student List		Name	EnrollmentDate
<a href="#">Edit</a> <a href="#">Delete</a>	Abercrombie, Kim		
<a href="#">Update</a> <a href="#">Cancel</a>	<input type="text" value="Barzdukas"/>	<input type="text" value="Gytis"/>	9/1/2005
<a href="#">Edit</a> <a href="#">Delete</a>	Justice, Peggy		9/1/2001
<a href="#">Edit</a> <a href="#">Delete</a>	Fakhouri, Fadi		
<a href="#">Edit</a> <a href="#">Delete</a>	Harui, Roger		
<a href="#">Edit</a> <a href="#">Delete</a>	Li, Yan		9/1/2002
<a href="#">Edit</a> <a href="#">Delete</a>	Norman, Laura		9/1/2003
<a href="#">Edit</a> <a href="#">Delete</a>	Olivotto, Nino		9/1/2005
<a href="#">Edit</a> <a href="#">Delete</a>	Tang, Wayne		9/1/2005
<a href="#">Edit</a> <a href="#">Delete</a>	Alonso, Meredith		9/1/2002
1 2 3 4			

The **Delete** button also works. Click delete for a row that has an enrollment date and the row disappears. (Rows without an enrollment date represent instructors and you may get a referential integrity error. In the next tutorial you'll filter this list to include just students.)

## Displaying Data from a Navigation Property

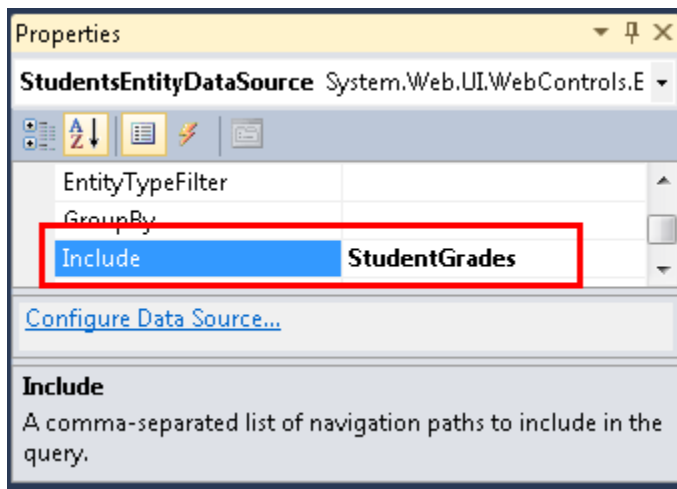
Now suppose you want to know how many courses each student is enrolled in. The Entity Framework provides that information in the **StudentGrades** navigation property of the **Person** entity. Because the database design does not allow a student to be enrolled in a course without having a grade assigned, for this tutorial you can assume that having a row in the **StudentGrade** table row that is associated with a course is the same as being enrolled in the course. (The **Courses** navigation property is only for instructors.)

When you use the **ContextTypeName** attribute of the **EntityDataSource** control, the Entity Framework automatically retrieves information for a navigation property when you access that property. This is called *lazy loading*. However, this can be inefficient, because it results in a separate call to the database each time additional information is needed. If you need data from the navigation property for every entity returned by the **EntityDataSource** control, it's more efficient to retrieve the related data along with the entity itself in a single call to the database. This is called *eager loading*, and you specify eager loading for a navigation property by setting the **Include** property of the **EntityDataSource** control.

In *Students.aspx*, you want to show the number of courses for every student, so eager loading is the best choice. If you were displaying all students but showing the number of courses only for a few of them (which would require writing some code in addition to the markup), lazy loading might be a better choice.

Open or switch to *Students.aspx*, switch to **Design** view, select **StudentsEntityDataSource**, and in the **Properties** window set the **Include** property to **StudentGrades**. (If you wanted to get multiple navigation

properties, you could specify their names separated by commas — for example, **StudentGrades, Courses**.)



Switch to **Source** view. In the **StudentsGridView** control, after the last **asp:TemplateField** element, add the following new template field:

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

In the **Eval** expression, you can reference the navigation property **StudentGrades**. Because this property contains a collection, it has a **Count** property that you can use to display the number of courses in which the student is enrolled. In a later tutorial you'll see how to display data from navigation properties that contain single entities instead of collections. (Note that you cannot use **BoundField** elements to display data from navigation properties.)

Run the page and you now see how many courses each student is enrolled in.

### STUDENT LIST

	Name	EnrollmentDate	Number of Courses
<a href="#">Edit</a> <a href="#">Delete</a>	Abercrombie, Kim		0
<a href="#">Edit</a> <a href="#">Delete</a>	Barzdukas, Gytis	9/1/2005	2
<a href="#">Edit</a> <a href="#">Delete</a>	Justice, Peggy	9/1/2001	2

## Using a DetailsView Control to Insert Entities

---

The next step is to create a page that has a **DetailsView** control that will let you add new students. Close the browser and then create a new web page using the *Site.Master* master page. Name the page *StudentsAdd.aspx*, and then switch to **Source** view.

Add the following markup to replace the existing markup for the **Content** control named **Content2**:

```
<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="People">
  </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>
```

This markup creates an **EntityDataSource** control that is similar to the one you created in *Students.aspx*, except it enables insertion. As with the **GridView** control, the bound fields of the **DetailsView** control are coded exactly as they would be for a data control that connects directly to a database, except that they reference entity properties. In this case, the **DetailsView** control is used only for inserting rows, so you have set the default mode to **Insert**.

Run the page and add a new student.

### ADD NEW STUDENTS

FirstMidName	<input type="text" value="John"/>
LastName	<input type="text" value="Smith"/>
EnrollmentDate	<input type="text" value="1/1/2011"/>
<a href="#">Insert</a> <a href="#">Cancel</a>	

Nothing will happen after you insert a new student, but if you now run *Students.aspx*, you'll see the new student information.

## Displaying Data in a Drop-Down List

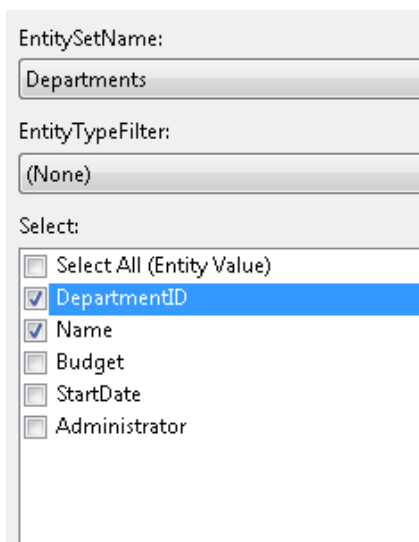
---

In the following steps you'll databind a **DropDownList** control to an entity set using an **EntityDataSource** control. In this part of the tutorial, you won't do much with this list. In subsequent parts, though, you'll use the list to let users select a department to display courses associated with the department.

Create a new web page named *Courses.aspx*. In **Source** view, add a heading to the **Content** control that's named **Content2**:

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

In **Design** view, add an **EntityDataSource** control to the page as you did before, except this time name it **DepartmentsEntityDataSource**. Select **Departments** as the **EntitySetName** value, and select only the **DepartmentID** and **Name** properties.



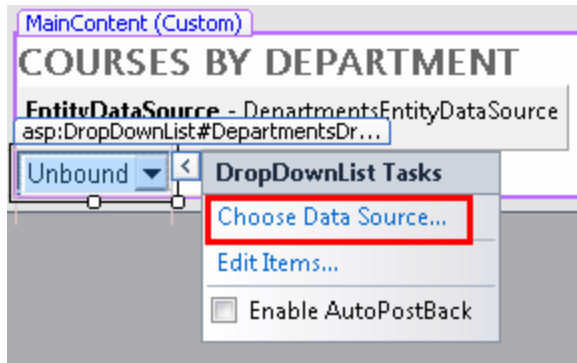
EntitySetName:  
Departments

EntityTypeFilter:  
(None)

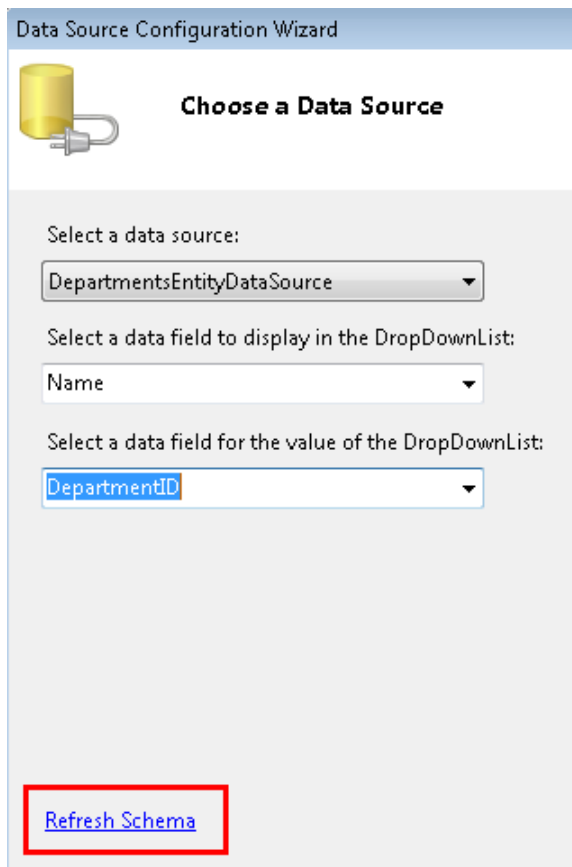
Select:

- ☐ Select All (Entity Value)
- ☒ DepartmentID
- ☒ Name
- ☐ Budget
- ☐ StartDate
- ☐ Administrator

From the **Standard** tab of the **Toolbox**, drag a **DropDownList** control to the page, name it **DepartmentsDropDownList**, click the smart tag, and select **Choose Data Source** to start the **DataSource Configuration Wizard**.



In the **Choose a Data Source** step, select **DepartmentsEntityDataSource** as the data source, click **Refresh Schema**, and then select **Name** as the data field to display and **DepartmentID** as the value data field. Click **OK**.



The method you use to databind the control using the Entity Framework is the same as with other ASP.NET data source controls except you're specifying entities and entity properties.

Switch to **Source** view and add "Select a department:" immediately before the **DropDownList** control.

Select a department:

```
<asp:DropDownList ID="DropDownList1" runat="server"
    DataSourceID="EntityDataSource1" DataTextField="Name"
    DataValueField="DepartmentID">
</asp:DropDownList>
```

As a reminder, change the markup for the **EntityDataSource** control at this point by replacing the **ConnectionString** and **DefaultContainerName** attributes with a **ContextTypeName="ContosoUniversity.DAL.SchoolEntities"** attribute. It's often best to wait until after you've created the data-bound control that is linked to the data source control before you change the **EntityDataSource** control markup, because after you make the change, the designer will not provide you with a **Refresh Schema** option in the data-bound control.

Run the page and you can select a department from the drop-down list.

## COURSES BY DEPARTMENT

Select a department:

Engineering	▼
Engineering	
English	
Economics	
Mathematics	

This completes the introduction to using the **EntityDataSource** control. Working with this control is generally no different from working with other ASP.NET data source controls, except that you reference entities and properties instead of tables and columns. The only exception is when you want to access navigation properties. In the next tutorial you'll see that the syntax you use with **EntityDataSource** control might also differ from other data source controls when you filter, group, and order data.

## Part 3: Filtering, Ordering, and Grouping Data

In the previous tutorial you used the **EntityDataSource** control to display and edit data. In this tutorial you'll filter, order, and group data. When you do this by setting properties of the **EntityDataSource** control, the syntax is different from other data source controls. As you'll see, however, you can use the **QueryExtender** control to minimize these differences.

You'll change the *Students.aspx* page to filter for students, sort by name, and search on name. You'll also change the *Courses.aspx* page to display courses for the selected department and search for courses by name. Finally, you'll add student statistics to the *About.aspx* page.

### STUDENT LIST

	Name	EnrollmentDate	Number of Courses
<a href="#">Edit</a> <a href="#">Delete</a>	Barzdukas, Gytis	9/1/2005	2
<a href="#">Edit</a> <a href="#">Delete</a>	Justice, Peggy	9/1/2001	2
<a href="#">Edit</a> <a href="#">Delete</a>	Li, Yan	9/1/2002	2

### 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

### 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



## FIND STUDENTS BY NAME

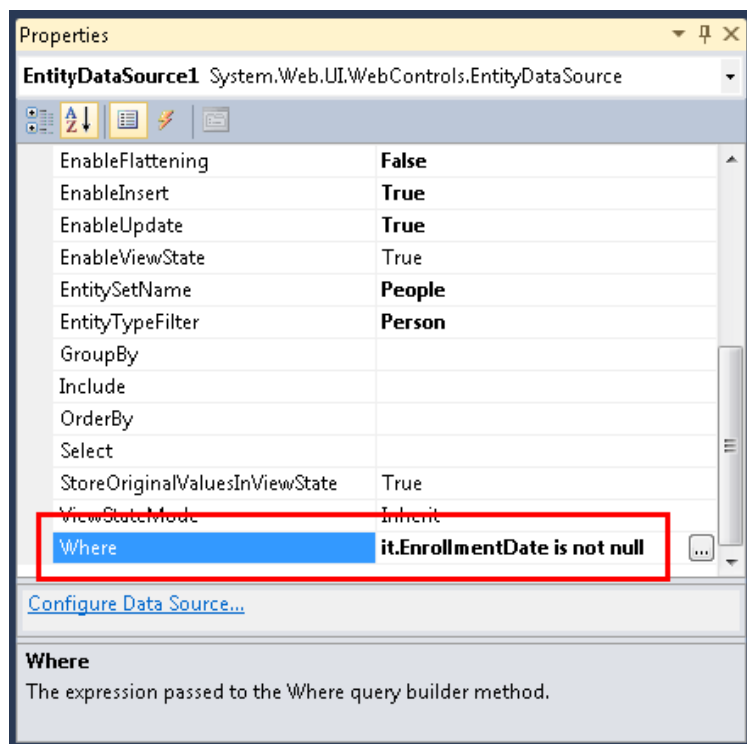
Enter any part of the name

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

## Using the EntityDataSource "Where" Property to Filter Data

Open the *Students.aspx* page that you created in the previous tutorial. As currently configured, the **GridView** control in the page displays all the names from the **People** entity set. However, you want to show only students, which you can find by selecting **Person** entities that have non-null enrollment dates.

Switch to **Design** view and select the **EntityDataSource** control. In the **Properties** window, set the **Where** property to `it.EnrollmentDate is not null`.



The syntax you use in the **Where** property of the **EntityDataSource** control is Entity SQL. Entity SQL is similar to Transact-SQL, but it's customized for use with entities rather than database objects. In the expression `it.EnrollmentDate is not null`, the word **it** represents a reference to the entity

returned by the query. Therefore, `it.EnrollmentDate` refers to the `EnrollmentDate` property of the `Person` entity that the `EntityDataSource` control returns.

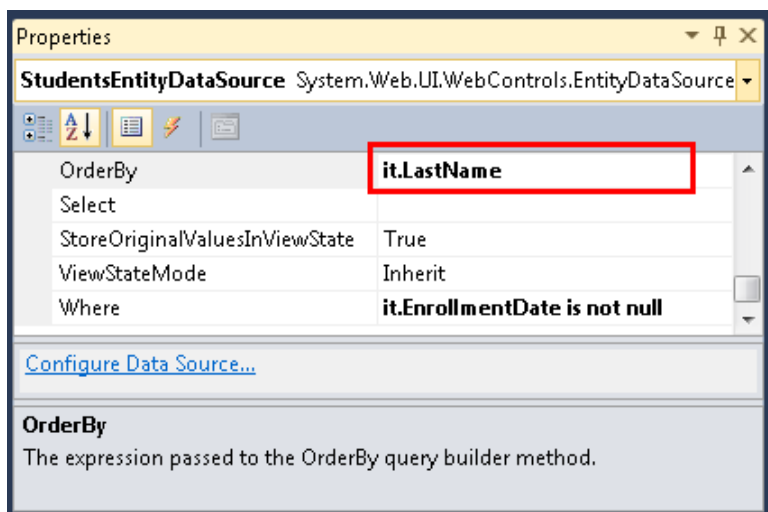
Run the page. The students list now contains only students. (There are no rows displayed where there's no enrollment date.)

### STUDENT LIST

	Name	EnrollmentDate	Number of Courses
<a href="#">Edit</a> <a href="#">Delete</a>	Barzdukas, Gytis	9/1/2005	2
<a href="#">Edit</a> <a href="#">Delete</a>	Justice, Peggy	9/1/2001	2
<a href="#">Edit</a> <a href="#">Delete</a>	Li, Yan	9/1/2002	2

## Using the EntityDataSource "OrderBy" Property to Order Data

You also want this list to be in name order when it's first displayed. With the `Students.aspx` page still open in **Design** view, and with the `EntityDataSource` control still selected, in the **Properties** window set the **OrderBy** property to `it.LastName`.



Run the page. The students list is now in order by last name.

### STUDENT LIST

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

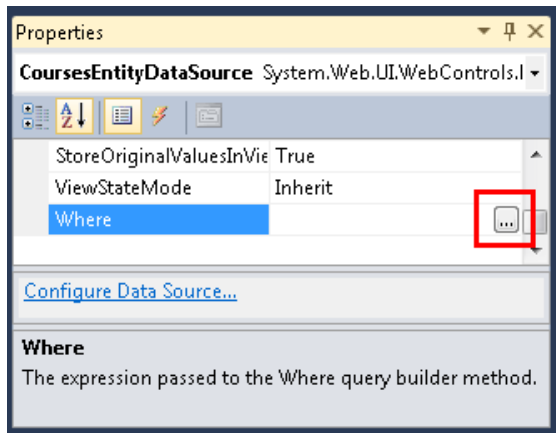
## Using a Control Parameter to Set the "Where" Property

---

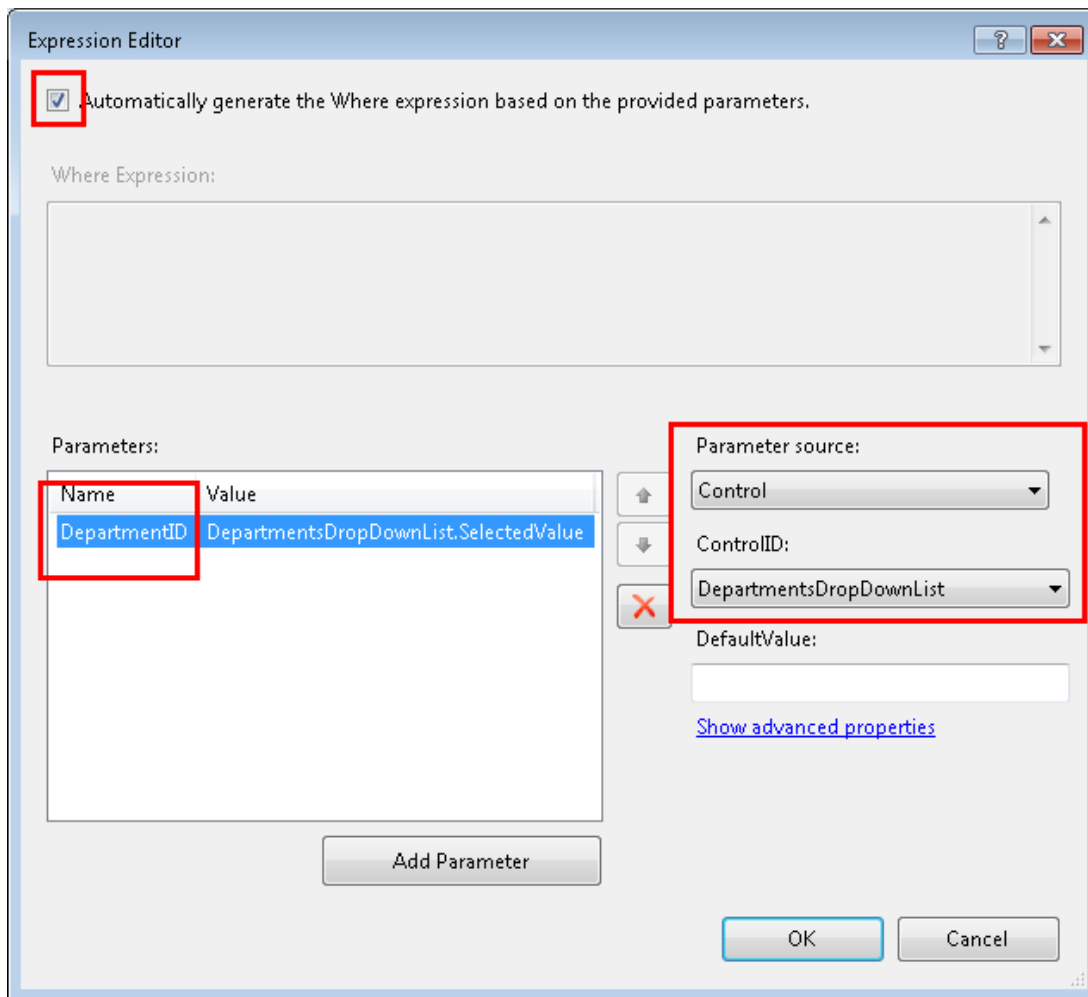
As with other data source controls, you can pass parameter values to the **Where** property. On the *Courses.aspx* page that you created in part 2 of the tutorial, you can use this method to display courses that are associated with the department that a user selects from the drop-down list.

Open *Courses.aspx* and switch to **Design** view. Add a second **EntityDataSource** control to the page, and name it **CoursesEntityDataSource**. Connect it to the **SchoolEntities** model, and select **Courses** as the **EntitySetName** value.

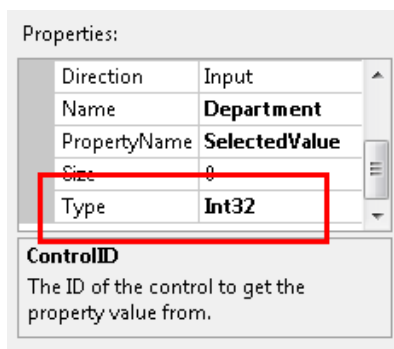
In the **Properties** window, click the ellipsis in the **Where** property box. (Make sure the **CoursesEntityDataSource** control is still selected before using the **Properties** window.)



The **Expression Editor** dialog box is displayed. In this dialog box, select **Automatically generate the Where expression based on the provided parameters**, and then click **Add Parameter**. Name the parameter **DepartmentID**, select **Control** as the **Parameter source** value, and select **DepartmentsDropDownList** as the **ControlID** value.



Click **Show advanced properties**, and in the **Properties** window of the **Expression Editor** dialog box, change the **Type** property to **Int32**.

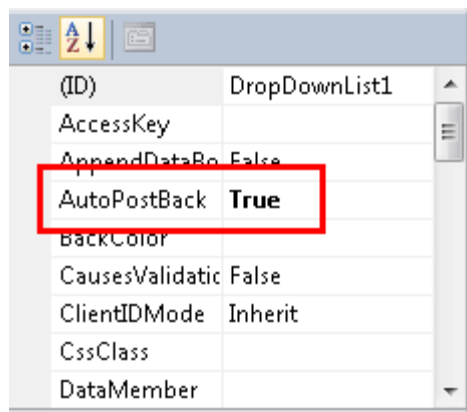


When you're done, click **OK**.

Below the drop-down list, add a **GridView** control to the page and name it **CoursesGridView**. Connect it to the **CoursesEntityDataSource** data source control, click **Refresh Schema**, click **Edit Columns**, and remove the **DepartmentID** column. The **GridView** control markup resembles the following example.

```
<asp:GridView ID="CoursesGridView" runat="server" AutoGenerateColumns="False"
    DataKeyNames="CourseID" DataSourceID="CoursesEntityDataSource">
    <Columns>
        <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" />
    </Columns>
</asp:GridView>
```

When the user changes the selected department in the drop-down list, you want the list of associated courses to change automatically. To make this happen, select the drop-down list, and in the **Properties** window set the **AutoPostBack** property to **True**.



Now that you're finished using the designer, switch to **Source** view and replace the **ConnectionString** and **DefaultContainer** name properties of the **CoursesEntityDataSource** control with the **ContextTypeName="ContosoUniversity.DAL.SchoolEntities"** attribute. When you're done, the markup for the control will look like the following example.

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

Run the page and use the drop-down list to select different departments. Only courses that are offered by the selected department are displayed in the **GridView** control.

## COURSES BY DEPARTMENT

Select a Department

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

## Using the EntityDataSource "GroupBy" Property to Group Data

---

Suppose Contoso University wants to put some student-body statistics on its About page. Specifically, it wants to show a breakdown of numbers of students by the date they enrolled.

Open *About.aspx*, and in **Source** view, replace the existing contents of the **BodyContent** control with "Student Body Statistics" between **h2** tags:

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

After the heading, add an **EntityDataSource** control and name it

**StudentStatisticsEntityDataSource**. Connect it to **SchoolEntities**, select the **People** entity set, and leave the **Select** box in the wizard unchanged. Set the following properties in the **Properties** window:

- To filter for students only, set the **Where** property to **it.EnrollmentDate is not null**.
- To group the results by the enrollment date, set the **GroupBy** property to **it.EnrollmentDate**.
- To select the enrollment date and the number of students, set the **Select** property to **it.EnrollmentDate, Count(it.EnrollmentDate) AS NumberOfStudents**.
- To order the results by the enrollment date, set the **OrderBy** property to **it.EnrollmentDate**.

In **Source** view, replace the **ConnectionString** and **DefaultContainer** name properties with a **ContextTypeName** property. The **EntityDataSource** control markup now resembles the following example.

```
<asp:EntityDataSource ID="StudentStatisticsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
    EntitySetName="People"
    Select="it.EnrollmentDate, Count(it.EnrollmentDate) AS NumberOfStudents"
```

```

        OrderBy="it.EnrollmentDate" GroupBy="it.EnrollmentDate"
        Where="it.EnrollmentDate is not null" >
</asp:EntityDataSource>

```

The syntax of the **Select**, **GroupBy**, and **Where** properties resembles Transact-SQL except for the **it** keyword that specifies the current entity.

Add the following markup to create a **GridView** control to display the data.

```

<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>

```

Run the page to see a list showing the number of students by enrollment date.

## 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

## Using the QueryExtender Control for Filtering and Ordering

The **QueryExtender** control provides a way to specify filtering and sorting in markup. The syntax is independent of the database management system (DBMS) you're using. It's also generally independent of the Entity Framework, with the exception that syntax you use for navigation properties is unique to the Entity Framework.

In this part of the tutorial you'll use a **QueryExtender** control to filter and order data, and one of the order-by fields will be a navigation property.

(If you prefer to use code instead of markup to extend the queries that are automatically generated by the **EntityDataSource** control, you can do that by handling the **QueryCreated** event. This is how the **QueryExtender** control extends **EntityDataSource** control queries also.)

Open the *Courses.aspx* page, and below the markup you added previously, insert the following markup to create a heading, a text box for entering search strings, a search button, and an **EntityDataSource** control that's bound to the **Courses** entity set.

```
<h2>Courses by Name</h2>
Enter a course 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="Courses"
    Include="Department" >
</asp:EntityDataSource>
```

Notice that the **EntityDataSource** control's **Include** property is set to **Department**. In the database, the **Course** table does not contain the department name; it contains a **DepartmentID** foreign key column. If you were querying the database directly, to get the department name along with course data, you would have to join the **Course** and **Department** tables. By setting the **Include** property to **Department**, you specify that the Entity Framework should do the work of getting the related **Department** entity when it gets a **Course** entity. The **Department** entity is then stored in the **Department** navigation property of the **Course** entity. (By default, the **SchoolEntities** class that was generated by the data model designer retrieves related data when it's needed, and you've bound the data source control to that class, so setting the **Include** property is not necessary. However, setting it improves performance of the page, because otherwise the Entity Framework would make separate calls to the database to retrieve data for the **Course** entities and for the related **Department** entities.)

After the **EntityDataSource** control you just created, insert the following markup to create a **QueryExtender** control that's bound to that **EntityDataSource** control.

```
<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>
```



```
</asp:OrderByExpression>
</asp:QueryExtender>
```

The **SearchExpression** element specifies that you want to select courses whose titles match the value entered in the text box. Only as many characters as are entered in the text box will be compared, because the **SearchType** property specifies **StartsWith**.

The **OrderByExpression** element specifies that the result set will be ordered by course title within department name. Notice how department name is specified: **Department.Name**. Because the association between the **Course** entity and the **Department** entity is one-to-one, the **Department** navigation property contains a **Department** entity. (If this were a one-to-many relationship, the property would contain a collection.) To get the department name, you must specify the **Name** property of the **Department** entity.

Finally, add a **GridView** control to display the list of courses:

```
<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>
```

The first column is a template field that displays the department name. The databinding expression specifies **Department.Name**, just as you saw in the **QueryExtender** control.

Run the page. The initial display shows a list of all courses in order by department and then by course title.

### 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

Enter an "m" and click **Search** to see all courses whose titles begin with "m" (the search is not case sensitive).

### 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

## Using the "Like" Operator to Filter Data

You can achieve an effect similar to the **QueryExtender** control's **StartsWith**, **Contains**, and **EndsWith** search types by using a **Like** operator in the **EntityDataSource** control's **Where** property. In this part of the tutorial, you'll see how to use the **Like** operator to search for a student by name.

Open *Students.aspx* in **Source** view. After the **GridView** control, add the following markup:

```
<h2>Find Students by Name</h2>
```

Enter any part of the name

```
<asp:TextBox ID="SearchTextBox" runat="server" AutoPostBack="true"></asp:TextBox>
```

```

    &nbsp;<asp:Button ID="SearchButton" runat="server" Text="Search" />
    <br />
    <br />
    <asp:EntityDataSource ID="SearchEntityDataSource" runat="server"
        ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
        EnableFlattening="False"
        EntitySetName="People"
        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>

```

This markup is similar to what you've seen earlier except for the **Where** property value. The second part of the **Where** expression defines a substring search (**LIKE %FirstMidName% or LIKE %LastName%**) that searches both the first and last names for whatever is entered in the text box.

Run the page. Initially you see all of the students because the default value for the **StudentName** parameter is "%".

### 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

Enter the letter "g" in the text box and click **Search**. You see a list of students that have a "g" in either the first or last name.

### 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

You've now displayed, updated, filtered, ordered, and grouped data from individual tables. In the next tutorial you'll begin to work with related data (master-detail scenarios).

## Part 4: Working with Related Data

In the previous tutorial you used the **EntityDataSource** control to filter, sort, and group data. In this tutorial you'll display and update related data.

You'll create the Instructors page that shows a list of instructors. When you select an instructor, you see a list of courses taught by that instructor. When you select a course, you see details for the course and a list of students enrolled in the course. You can edit the instructor name, hire date, and office assignment. The office assignment is a separate entity set that you access through a navigation property.

You can link master data to detail data in markup or in code. In this part of the tutorial, you'll use both methods.

### INSTRUCTORS

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

### COURSES TAUGHT

	ID	Title	Department
<a href="#">Select</a>	2030	Poetry	English

### COURSE DETAILS

ID	2030
Title	Poetry
Credits	2
Department	English
Location	
URL	<a href="http://www.fineartschool.n">http://www.fineartschool.n</a>

### STUDENT GRADES

Name	Grade
Barzdukas, Gytis	3.50
Justice, Peggy	4.00

## Displaying and Updating Related Entities in a GridView Control

Create a new web page named *Instructors.aspx* that uses the *Site.Master* master page, and add the following markup to the **Content** control named **Content2**:

```
<h2>Instructors</h2>
<div style="float: left; margin-right: 20px;">
  <asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
    EntitySetName="People"
    Where="it.HireDate is not null" Include="OfficeAssignment" EnableUpdate="True">
  </asp:EntityDataSource>
</div>
```

This markup creates an **EntityDataSource** control that selects instructors and enables updates. The **div** element configures markup to render on the left so that you can add a column on the right later.

Between the **EntityDataSource** markup and the closing **</div>** tag, add the following markup that creates a **GridView** control and a **Label** control that you'll use for error messages:

```
<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>

```

This **GridView** control enables row selection, highlights the selected row with a light gray background color, and specifies handlers (which you'll create later) for the **SelectedIndexChanged** and **Updating** events. It also specifies **PersonID** for the **DataKeyNames** property, so that the key value of the selected row can be passed to another control that you'll add later.

The last column contains the instructor's office assignment, which is stored in a navigation property of the **Person** entity because it comes from an associated entity. Notice that the **EditItemTemplate** element specifies **Eval** instead of **Bind**, because the **GridView** control cannot directly bind to navigation properties in order to update them. You'll update the office assignment in code. To do that, you'll need a reference to the **TextBox** control, and you'll get and save that in the **TextBox** control's **Init** event.

Following the **GridView** control is a **Label** control that's used for error messages. The control's **Visible** property is **false**, and view state is turned off, so that the label will appear only when code makes it visible in response to an error.

Open the *Instructors.aspx.cs* file and add the following **using** statement:

```
using ContosoUniversity.DAL;
```

Add a private class field immediately after the partial-class name declaration to hold a reference to the office assignment text box.

```
private TextBox instructorOfficeTextBox;
```

Add a stub for the **SelectedIndexChanged** event handler that you'll add code for later. Also add a handler for the office assignment **TextBox** control's **Init** event so that you can store a reference to the **TextBox** control. You'll use this reference to get the value the user entered in order to update the entity associated with the navigation property.

```

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

```

```

        instructorOfficeTextBox = sender as TextBox;
    }

```

You'll use the **GridView** control's **Updating** event to update the **Location** property of the associated **OfficeAssignment** entity. Add the following handler for the **Updating** event:

```

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.OfficeAssignments
                                where o.InstructorID == instructorBeingUpdated
                                select o).FirstOrDefault();

        try
        {
            if (String.IsNullOrEmpty(instructorOfficeTextBox.Text) == false)
            {
                if (officeAssignment == null)
                {
                    context.OfficeAssignments.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.
        }
    }
}

```



```

    }
}
}

```

This code is run when the user clicks **Update** in a **GridView** row. The code uses LINQ to Entities to retrieve the **OfficeAssignment** entity that's associated with the current **Person** entity, using the **PersonID** of the selected row from the event argument.

The code then takes one of the following actions depending on the value in the **InstructorOfficeTextBox** control:

- If the text box has a value and there's no **OfficeAssignment** entity to update, it creates one.
- If the text box has a value and there's an **OfficeAssignment** entity, it updates the **Location** property value.
- If the text box is empty and an **OfficeAssignment** entity exists, it deletes the entity.

After this, it saves the changes to the database. If an exception occurs, it displays an error message.

Run the page.

#### INSTRUCTORS

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

Click **Edit** and all fields change to text boxes.

#### INSTRUCTORS

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

Change any of these values, including **Office Assignment**. Click **Update** and you'll see the changes reflected in the list.

## Displaying Related Entities in a Separate Control

---

Each instructor can teach one or more courses, so you'll add an **EntityDataSource** control and a **GridView** control to list the courses associated with whichever instructor is selected in the **InstructorsGridView** control. To create a heading and the **EntityDataSource** control for courses entities, add the following markup between the error message **Label** control and the closing **</div>** tag:

```
<h3>Courses Taught</h3>
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="Courses"
    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>
```

The **Where** parameter contains the value of the **PersonID** of the instructor whose row is selected in the **InstructorsGridView** control. The **Where** property contains a subselect command that gets all associated **Person** entities from a **Course** entity's **People** navigation property and selects the **Course** entity only if one of the associated **Person** entities contains the selected **PersonID** value.

To create the **GridView** control, add the following markup immediately following the **CoursesEntityDataSource** control (before the closing **</div>** tag):

```
<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" />
    </Columns>
</asp:GridView>
```

```

<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>

```

Because no courses will be displayed if no instructor is selected, an **EmptyDataTemplate** element is included.

Run the page.

## INSTRUCTORS

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

## COURSES TAUGHT

No courses found.

Select an instructor who has one or more courses assigned, and the course or courses appear in the list. (Note: although the database schema allows multiple courses, in the test data supplied with the database no instructor actually has more than one course. You can add courses to the database yourself using the **Server Explorer** window or the *CoursesAdd.aspx* page, which you'll add in a later tutorial.)

## INSTRUCTORS

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

## COURSES TAUGHT

	ID	Title	Department
<a href="#">Select</a>	2030	Poetry	English

The **CoursesGridView** control shows only a few course fields. To display all the details for a course, you'll use a **DetailsView** control for the course that the user selects. In *Instructors.aspx*, add the following markup after the closing `</div>` tag (make sure you place this markup **after** the closing div tag, not before it):

```
<div>
  <h3>Course Details</h3>
  <asp:EntityDataSource ID="CourseDetailsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
    EntitySetName="Courses"
    AutoGenerateWhereClause="False" Where="it.CourseID = @CourseID"
    Include="Department,OnlineCourse,OnsiteCourse,StudentGrades.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>

```

This markup creates an **EntityDataSource** control that's bound to the **Courses** entity set. The **Where** property selects a course using the **CourseID** value of the selected row in the **courses GridView** control. The markup specifies a handler for the **Selected** event, which you'll use later for displaying student grades, which is another level lower in the hierarchy.

In *Instructors.aspx.cs*, create the following stub for the **CourseDetailsEntityDataSource\_Selected** method. (You'll fill this stub out later in the tutorial; for now, you need it so that the page will compile and run.)

```

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

```

Run the page.

## INSTRUCTORS

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

## COURSE DETAILS

No course selected.

## COURSES TAUGHT

No courses found.

Initially there are no course details because no course is selected. Select an instructor who has a course assigned, and then select a course to see the details.

## INSTRUCTORS

	Name	Hire Date	Office Assignment
<a href="#">Edit</a> <a href="#">Select</a>	Abercrombie, Kim	3/11/1995	17 Smith
<a href="#">Edit</a> <a href="#">Select</a>	Fakhouri, Fadi	8/6/2002	29 Adams
<a href="#">Edit</a> <a href="#">Select</a>	Harui, Roger	7/1/1998	37 Williams
<a href="#">Edit</a> <a href="#">Select</a>	Zheng, Roger	2/12/2004	143 Smith
<a href="#">Edit</a> <a href="#">Select</a>	Kapoor, Candace	1/15/2001	57 Adams
<a href="#">Edit</a> <a href="#">Select</a>	Serrano, Stacy	6/1/1999	271 Williams
<a href="#">Edit</a> <a href="#">Select</a>	Stewart, Jasmine	10/12/1997	131 Smith
<a href="#">Edit</a> <a href="#">Select</a>	Xu, Kristen	7/23/2001	203 Williams
<a href="#">Edit</a> <a href="#">Select</a>	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
<a href="#">Select</a>	2030	Poetry	English

## Using the EntityDataSource "Selected" Event to Display Related Data

Finally, you want to show all of the enrolled students and their grades for the selected course. To do this, you'll use the **Selected** event of the **EntityDataSource** control bound to the course **DetailsView**.

In *Instructors.aspx*, add the following markup after the **DetailsView** control:

```
<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>
    </table>
</LayoutTemplate>
<ItemTemplate>
    <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>

```

This markup creates a **ListView** control that displays a list of students and their grades for the selected course. No data source is specified because you'll databind the control in code. The **EmptyDataTemplate** element provides a message to display when no course is selected—in that case, there are no students to display. The **LayoutTemplate** element creates an HTML table to display the list, and the **ItemTemplate** specifies the columns to display. The student ID and the student grade are from the **StudentGrade** entity, and the student name is from the **Person** entity that the Entity Framework makes available in the **Person** navigation property of the **StudentGrade** entity.

In *Instructors.aspx.cs*, replace the stubbed-out **CourseDetailsEntityDataSource\_Selected** method with the following code:

```
protected void CourseDetailsEntityDataSource_Selected(object sender,
```

```

        EntityDataSourceSelectedEventArgs e)
    {
        var course = e.Results.Cast<Course>().FirstOrDefault();
        if (course != null)
        {
            var studentGrades = course.StudentGrades.ToList();
            GradesListView.DataSource = studentGrades;
            GradesListView.DataBind();
        }
    }
}

```

The event argument for this event provides the selected data in the form of a collection, which will have zero items if nothing is selected or one item if a **Course** entity is selected. If a **Course** entity is selected, the code uses the **First** method to convert the collection to a single object. It then gets **StudentGrade** entities from the navigation property, converts them to a collection, and binds the **GradesListView** control to the collection.

This is sufficient to display grades, but you want to make sure that the message in the empty data template is displayed the first time the page is displayed and whenever a course is not selected. To do that, create the following method, which you'll call from two places:

```

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

```

Call this new method from the **Page\_Load** method to display the empty data template the first time the page is displayed. And call it from the **InstructorsGridView\_SelectedIndexChanged** method because that event is raised when an instructor is selected, which means new courses are loaded into the **courses GridView** control and none is selected yet. Here are the two calls:

```

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

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

```



Run the page.

#### INSTRUCTORS

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

#### COURSE DETAILS

No course selected.

#### STUDENT GRADES

No student grades found.

#### COURSES TAUGHT

No courses found.

Select an instructor that has a course assigned, and then select the course.

#### INSTRUCTORS

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

#### 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

#### COURSES TAUGHT

	ID	Title	Department
<a href="#">Select</a>	2030	Poetry	English

You have now seen a few ways to work with related data. In the following tutorial, you'll learn how to add relationships between existing entities, how to remove relationships, and how to add a new entity that has a relationship to an existing entity.

## Part 5: Working with Related Data, Continued

In the previous tutorial you began to use the `EntityDataSource` control to work with related data. You displayed multiple levels of hierarchy and edited data in navigation properties. In this tutorial you'll continue to work with related data by adding and deleting relationships and by adding a new entity that has a relationship to an existing entity.

You'll create a page that adds courses that are assigned to departments. The departments already exist, and when you create a new course, at the same time you'll establish a relationship between it and an existing department.

### ADD COURSES

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

You'll also create a page that works with a many-to-many relationship by assigning an instructor to a course (adding a relationship between two entities that you select) or removing an instructor from a course (removing a relationship between two entities that you select). In the database, adding a relationship between an instructor and a course results in a new row being added to the `CourseInstructor` association table; removing a relationship involves deleting a row from the `CourseInstructor` association table. However, you do this in the Entity Framework by setting navigation properties, without referring to the `CourseInstructor` table explicitly.

### ASSIGN INSTRUCTORS TO COURSES OR REMOVE FROM COURSES

Select an Instructor:  ▼

#### ASSIGN A COURSE

Select a Course:  ▼

#### REMOVE A COURSE

Select a Course:  ▼

## Adding an Entity with a Relationship to an Existing Entity

---

Create a new web page named *CoursesAdd.aspx* that uses the *Site.Master* master page, and add the following markup to the **Content** control named **Content2**:

```
<h2>Add Courses</h2>
<asp:EntityDataSource ID="CoursesEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="Courses"
    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>
```

This markup creates an **EntityDataSource** control that selects courses, that enables inserting, and that specifies a handler for the **Inserting** event. You'll use the handler to update the **Department** navigation property when a new **Course** entity is created.

The markup also creates a **DetailsView** control to use for adding new **Course** entities. The markup uses bound fields for **Course** entity properties. You have to enter the **CourseID** value because this is not a

system-generated ID field. Instead, it's a course number that must be specified manually when the course is created.

You use a template field for the `Department` navigation property because navigation properties cannot be used with `BoundField` controls. The template field provides a drop-down list to select the department. The drop-down list is bound to the `Departments` entity set by using `Eval` rather than `Bind`, again because you cannot directly bind navigation properties in order to update them. You specify a handler for the `DropDownList` control's `Init` event so that you can store a reference to the control for use by the code that updates the `DepartmentID` foreign key.

In *CoursesAdd.aspx.cs* just after the partial-class declaration, add a class field to hold a reference to the `DepartmentsDropDownList` control:

```
private DropDownList departmentDropDownList;
```

Add a handler for the `DepartmentsDropDownList` control's `Init` event so that you can store a reference to the control. This lets you get the value the user has entered and use it to update the `DepartmentID` value of the `Course` entity.

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

Add a handler for the `DetailsView` control's `Inserting` event:

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

When the user clicks `Insert`, the `Inserting` event is raised before the new record is inserted. The code in the handler gets the `DepartmentID` from the `DropDownList` control and uses it to set the value that will be used for the `DepartmentID` property of the `Course` entity.

The Entity Framework will take care of adding this course to the `Courses` navigation property of the associated `Department` entity. It also adds the department to the `Department` navigation property of the `Course` entity.

Run the page.

### ADD COURSES

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

Enter an ID, a title, a number of credits, and select a department, then click **Insert**.

Run the *Courses.aspx* page, and select the same department to see the new course.

### COURSES BY DEPARTMENT

Select a Department  ▼

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

## Working with Many-to-Many Relationships

---

The relationship between the **Courses** entity set and the **People** entity set is a many-to-many relationship. A **Course** entity has a navigation property named **People** that can contain zero, one, or more related **Person** entities (representing instructors assigned to teach that course). And a **Person** entity has a navigation property named **Courses** that can contain zero, one, or more related **Course** entities (representing courses that that instructor is assigned to teach). One instructor might teach multiple courses, and one course might be taught by multiple instructors. In this section of the walkthrough, you'll add and remove relationships between **Person** and **Course** entities by updating the navigation properties of the related entities.

Create a new web page named *InstructorsCourses.aspx* that uses the *Site.Master* master page, and add the following markup to the **Content** control named **Content2**:

```
<h2>Assign Instructors to Courses or Remove from Courses</h2>
<br />
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="People">
```

```

        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="Assignment
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="Removal successful"></asp:Label>

```

This markup creates an **EntityDataSource** control that retrieves the name and **PersonID** of **Person** entities for instructors. A **DropDrownList** control is bound to the **EntityDataSource** control. The **DropDownList** control specifies a handler for the **DataBound** event. You'll use this handler to databind the two drop-down lists that display courses.

The markup also creates the following group of controls to use for assigning a course to the selected instructor:

- A **DropDownList** control for selecting a course to assign. This control will be populated with courses that are currently not assigned to the selected instructor.

- A **Button** control to initiate the assignment.
- A **Label** control to display an error message if the assignment fails.

Finally, the markup also creates a group of controls to use for removing a course from the selected instructor.

In *InstructorsCourses.aspx.cs*, add a using statement:

```
using ContosoUniversity.DAL;
```

Add a method for populating the two drop-down lists that display courses:

```
private void PopulateDropDownLists()
{
    using (var context = new SchoolEntities())
    {
        var allCourses = (from c in context.Courses
                        select c).ToList();

        var instructorID = Convert.ToInt32(InstructorsDropDownList.SelectedValue);
        var instructor = (from p in context.People.Include("Courses")
                        where p.PersonID == instructorID
                        select p).First();

        var assignedCourses = instructor.Courses.ToList();
        var unassignedCourses =
            allCourses.Except(assignedCourses.AsEnumerable()).ToList();

        UnassignedCoursesDropDownList.DataSource = unassignedCourses;
        UnassignedCoursesDropDownList.DataBind();
        UnassignedCoursesDropDownList.Visible = true;

        AssignedCoursesDropDownList.DataSource = assignedCourses;
        AssignedCoursesDropDownList.DataBind();
        AssignedCoursesDropDownList.Visible = true;
    }
}
```

This code gets all courses from the **Courses** entity set and gets the courses from the **Courses** navigation property of the **Person** entity for the selected instructor. It then determines which courses are assigned to that instructor and populates the drop-down lists accordingly.

Add a handler for the **Assign** button's **Click** event:

```
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.People
                          where p.PersonID == instructorID
                          select p).First();
        var courseID = Convert.ToInt32(UnassignedCoursesDropDownList.SelectedValue);
        var course = (from c in context.Courses
                      where c.CourseID == courseID
                      select c).First();
        instructor.Courses.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;
    }
}

```

This code gets the **Person** entity for the selected instructor, gets the **Course** entity for the selected course, and adds the selected course to the **Courses** navigation property of the instructor's **Person** entity. It then saves the changes to the database and repopulates the drop-down lists so the results can be seen immediately.

Add a handler for the **Remove** button's **Click** event:

```

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.People
                          where p.PersonID == instructorID
                          select p).First();
        var courseID = Convert.ToInt32(AssignedCoursesDropDownList.SelectedValue);
        var courses = instructor.Courses;
        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;
}
}

```

This code gets the **Person** entity for the selected instructor, gets the **Course** entity for the selected course, and removes the selected course from the **Person** entity's **Courses** navigation property. It then saves the changes to the database and repopulates the drop-down lists so the results can be seen immediately.

Add code to the **Page\_Load** method that makes sure the error messages are not visible when there's no error to report, and add handlers for the **DataBound** and **SelectedIndexChanged** events of the instructors drop-down list to populate the courses drop-down lists:

```

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();  
    }
```

Run the page.

### ASSIGN INSTRUCTORS TO COURSES OR REMOVE FROM COURSES

Select an Instructor:

#### ASSIGN A COURSE

Select a Course:

#### REMOVE A COURSE

Select a Course:

Select an instructor. The **Assign a Course** drop-down list displays the courses that the instructor doesn't teach, and the **Remove a Course** drop-down list displays the courses that the instructor is already assigned to. In the **Assign a Course** section, select a course and then click **Assign**. The course moves to the **Remove a Course** drop-down list. Select a course in the **Remove a Course** section and click **Remove**. The course moves to the **Assign a Course** drop-down list.

You have now seen some more ways to work with related data. In the following tutorial, you'll learn how to use inheritance in the data model to improve the maintainability of your application.

## Part 6: Implementing Table-per-Hierarchy Inheritance

In the previous tutorial you worked with related data by adding and deleting relationships and by adding a new entity that had a relationship to an existing entity. This tutorial will show you how to implement inheritance in the data model.

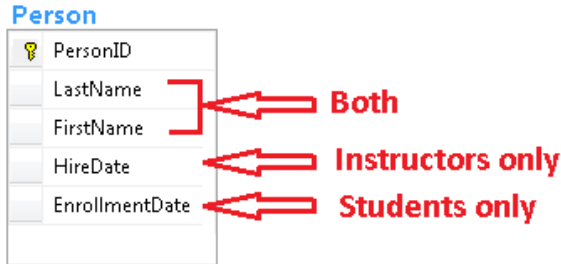
In object-oriented programming, you can use inheritance to make it easier to work with related classes. For example, you could create **Instructor** and **Student** classes that derive from a **Person** base class. You can create the same kinds of inheritance structures among entities in the Entity Framework.

In this part of the tutorial, you won't create any new web pages. Instead, you'll add derived entities to the data model and modify existing pages to use the new entities.

### Table-per-Hierarchy versus Table-per-Type Inheritance

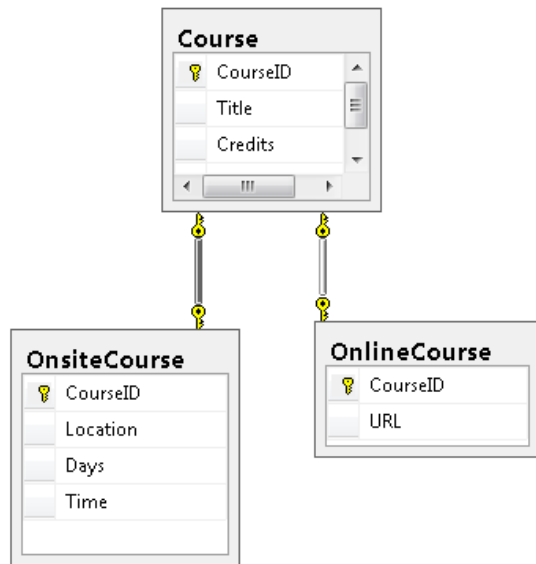
---

A database can store information about related objects in one table or in multiple tables. For example, in the **School** database, the **Person** table includes information about both students and instructors in a single table. Some of the columns apply only to instructors (**HireDate**), some only to students (**EnrollmentDate**), and some to both (**LastName**, **FirstName**).



You can configure the Entity Framework to create **Instructor** and **Student** entities that inherit from the **Person** entity. This pattern of generating an entity inheritance structure from a single database table is called *table-per-hierarchy* (TPH) inheritance.

For courses, the **School** database uses a different pattern. Online courses and onsite courses are stored in separate tables, each of which has a foreign key that points to the **Course** table. Information common to both course types is stored only in the **Course** table.



You can configure the Entity Framework data model so that **OnlineCourse** and **OnsiteCourse** entities inherit from the **Course** entity. This pattern of generating an entity inheritance structure from separate tables for each type, with each separate table referring back to a table that stores data common to all types, is called *table per type* (TPT) inheritance.

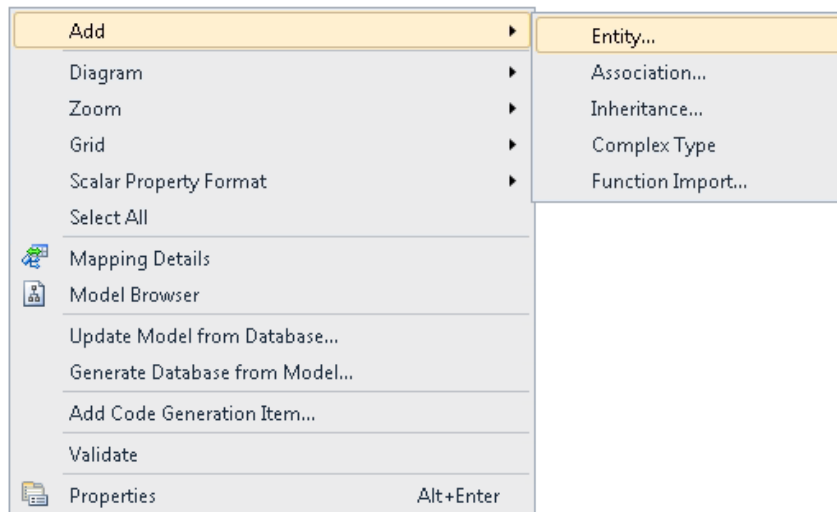
TPH inheritance patterns generally deliver better performance in the Entity Framework than TPT inheritance patterns, because TPT patterns can result in complex join queries. This walkthrough demonstrates how to implement TPH inheritance. You'll do that by performing the following steps:

- Create **Instructor** and **Student** entity types that derive from **Person**.
- Move properties that pertain to the derived entities from the **Person** entity to the derived entities.
- Set constraints on properties in the derived types.
- Make the **Person** entity an abstract entity.
- Map each derived entity to the **Person** table with a condition that specifies how to determine whether a **Person** row represents that derived type.

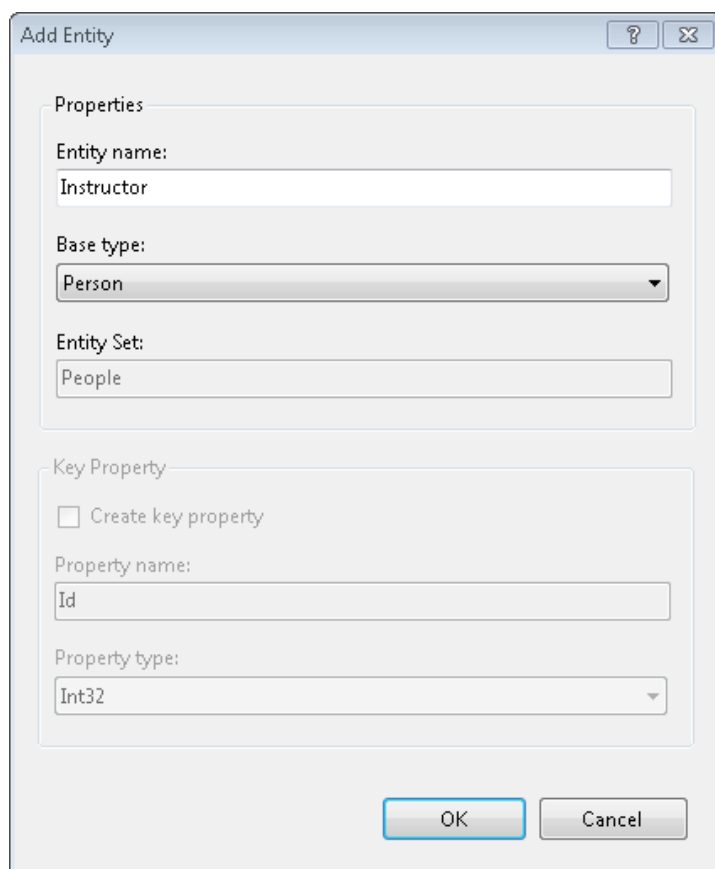
## Adding Instructor and Student Entities

---

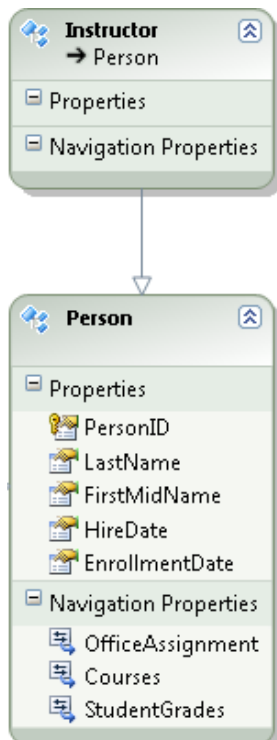
Open the *SchoolModel.edmx* file, right-click an unoccupied area in the designer, select **Add**, then select **Entity**.



In the **Add Entity** dialog box, name the entity **Instructor** and set its **Base type** option to **Person**.

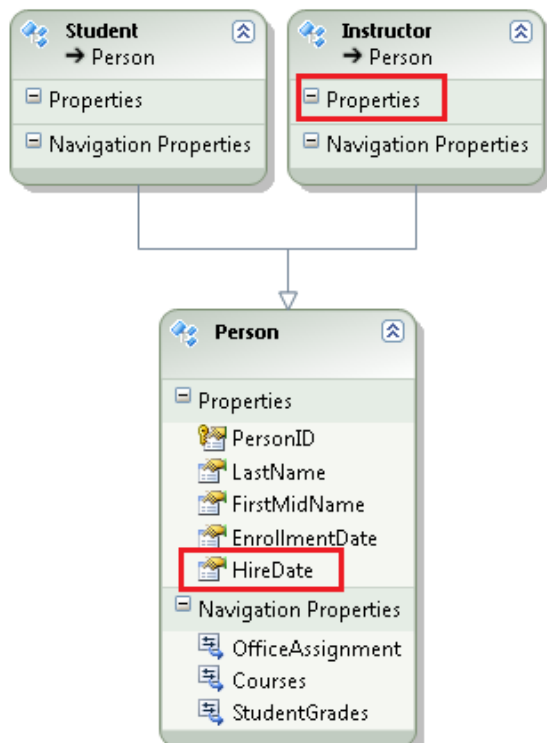


Click **OK**. The designer creates an **Instructor** entity that derives from the **Person** entity. The new entity does not yet have any properties.

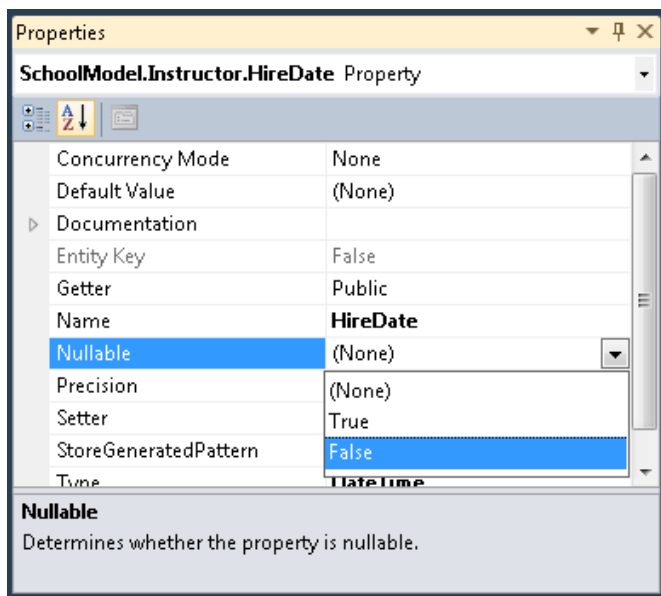


Repeat the procedure to create a **Student** entity that also derives from **Person**.

Only instructors have hire dates, so you need to move that property from the **Person** entity to the **Instructor** entity. In the **Person** entity, right-click the **HireDate** property and click **Cut**. Then right-click **Properties** in the **Instructor** entity and click **Paste**.

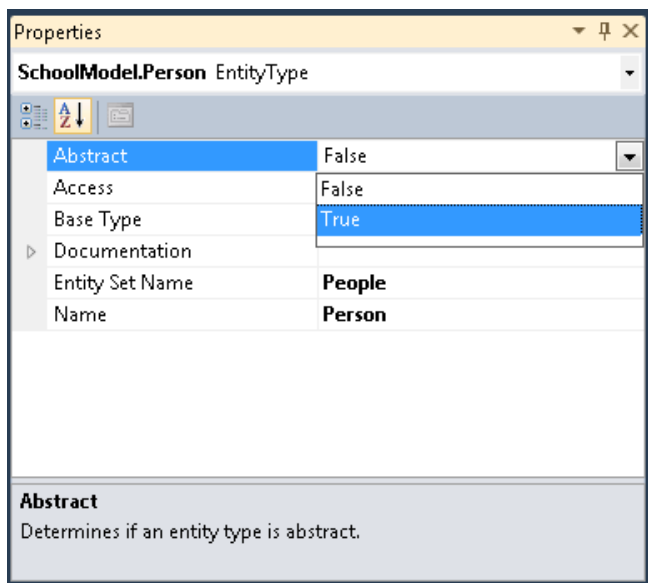


The hire date of an **Instructor** entity cannot be null. Right-click the **HireDate** property, click **Properties**, and then in the **Properties** window change **Nullable** to **False**.



Repeat the procedure to move the **EnrollmentDate** property from the **Person** entity to the **Student** entity. Make sure that you also set **Nullable** to **False** for the **EnrollmentDate** property.

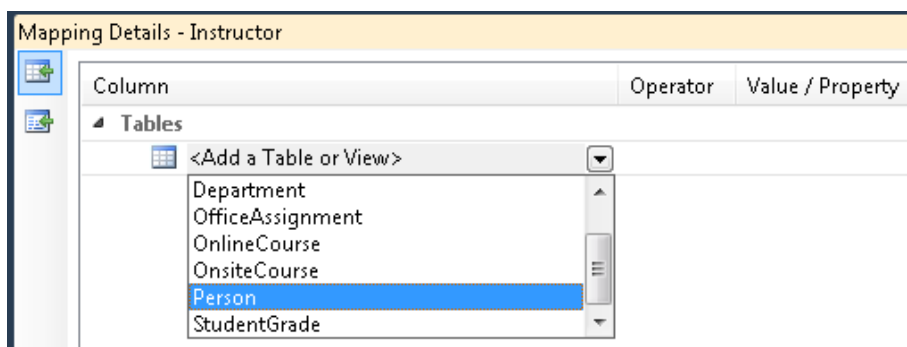
Now that a **Person** entity has only the properties that are common to **Instructor** and **Student** entities (aside from navigation properties, which you're not moving), the entity can only be used as a base entity in the inheritance structure. Therefore, you need to ensure that it's never treated as an independent entity. Right-click the **Person** entity, select **Properties**, and then in the **Properties** window change the value of the **Abstract** property to **True**.



## Mapping Instructor and Student Entities to the Person Table

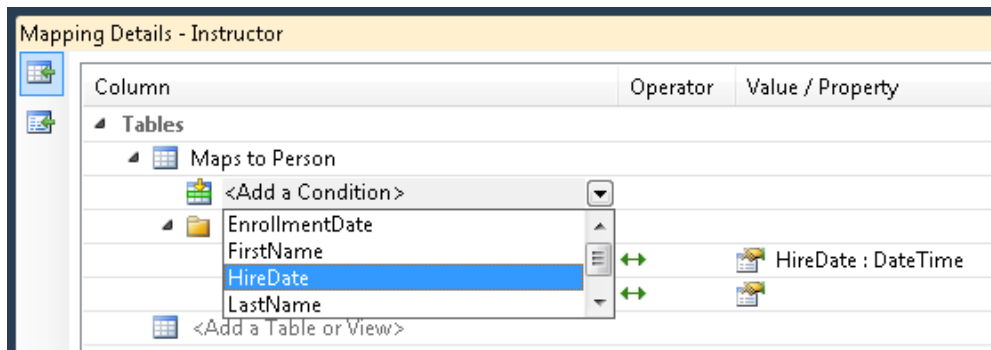
Now you need to tell the Entity Framework how to differentiate between **Instructor** and **Student** entities in the database.

Right-click the **Instructor** entity and select **Table Mapping**. In the **Mapping Details** window, click **Add a Table or View** and select **Person**.

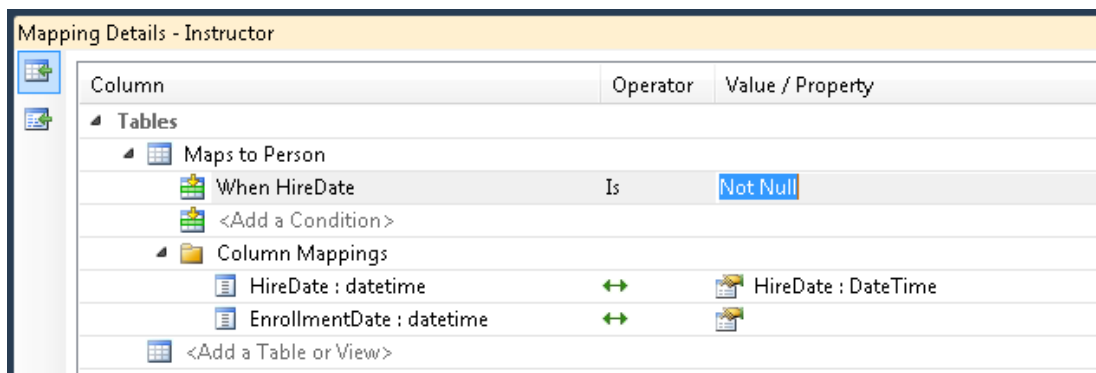


Click **Add a Condition**, and then select **HireDate**.





Change **Operator** to **Is** and **Value / Property** to **Not Null**.



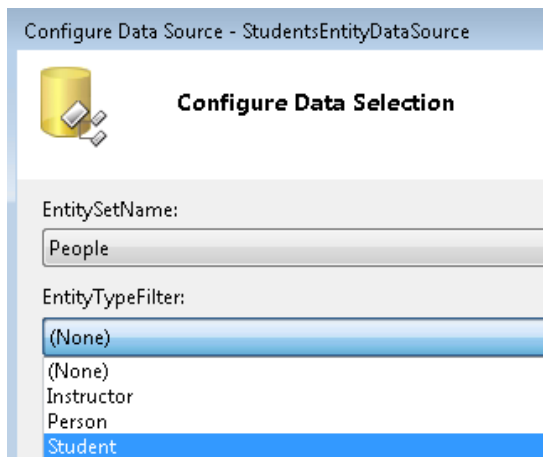
Repeat the procedure for the **Students** entity, specifying that this entity maps to the **Person** table when the **EnrollmentDate** column is not null. Then save and close the data model.

Build the project in order to create the new entities as classes and make them available in the designer.

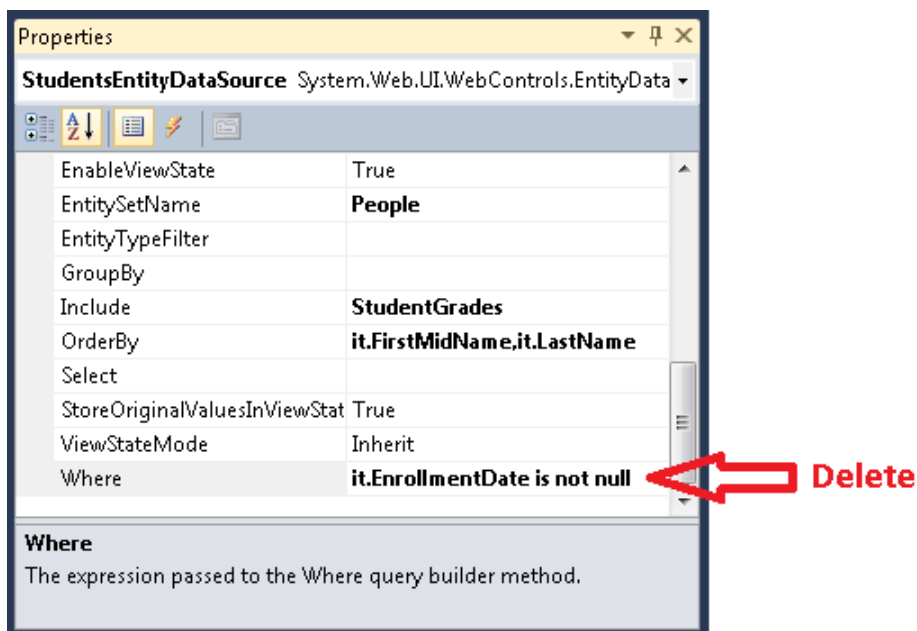
## Using the Instructor and Student Entities

When you created the web pages that work with student and instructor data, you databound them to the **Person** entity set, and you filtered on the **HireDate** or **EnrollmentDate** property to restrict the returned data to students or instructors. However, now when you bind each data source control to the **Person** entity set, you can specify that only **Student** or **Instructor** entity types should be selected. Because the Entity Framework knows how to differentiate students and instructors in the **Person** entity set, you can remove the **Where** property settings you entered manually to do that.

In the Visual Studio Designer, you can specify the entity type that an **EntityDataSource** control should select in the **EntityTypeFilter** drop-down box of the **Configure Data Source** wizard, as shown in the following example.



And in the **Properties** window you can remove **Where** clause values that are no longer needed, as shown in the following example.



However, because you've changed the markup for **EntityDataSource** controls to use the **ContextTypeName** attribute, you cannot run the **Configure Data Source** wizard on **EntityDataSource** controls that you've already created. Therefore, you'll make the required changes by changing markup instead.

Open the *Students.aspx* page. In the **StudentsEntityDataSource** control, remove the **Where** attribute and add an **EntityTypeFilter="Student"** attribute. The markup will now resemble the following example:

```
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities" EnableFlattening="False"
```

```

        EntitySetName="People" EntityTypeFilter="Student"
        Include="StudentGrades"
        EnableDelete="True" EnableUpdate="True"
        OrderBy="it.LastName" >
</asp:EntityDataSource>

```

Setting the **EntityTypeFilter** attribute ensures that the **EntityDataSource** control will select only the specified entity type. If you wanted to retrieve both **Student** and **Instructor** entity types, you would not set this attribute. (You have the option of retrieving multiple entity types with one **EntityDataSource** control only if you're using the control for read-only data access. If you're using an **EntityDataSource** control to insert, update, or delete entities, and if the entity set it's bound to can contain multiple types, you can only work with one entity type, and you have to set this attribute.)

Repeat the procedure for the **SearchEntityDataSource** control, except remove only the part of the **Where** attribute that selects **Student** entities instead of removing the property altogether. The opening tag of the control will now resemble the following example:

```

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

```

Run the page to verify that it still works as it did before.

## STUDENT LIST

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

## 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 <a href="#">2</a> <a href="#">3</a>	

Update the following pages that you created in earlier tutorials so that they use the new **Student** and **Instructor** entities instead of **Person** entities, then run them to verify that they work as they did before:

- In *StudentsAdd.aspx*, add **EntityTypeFilter="Student"** to the **StudentsEntityDataSource** control. The markup will now resemble the following example:

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

## ADD NEW STUDENTS

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

- In *About.aspx*, add `EntityTypeFilter="Student"` to the `StudentStatisticsEntityDataSource` control and remove `Where="it.EnrollmentDate is not null"`. The markup will now resemble the following example:

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

- In *Instructors.aspx* and *InstructorsCourses.aspx*, add `EntityTypeFilter="Instructor"` to the `InstructorsEntityDataSource` control and remove `Where="it.HireDate is not null"`. The markup in *Instructors.aspx* now resembles the following example:

```
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="false"
    EntitySetName="People" EntityTypeFilter="Instructor"
    Include="OfficeAssignment"
    EnableUpdate="True">
</asp:EntityDataSource>
```

## INSTRUCTORS

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

## COURSE DETAILS

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

## COURSES TAUGHT

	ID	Title	Department
<a href="#">Select</a>	2042	Literature	English
<a href="#">Select</a>	3141	Trigonometry	Mathematics
<a href="#">Select</a>	4022	Microeconomics	Economics
<a href="#">Select</a>	4041	Macroeconomics	Economics
<a href="#">Select</a>	4061	Quantitative	Economics
<a href="#">Select</a>	4062	New engineering course	Engineering
<a href="#">Select</a>	4063	new course	Economics

## STUDENT GRADES

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

The markup in *InstructorsCourses.aspx* will now resemble the following example:

```
<asp:EntityDataSource ID="InstructorsEntityDataSource" runat="server"
    ContextTypeName="ContosoUniversity.DAL.SchoolEntities"
    EnableFlattening="False"
    EntitySetName="People" EntityTypeFilter="Instructor"
    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:  ▼

As a result of these changes, you've improved the Contoso University application's maintainability in several ways. You've moved selection and validation logic out of the UI layer (*.aspx* markup) and made it an integral part of the data access layer. This helps to isolate your application code from changes that you might make in the future to the database schema or the data model. For example, you could decide that students might be hired as teachers' aids and therefore would get a hire date. You could then add a new property to differentiate students from instructors and update the data model. No code in the web application would need to change except where you wanted to show a hire date for students. Another benefit of adding **Instructor** and **Student** entities is that your code is more readily understandable than when it referred to **Person** objects that were actually students or instructors.

You've now seen one way to implement an inheritance pattern in the Entity Framework. In the following tutorial, you'll learn how to use stored procedures in order to have more control over how the Entity Framework accesses the database.

## Part 7: Using Stored Procedures

In the previous tutorial you implemented a table-per-hierarchy inheritance pattern. This tutorial will show you how to use stored procedures to gain more control over database access.

The Entity Framework lets you specify that it should use stored procedures for database access. For any entity type, you can specify a stored procedure to use for creating, updating, or deleting entities of that type. Then in the data model you can add references to stored procedures that you can use to perform tasks such as retrieving sets of entities.

Using stored procedures is a common requirement for database access. In some cases a database administrator may require that all database access go through stored procedures for security reasons. In other cases you may want to build business logic into some of the processes that the Entity Framework uses when it updates the database. For example, whenever an entity is deleted you might want to copy it to an archive database. Or whenever a row is updated you might want to write a row to a logging table that records who made the change. You can perform these kinds of tasks in a stored procedure that's called whenever the Entity Framework deletes an entity or updates an entity.

As in the previous tutorial, you'll not create any new pages. Instead, you'll change the way the Entity Framework accesses the database for some of the pages you already created.

In this tutorial you'll create stored procedures in the database for inserting **Student** and **Instructor** entities. You'll add them to the data model, and you'll specify that the Entity Framework should use them for adding **Student** and **Instructor** entities to the database. You'll also create a stored procedure that you can use to retrieve **Course** entities.

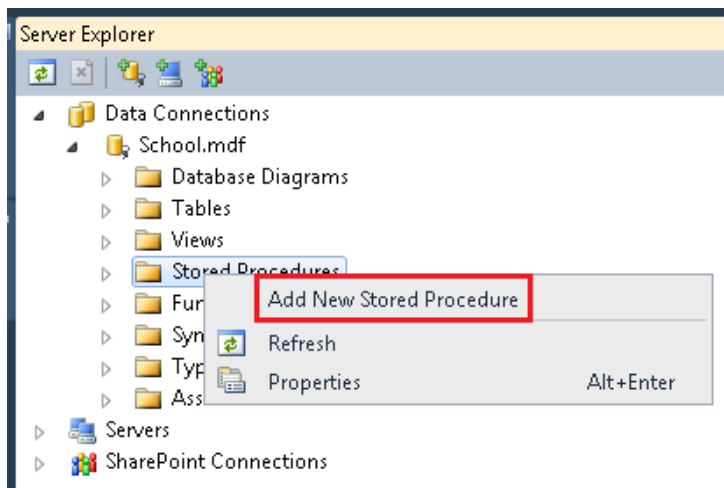
### Creating Stored Procedures in the Database

---

(If you're using the *School.mdf* file from the project available for download with this tutorial, you can skip this section because the stored procedures already exist.)

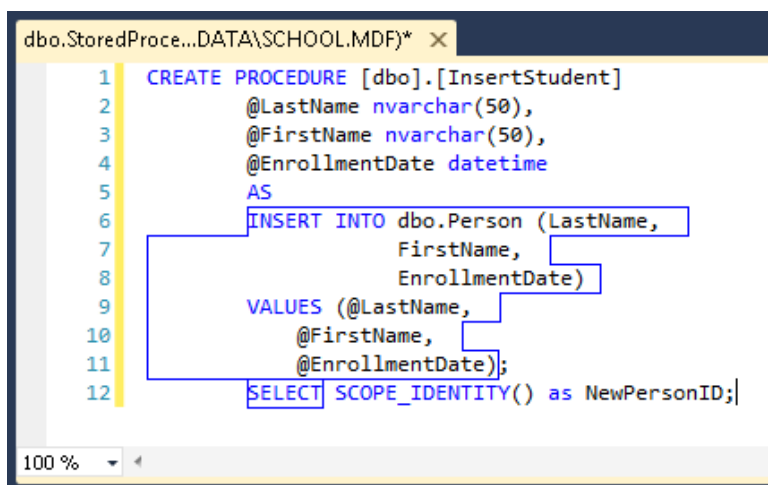
In **Server Explorer**, expand *School.mdf*, right-click **Stored Procedures**, and select **Add New Stored Procedure**.





Copy the following SQL statements and paste them into the stored procedure window, replacing the skeleton stored procedure.

```
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;
```



**Student** entities have four properties: **PersonID**, **LastName**, **FirstName**, and **EnrollmentDate**. The database generates the ID value automatically, and the stored procedure accepts parameters for the

other three. The stored procedure returns the value of the new row's record key so that the Entity Framework can keep track of that in the version of the entity it keeps in memory.

Save and close the stored procedure window.

Create an **InsertInstructor** stored procedure in the same manner, using the following SQL statements:

```
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;
```

Create **Update** stored procedures for **Student** and **Instructor** entities also. (The database already has a **DeletePerson** stored procedure which will work for both **Instructor** and **Student** entities.)

```
CREATE PROCEDURE [dbo].[UpdateStudent]
    @PersonID int,
    @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;
```

In this tutorial you'll map all three functions -- insert, update, and delete -- for each entity type. The Entity Framework version 4 allows you to map just one or two of these functions to stored procedures without mapping the others, with one exception: if you map the update function but not the delete function, the Entity Framework will throw an exception when you attempt to delete an entity. In the Entity Framework version 3.5, you did not have this much flexibility in mapping stored procedures: if you mapped one function you were required to map all three.

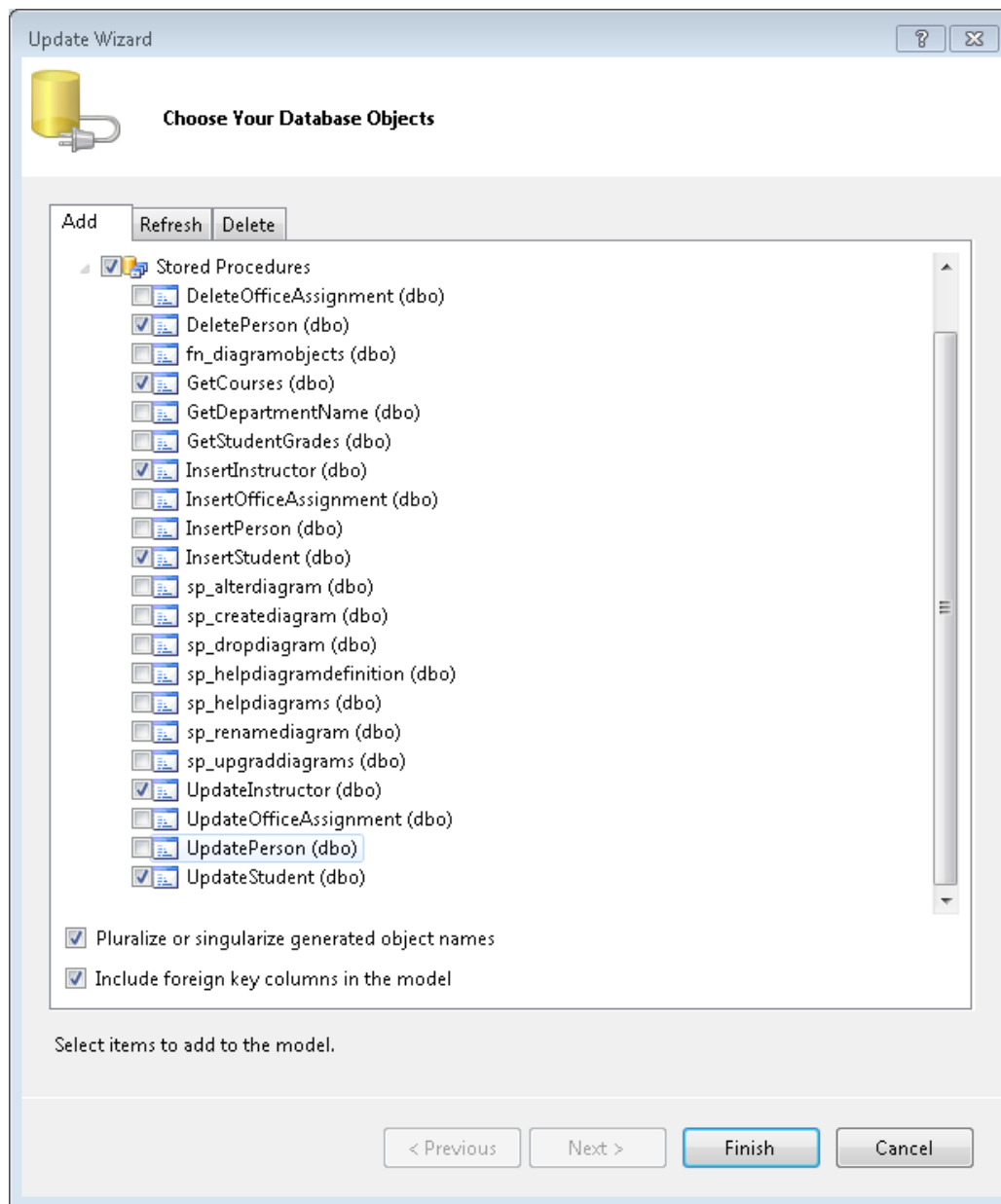
To create a stored procedure that reads rather than updates data, create one that selects all **Course** entities, using the following SQL statements:

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

## Adding the Stored Procedures to the Data Model

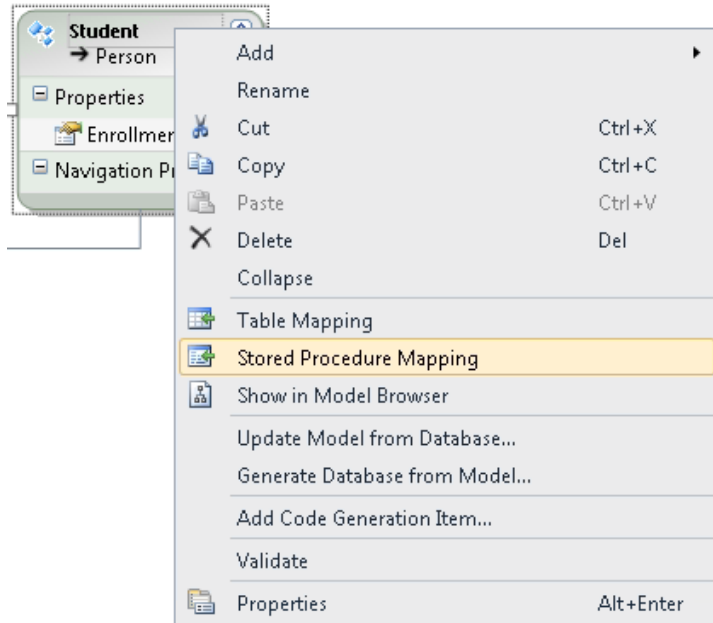
---

The stored procedures are now defined in the database, but they must be added to the data model to make them available to the Entity Framework. Open *SchoolModel.edmx*, right-click the design surface, and select **Update Model from Database**. In the **Add** tab of the **Choose Your Database Objects** dialog box, expand **Stored Procedures**, select the newly created stored procedures and the **DeletePerson** stored procedure, and then click **Finish**.

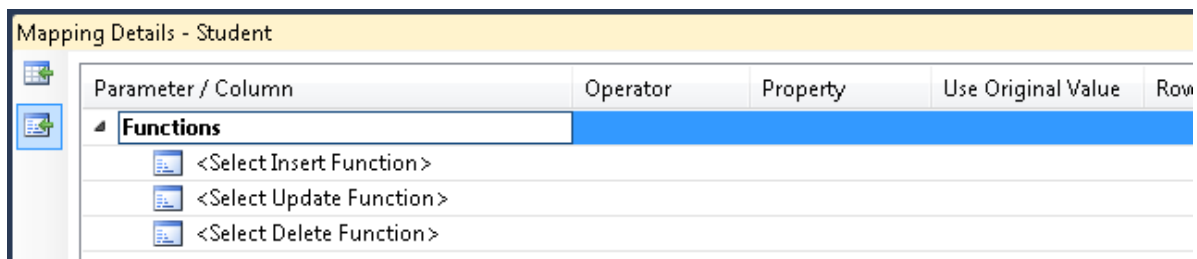


## Mapping the Stored Procedures

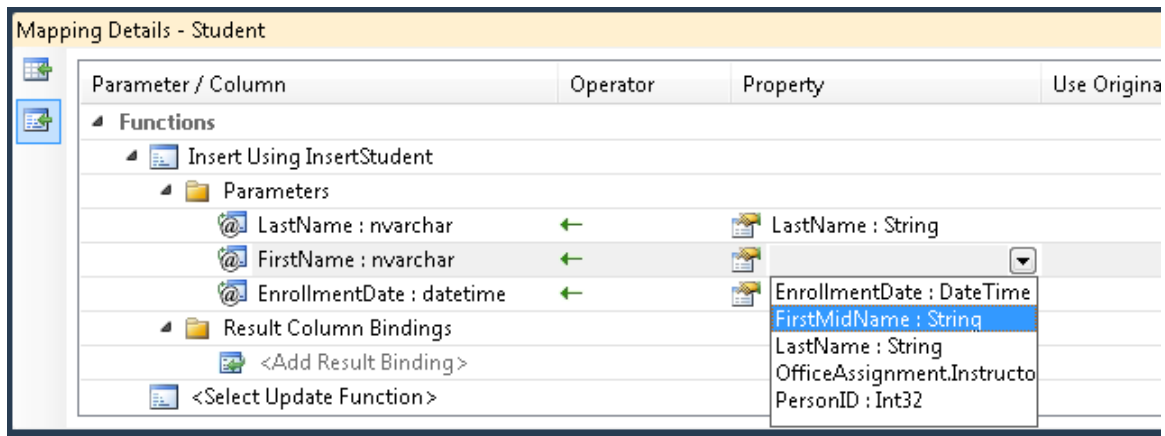
In the data model designer, right-click the **Student** entity and select **Stored Procedure Mapping**.



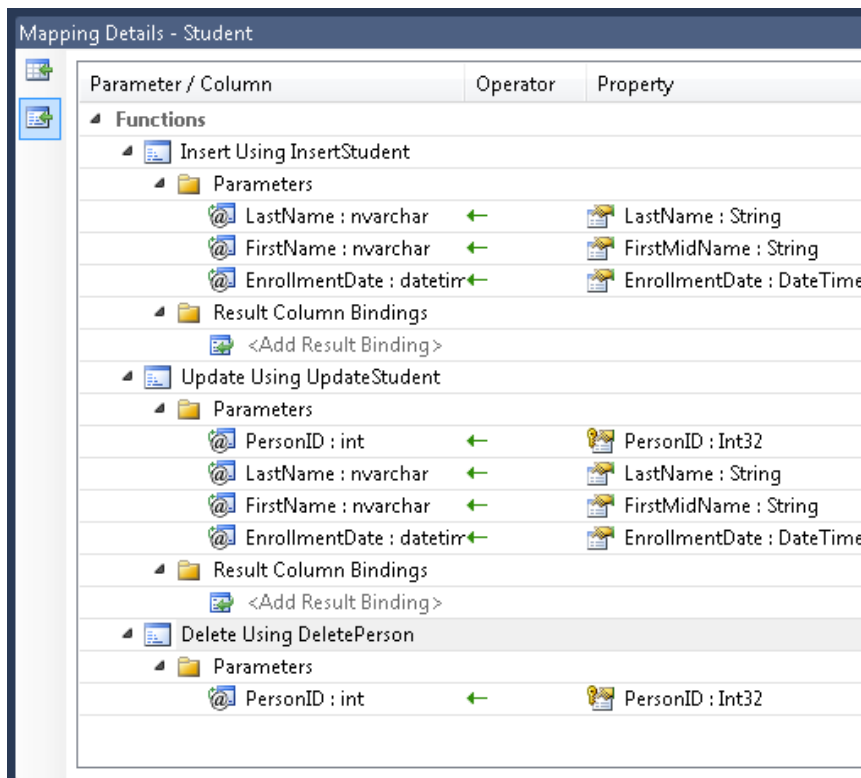
The **Mapping Details** window appears, in which you can specify stored procedures that the Entity Framework should use for inserting, updating, and deleting entities of this type.



Set the **Insert** function to **InsertStudent**. The window shows a list of stored procedure parameters, each of which must be mapped to an entity property. Two of these are mapped automatically because the names are the same. There's no entity property named **FirstName**, so you must manually select **FirstMidName** from a drop-down list that shows available entity properties. (This is because you changed the name of the **FirstName** property to **FirstMidName** in the first tutorial.)



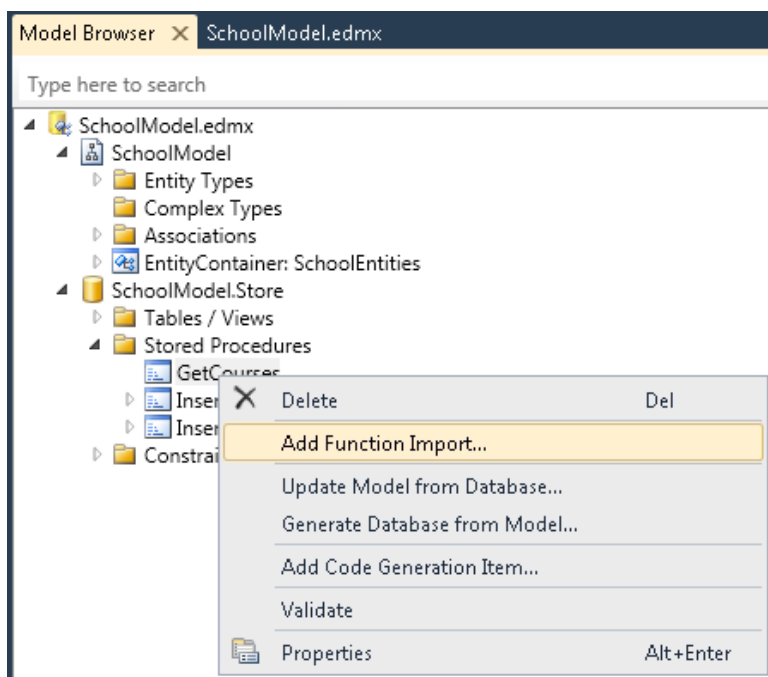
In the same **Mapping Details** window, map the **Update** function to the **UpdateStudent** stored procedure (make sure you specify **FirstMidName** as the parameter value for **FirstName**, as you did for the **Insert** stored procedure) and the **Delete** function to the **DeletePerson** stored procedure.



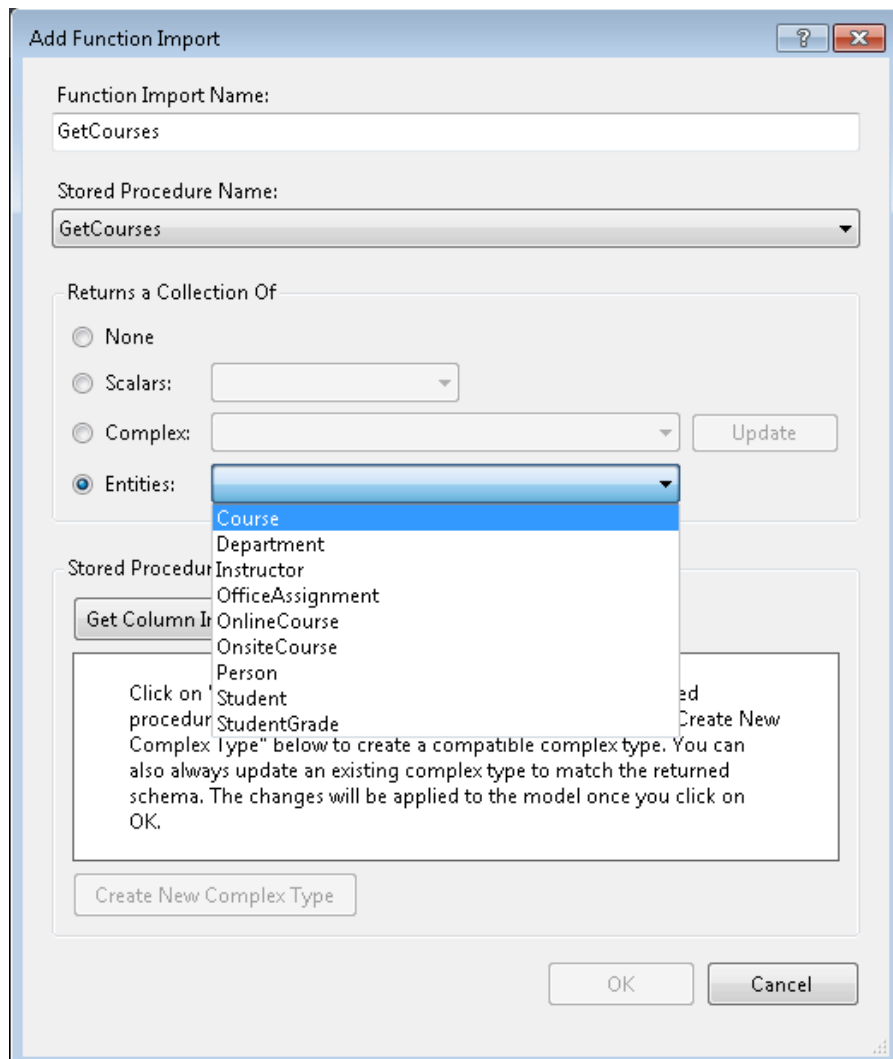
Follow the same procedure to map the insert, update, and delete stored procedures for instructors to the **Instructor** entity.

Mapping Details - Instructor		
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

For stored procedures that read rather than update data, you use the **Model Browser** window to map the stored procedure to the entity type it returns. In the data model designer, right-click the design surface and select **Model Browser**. Open the **SchoolModel.Store** node and then open the **Stored Procedures** node. Then right-click the **GetCourses** stored procedure and select **Add Function Import**.



In the **Add Function Import** dialog box, under **Returns a Collection Of** select **Entities**, and then select **Course** as the entity type returned. When you're done, click **OK**. Save and close the **.edmx** file.



## Using Insert, Update, and Delete Stored Procedures

Stored procedures to insert, update, and delete data are used by the Entity Framework automatically after you've added them to the data model and mapped them to the appropriate entities. You can now run the *StudentsAdd.aspx* page, and every time you create a new student, the Entity Framework will use the **InsertStudent** stored procedure to add the new row to the **Student** table.



## 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"/>
<a href="#">Insert</a> <a href="#">Cancel</a>	

Run the *Students.aspx* page and the new student appears in the list.

## STUDENT LIST

	ID	Name	Enrollment Date	Number of Courses
<a href="#">Edit</a> <a href="#">Delete</a>	26	Rogers, Cody	9/1/2002	2
<a href="#">Edit</a> <a href="#">Delete</a>	28	White, Anthony	9/1/2001	2
<a href="#">Edit</a> <a href="#">Delete</a>	29	Griffin, Rachel	9/1/2004	1
<a href="#">Edit</a> <a href="#">Delete</a>	30	Shan, Alicia	9/1/2003	2
<a href="#">Edit</a> <a href="#">Delete</a>	33	Gao, Erica	1/30/2003	0
<a href="#">Edit</a> <a href="#">Delete</a>	35	Smith, John	1/1/2011	0
<a href="#">Edit</a> <a href="#">Delete</a>	38	Student, New	9/10/2010	0
<a href="#">1</a> <a href="#">2</a> <a href="#">3</a>				

Change the name to verify that the update function works, and then delete the student to verify that the delete function works.

## STUDENT LIST

	ID	Name	Enrollment Date	Number of Courses
<a href="#">Edit</a> <a href="#">Delete</a>	26	Rogers, Cody	9/1/2002	2
<a href="#">Edit</a> <a href="#">Delete</a>	28	White, Anthony	9/1/2001	2
<a href="#">Edit</a> <a href="#">Delete</a>	29	Griffin, Rachel	9/1/2004	1
<a href="#">Edit</a> <a href="#">Delete</a>	30	Shan, Alicia	9/1/2003	2
<a href="#">Edit</a> <a href="#">Delete</a>	33	Gao, Erica	1/30/2003	0
<a href="#">Edit</a> <a href="#">Delete</a>	35	Smith, John	1/1/2011	0
<a href="#">Update</a> <a href="#">Cancel</a>	38	Student	<input type="text" value="ToBeDeleted"/>	0
<a href="#">1</a> <a href="#">2</a> <a href="#">3</a>				

## Using Select Stored Procedures

The Entity Framework does not automatically run stored procedures such as *GetCourses*, and you cannot use them with the *EntityDataSource* control. To use them, you call them from code.

Open the *InstructorsCourses.aspx.cs* file. The *PopulateDropDownLists* method uses a LINQ-to-Entities query to retrieve all course entities so that it can loop through the list and determine which ones an instructor is assigned to and which ones are unassigned:

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

Replace this with the following code:

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

The page now uses the **GetCourses** stored procedure to retrieve the list of all courses. Run the page to verify that it works as it did before.

(Navigation properties of entities retrieved by a stored procedure might not be automatically populated with the data related to those entities, depending on **ObjectContext** default settings. For more information, see [Loading Related Objects](#) in the MSDN Library.)

In the next tutorial, you'll learn how to use Dynamic Data functionality to make it easier to program and test data formatting and validation rules. Instead of specifying on each web page rules such as data format strings and whether or not a field is required, you can specify such rules in data model metadata and they're automatically applied on every page.

## Part 8: Using Dynamic Data Functionality to Format and Validate Data

In the previous tutorial you implemented stored procedures. This tutorial will show you how Dynamic Data functionality can provide the following benefits:

- Fields are automatically formatted for display based on their data type.
  - Fields are automatically validated based on their data type.
  - You can add metadata to the data model to customize formatting and validation behavior.
- When you do this, you can add the formatting and validation rules in just one place, and they're automatically applied everywhere you access the fields using Dynamic Data controls.

To see how this works, you'll change the controls you use to display and edit fields in the existing *Students.aspx* page, and you'll add formatting and validation metadata to the name and date fields of the **Student** entity type.

### STUDENT LIST

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

### FIND STUDENTS BY NAME

Enter any part of the name

Name	Enrollment Date
Barzdukas, Gytis	9/1/2005

## Using DynamicField and DynamicControl Controls

Open the *Students.aspx* page and in the **StudentsGridView** control replace the **Name** and **Enrollment Date** **TemplateField** elements with the following markup:

```
<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="Enrollment Date"
    SortExpression="EnrollmentDate" />

```

This markup uses **DynamicControl** controls in place of **TextBox** and **Label** controls in the student name template field, and it uses a **DynamicField** control for the enrollment date. No format strings are specified.

Add a **ValidationSummary** control after the **StudentsGridView** control.

```

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

```

In the **SearchGridView** control replace the markup for the **Name** and **Enrollment Date** columns as you did in the **StudentsGridView** control, except omit the **EditItemTemplate** element. The **Columns** element of the **SearchGridView** control now contains the following markup:

```

<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="Enrollment Date"
    SortExpression="EnrollmentDate" />

```

Open *Students.aspx.cs* and add the following **using** statement:

```
using ContosoUniversity.DAL;
```

Add a handler for the page's **Init** event:

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

This code specifies that Dynamic Data will provide formatting and validation in these data-bound controls for fields of the **Student** entity. If you get an error message like the following example when you run the page, it typically means you've forgotten to call the **EnableDynamicData** method in **Page\_Init**:

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.

Run the page.

#### STUDENT LIST

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

In the **Enrollment Date** column, the time is displayed along with the date because the property type is **DateTime**. You'll fix that later.

For now, notice that Dynamic Data automatically provides basic data validation. For example, click **Edit**, clear the date field, click **Update**, and you see that Dynamic Data automatically makes this a required field because the value is not nullable in the data model. The page displays an asterisk after the field and an error message in the **ValidationSummary** control:

## STUDENT LIST

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

- The EnrollmentDate field is required.

You could omit the **ValidationSummary** control, because you can also hold the mouse pointer over the asterisk to see the error message:

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

Dynamic Data will also validate that data entered in the **Enrollment Date** field is a valid date:

## STUDENT LIST

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

- The value is not valid.

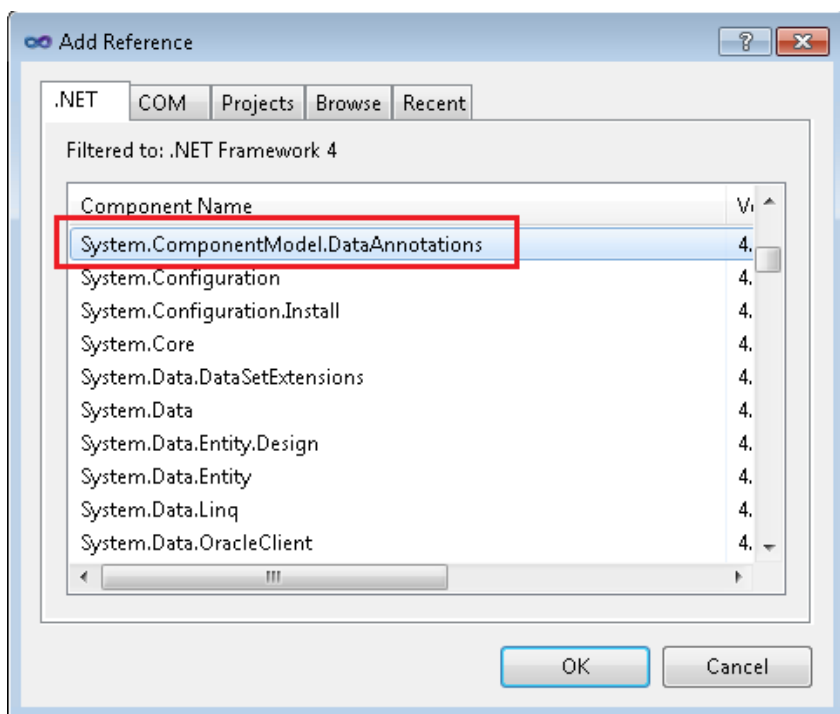
As you can see, this is a generic error message. In the next section you'll see how to customize messages as well as validation and formatting rules.

## Adding Metadata to the Data Model

---

Typically, you want to customize the functionality provided by Dynamic Data. For example, you might change how data is displayed and the content of error messages. You typically also customize data validation rules to provide more functionality than what Dynamic Data provides automatically based on data types. To do this, you create partial classes that correspond to entity types.

In **Solution Explorer**, right-click the **ContosoUniversity** project, select **Add Reference**, and add a reference to **System.ComponentModel.DataAnnotations**.



In the *DAL* folder, create a new class file, name it *Student.cs*, and replace the template code in it with the following code.

```
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; }
}
}

```

This code creates a partial class for the **Student** entity. The **MetadataType** attribute applied to this partial class identifies the class that you're using to specify metadata. The metadata class can have any name, but using the entity name plus "Metadata" is a common practice.

The attributes applied to properties in the metadata class specify formatting, validation, rules, and error messages. The attributes shown here will have the following results:

- **EnrollmentDate** will display as a date (without a time).
- Both name fields must be 25 characters or less in length, and a custom error message is provided.
- Both name fields are required, and a custom error message is provided.

Run the *Students.aspx* page again, and you see that the dates are now displayed without times:

## STUDENT LIST

	<u>Name</u>	<u>Enrollment Date</u>	<u>Number of Courses</u>
<a href="#">Edit</a> <a href="#">Delete</a>	Alexander, Carson	9/1/2055	3
<a href="#">Edit</a> <a href="#">Delete</a>	Alonso, Meredith	9/1/2002	1
<a href="#">Edit</a> <a href="#">Delete</a>	Anand, Arturo	9/1/2003	2

Edit a row and try to clear the values in the name fields. The asterisks indicating field errors appear as soon as you leave a field, before you click **Update**. When you click **Update**, the page displays the error message text you specified.



## STUDENT LIST

		Name
<a href="#">Update</a> <a href="#">Cancel</a>	<input type="text"/>	<input type="text"/>
<a href="#">Edit</a> <a href="#">Delete</a>	Alonso, Meredith	
<a href="#">Edit</a> <a href="#">Delete</a>	Anand, Arturo	
<a href="#">Edit</a> <a href="#">Delete</a>	Barzdukas, Gytis	
<a href="#">Edit</a> <a href="#">Delete</a>	Browning, Meredith	
<a href="#">Edit</a> <a href="#">Delete</a>	Bryant, Carson	
<a href="#">Edit</a> <a href="#">Delete</a>	Carlson, Robyn	
<a href="#">Edit</a> <a href="#">Delete</a>	Gao, Erica	
<a href="#">Edit</a> <a href="#">Delete</a>	Griffin, Rachel	
<a href="#">Edit</a> <a href="#">Delete</a>	Holt, Roger	

1 2 3

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

Try to enter names that are longer than 25 characters, click **Update**, and the page displays the error message text you specified.

## STUDENT LIST

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

Now that you've set up these formatting and validation rules in the data model metadata, the rules will automatically be applied on every page that displays or allows changes to these fields, so long as you use **DynamicControl** or **DynamicField** controls. This reduces the amount of redundant code you have

to write, which makes programming and testing easier, and it ensures that data formatting and validation are consistent throughout an application.

This concludes this series of tutorials on Getting Started with the Entity Framework. For more resources to help you learn how to use the Entity Framework, visit the following sites:

- [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](#)

# Disclaimer

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. This document is confidential and proprietary to Microsoft. It is disclosed and can be used only pursuant to a non-disclosure agreement.

© 2011 Microsoft. All Rights Reserved.

Microsoft is a trademark of the Microsoft group of companies. All other trademarks are property of their respective owners.