

Н.А. Осипов

**Разработка приложений ASP.NET
с применением Entity Framework**

Учебное пособие



ASP.NET

**Entity
Framework**

Санкт-Петербург

2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
УНИВЕРСИТЕТ ИТМО

Н.А. Осипов

**Разработка приложений ASP.NET
с применением Entity Framework**

Учебное пособие



Санкт-Петербург

2016

Осипов Н.А., Разработка приложений ASP.NET с применением Entity Framework. – СПб: Университет ИТМО, 2016. – 80 с.

В пособии описывается процесс создания Web-приложения на основе ASP.NET Web Forms с использованием технологии Entity Framework в среде разработки Visual Studio. В результате последовательного выполнения лабораторных работ будет построен вебсайт вымышленного университета.

Предназначено для подготовки бакалавров по направлению «11.03.02 Инфокоммуникационные технологии и системы связи» по бакалаврским программам: «Инфокоммуникационные системы» и «Облачные технологии».

Рекомендовано к печати Ученым советом факультета Инфокоммуникационных технологий, протокол № 09/16 от 24.11.2016г.



Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2016

© Н.А. Осипов, 2016

Оглавление

Введение	5
Лабораторная работа 1. Разработка базовой основы Web приложения	5
Упражнение 1. Создание Web приложения	5
Упражнение 2. Создание базы данных	8
Упражнение 3. Работа с моделью данных.....	13
Лабораторная работа 2. Применение EntityDataSource	18
Упражнение 1. Добавление и настройка элемента EntityDataSource	18
Упражнение 2. Настройка правила: разрешить удаление	21
Упражнение 3. Применение GridView для чтения и обновления данных.....	24
Упражнение 4. Настройка EntityDataSource для улучшения производительности	28
Упражнение 5. Отображение данных с помощью свойства навигации (Navigation Property)	28
Упражнение 6. Применение DetailsView для вставки данных.....	29
Упражнение 7. Отображение данных в списке Drop-Down .	30
Лабораторная работа 3. Фильтрация, упорядочивание и группирование данных	32
Упражнение 1. Настройка свойств элемента EntityDataSource для более наглядного отображения данных	33
Упражнение 2. Реализация поиска данных	37
Лабораторная работа 4. Работа со связанными данными	40
Упражнение 1. Отображение и обновление связанных данных в элементе GridView	40
Упражнение 2. Отображение связанных данных в отдельном элементе управления	44
Упражнение 3. Применение события "Selected" EntityDataSource для отображения связанных данных.....	47
Лабораторная работа 5. Работа со связанными данными (продолжение)	49
Упражнение 1. Добавление сущности с отношением к другой сущности.....	50
Упражнение 2. Применение отношения многое ко многим.	51

Лабораторная работа 6. Реализация наследования Table-per-Hierarchy	56
Упражнение 1. Добавление производных сущностей Instructor и Student	58
Упражнение 2. Применение сущностей Instructor и Student	62
Лабораторная работа 7. Использование хранимых процедур	65
Упражнение 1. Создание хранимых процедур в базе данных.....	65
Упражнение 2. Размещение хранимых процедур в модели данных.....	67
Упражнение 3. Применение хранимых процедур	72
Лабораторная работа 8. Применение функциональности Dynamic Data для форматирования и валидации данных.....	73
Упражнение 1. Применение элементов DynamicField и DynamicControl.....	73
Упражнение 2. Добавление правил валидации и форматирования.....	75
Список литературы	77

Введение

В данном пособии рассматривается процесс создания Web-приложения на основе ASP.NET Web Forms с использованием технологии Entity Framework в среде разработки Visual Studio. В результате последовательного выполнения лабораторных работ будет построен вебсайт вымышленного университета Contoso. Пользователи сайта смогут просматривать список студентов, обновлять данные о студентах, курсах и вводить другую служебную информацию.

В пособии применяется подход к реализации данных в Entity Framework, называемый *Database First*. В этом случае на основе готовой базы Entity Framework создает в Visual Studio модель данных, включающую в себя классы и свойства, которые соответствуют объектам базы данных – таблицам и столбцам. Модель базы данных хранится в XML файле с расширением .edmx. Дизайнер Entity Framework предоставляет графический интерфейс для отображения и редактирования модели.

Лабораторная работа 1. Разработка базовой основы Web приложения

В этой лабораторной работе вы создадите основу web приложения, добавите базу данных и создадите модель данных.

Упражнение 1. Создание Web приложения

1. Откройте Visual Studio и в окне создания проекта (см. рис. 1.1.1), создайте новый проект приложения ASP.NET Web на основе шаблона **ASP.NET Web Application** (расположение проекта выберите на свое усмотрение):

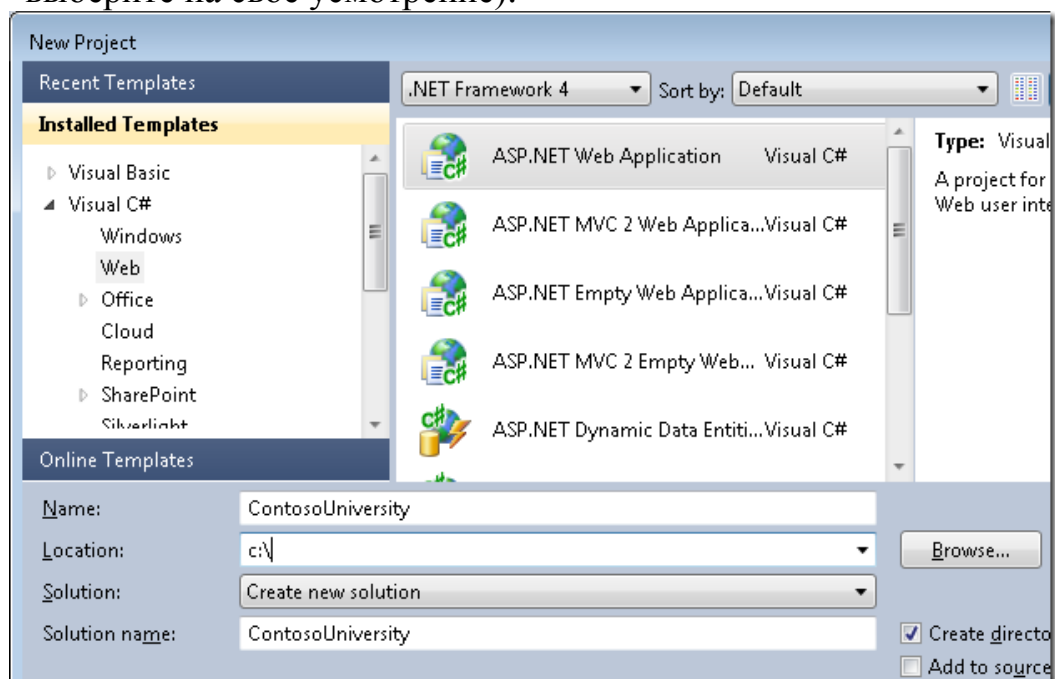


Рисунок 1.1.1 Выбор шаблона ASP.NET приложения

2. Изучите структуру созданного на основе этого шаблона проекта web-приложения, который включает основные страницы, стили CSS и главную (master) страницу (см. рисунок 1.1.2):

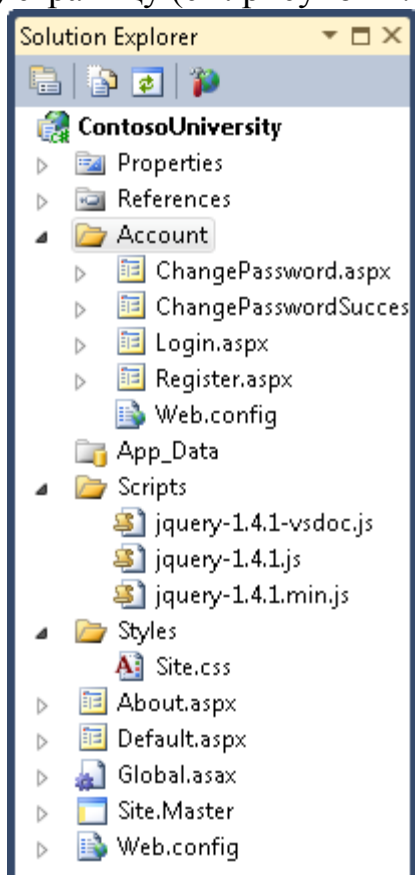


Рисунок 1.1.2 Структура проекта ASP.NET приложения

3. Откройте файл *Site.Master* и измените "My ASP.NET Application" на "Contoso University".

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

4. Найдите элемент *Menu* с именем *NavigationMenu* и добавьте элементы меню, для открытия страниц, которые будут добавлены в проект далее:

```
<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
EnableViewState="false" IncludeStyleBlock="false"
Orientation="Horizontal">
    <asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
EnableViewState="false"
IncludeStyleBlock="false" Orientation="Horizontal">
    <Items>
        <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home" />
        <asp:MenuItem NavigateUrl="~/About.aspx" Text="About" />
        <asp:MenuItem NavigateUrl="~/Students.aspx"
Text="Students">
```

```

        <asp:MenuItem NavigateUrl="~/StudentsAdd.aspx"
Text="Add Students" />
    </asp:MenuItem>
    <asp:MenuItem NavigateUrl="~/Courses.aspx"
Text="Courses">
        <asp:MenuItem NavigateUrl="~/CoursesAdd.aspx"
Text="Add Courses" />
    </asp:MenuItem>
    <asp:MenuItem NavigateUrl="~/Instructors.aspx"
Text="Instructors">
        <asp:MenuItem
NavigateUrl="~/InstructorsCourses.aspx" Text="Course Assignments" />
        <asp:MenuItem
NavigateUrl="~/OfficeAssignments.aspx" Text="Office Assignments" />
    </asp:MenuItem>
    <asp:MenuItem NavigateUrl="~/Departments.aspx"
Text="Departments">
        <asp:MenuItem NavigateUrl="~/DepartmentsAdd.aspx"
Text="Add Departments" />
    </asp:MenuItem>
</Items>
</asp:Menu>

```

- Откройте страницу *Default.aspx* и измените содержимое заголовка элемента Content с именем BodyContent, а остальную разметку удалите:

```

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

```

- Постойте и запустите приложение. Загрузится сайт и отобразится стартовая страница, как на рисунке 1.1.3. Изучите созданное меню.

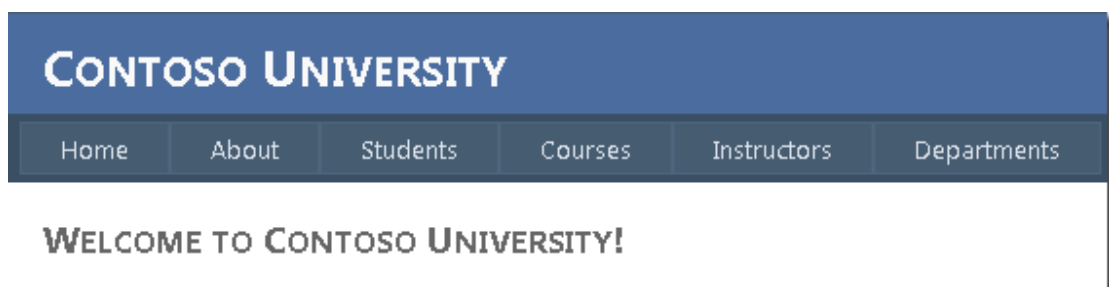


Рисунок 1.1.3 Стартовая страница сайта

Упражнение 2. Создание базы данных

В этом упражнении вы используете дизайнер Entity Framework для автоматического создания модели данных на основе уже существующей базы данных *School*. Эта база данных и файл сценария ее создания (если возникнет необходимость в этом) находится в папке *ASP.NET Web Forms Application Using Entity Framework 4.0 Database First*.

Добавление в проект базы данных

1. Для добавления в проект базы данных выберите в контекстном меню папки *App_Data* команду **Add → Existing Item**, затем найдите файл базы данных *School.mdf* и добавьте его в папку проекта.
2. Постройте приложение.
3. Создайте схему базы данных, для этого в обозревателе сервера (**Server Explorer**) раскройте последовательно папки **Подключение данных (Data Connections)** и *School.mdf*, затем в контекстном меню папки **Схема базы данных (Database Diagrams)** выберите команду **Добавить новую диаграмму (Add New Diagram)**, как на рисунке 1.2.1:

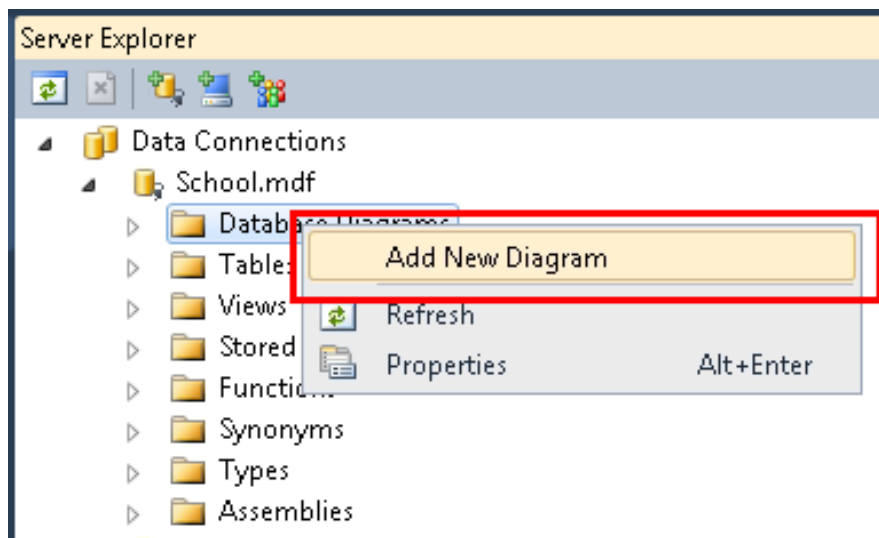


Рисунок 1.2.1 Добавление диаграммы

4. В окне **Добавить таблицу (Add Table)** выберите все таблицы и нажмите **Добавить (Add)**.
5. Изучите созданную диаграмму, которая показывает таблицы, столбцы и отношения между таблицами (см. рисунок 1.2.2).

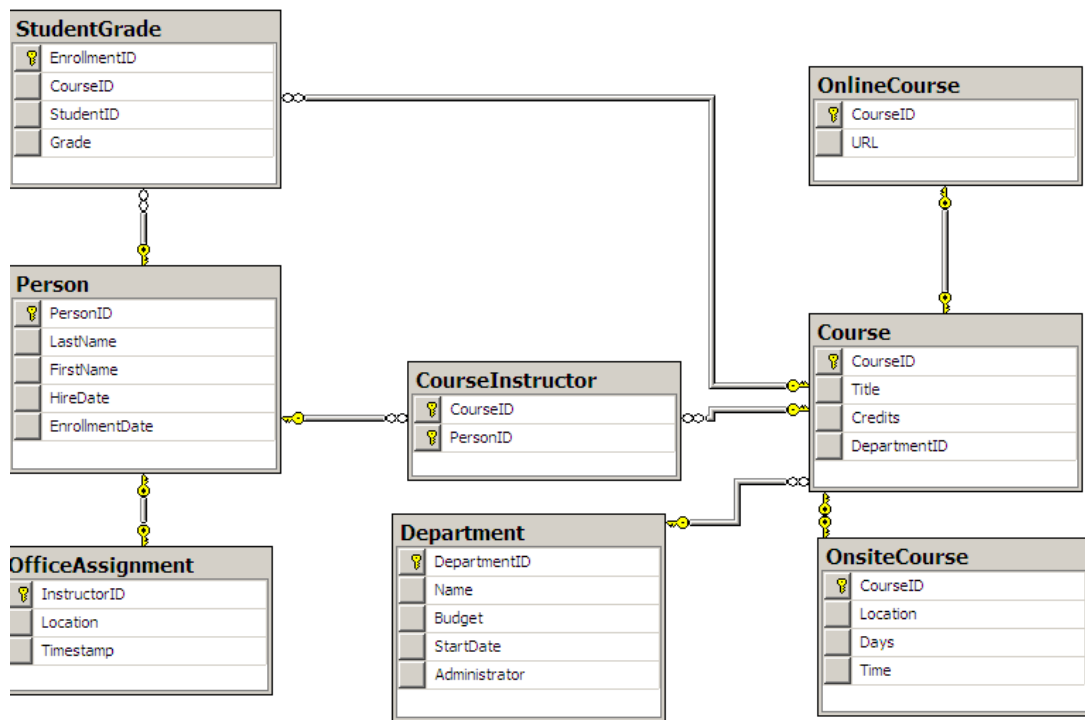


Рисунок 1.2.2 Схема базы данных School

6. Сохраните диаграмму как "SchoolDiagram" и закройте ее. Проверьте, что в папке диаграмм (см. рисунок 1.2.3) сохранилась созданная диаграмма.

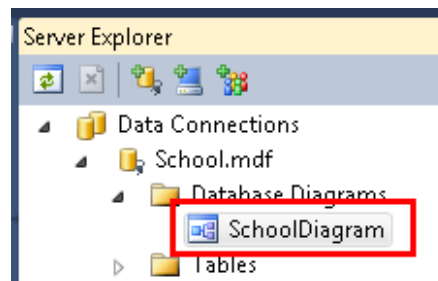


Рисунок 1.2.3 Отображение созданной диаграммы

Создание модели данных Entity Framework

1. Создайте в корне проекта папку для хранения модели, назовите ее *DAL* (будем понимать ее как *Data Access Layer*).
2. Добавьте в папку *DAL* требуемую модель, для этого выполните следующие действия:
 - а. В контекстном меню папки выберите **Добавить (Add)**, далее **Создать элемент (New Item)**,
 - б. В окне добавления нового элемента (см. рисунок 1.2.4) в категории шаблонов выберите **Данные (Data)**, затем **модель ADO.NET EDM (ADO.NET Entity Data Model)**, имя укажите *SchoolModel.edmx* и нажмите **Добавить (Add)**.

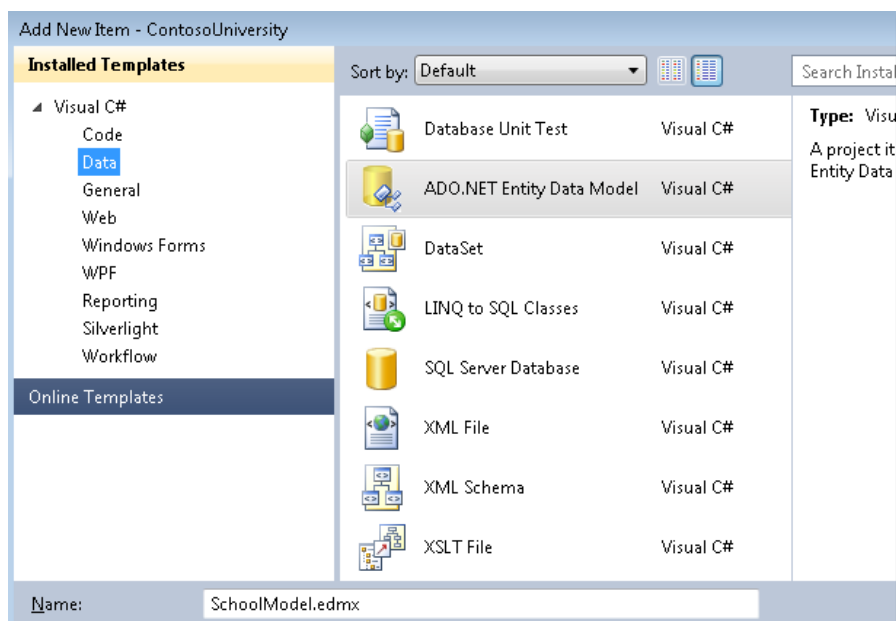


Рисунок 1.2.4 Добавление модели данных

Запустится мастер построения моделей **Entity Data Model Wizard**.

3. На первом шаге мастера (см. рисунок 1.2.5) оставьте опцию **Создать из базы данных (Generate from database)**, отмеченную по умолчанию. Нажмите **Далее (Next)**.

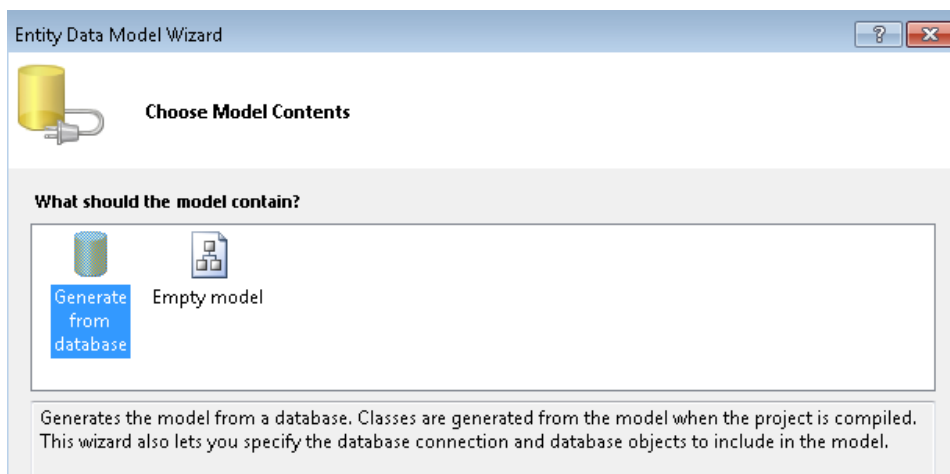


Рисунок 1.2.5 Выбор содержимого модели

4. На шаге **Выбор источника подключения (Choose Your Data Connection)** в списке соединения укажите базу данных School (см. рисунок 1.2.6) и сохраните подключение в файле *Web.config* как **SchoolEntities**, нажмите **Далее (Next)**.

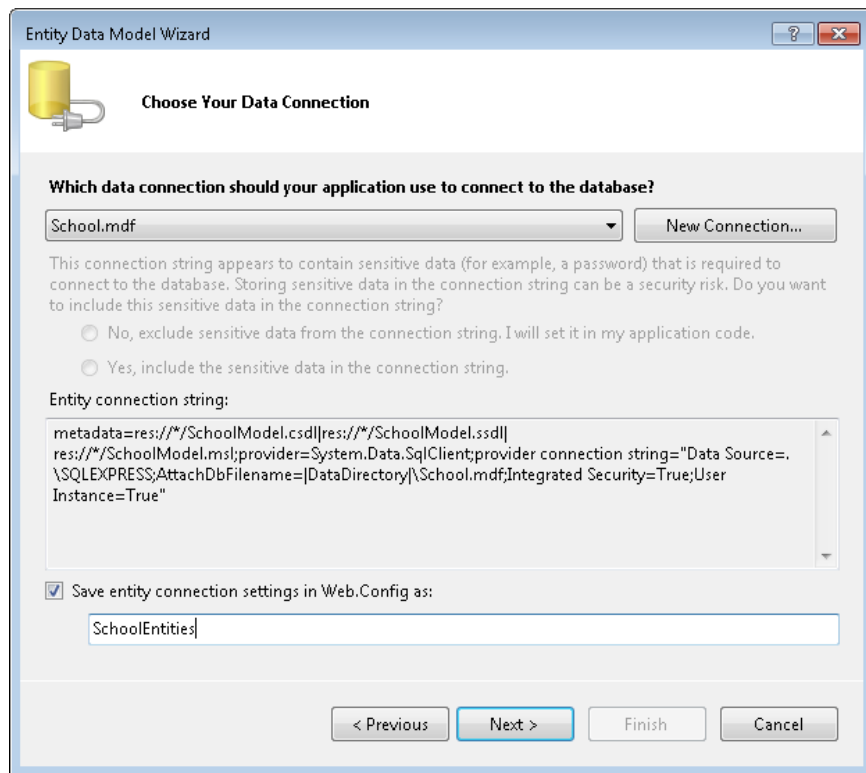


Рисунок 1.2.6 Выбор источника подключения

5. На шаге **Выбор объектов базы данных (Choose Your Database Objects)** выберите все таблицы, кроме sysdiagrams (она была созданная ранее); установите флажки формирования имен и включения столбцов внешних ключей (см. рисунок 1.2.7), нажмите **Готово (Finish)**.

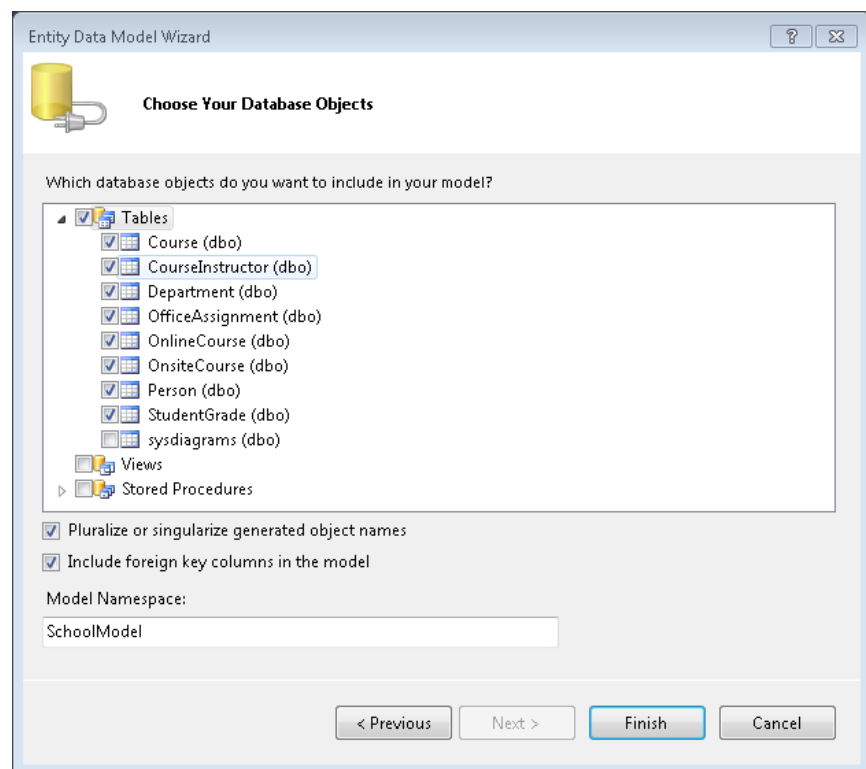


Рисунок 1.2.7 Выбор таблиц базы данных

6. Изучите графическое представление объектов (сущностей) Entity Framework, которые соответствуют таблицам базы данных, изображенное на рисунке 1.2.8:

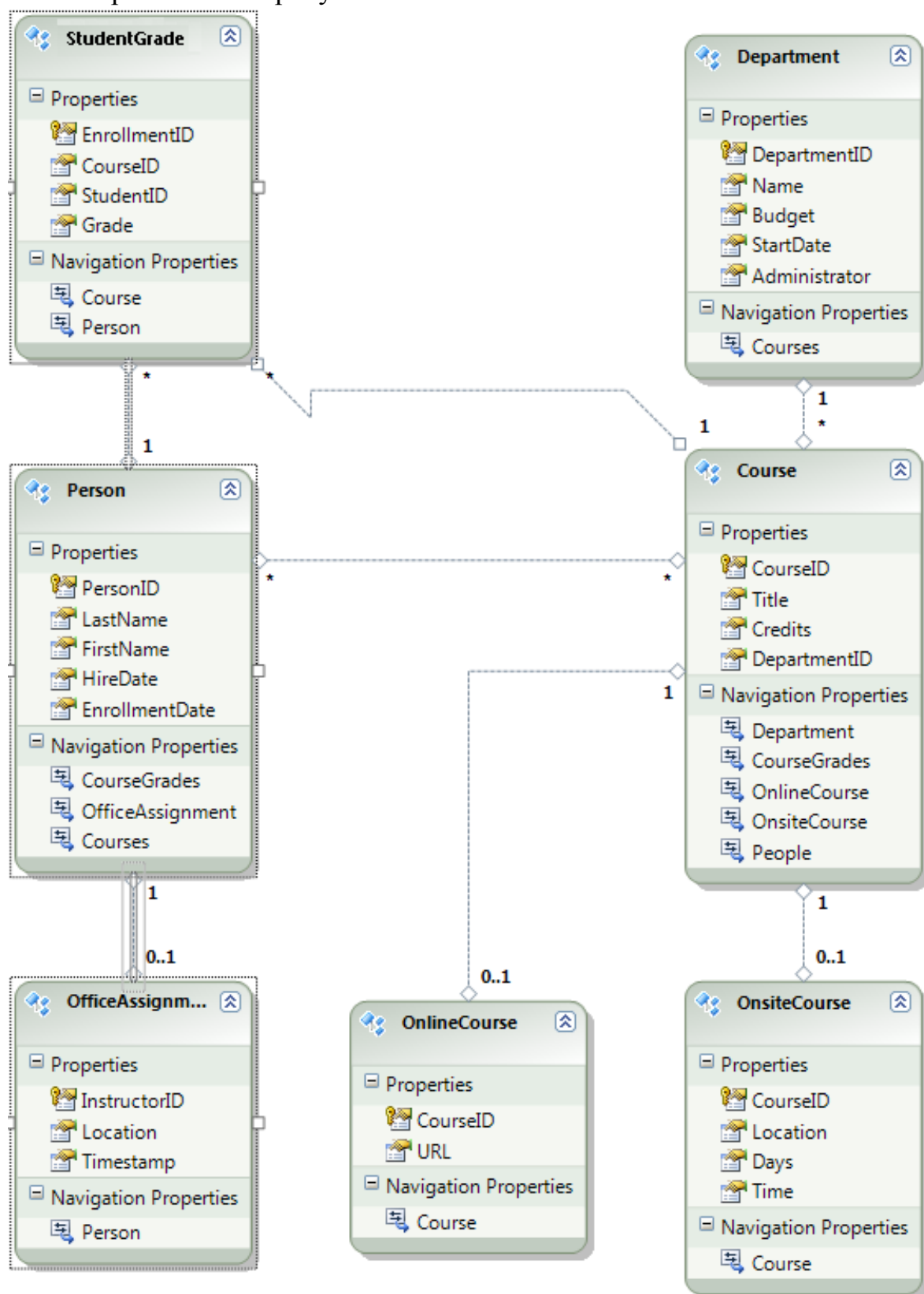


Рисунок 1.2.8 Графическое представление объектов (сущностей) Entity Framework

Упражнение 3. Работа с моделью данных

В этом упражнении вы подробно изучите построенную модель, внесете в нее изменения и познакомитесь с различными способами просмотра модели.

Изучение модели данных Entity Framework

Обратите внимание, что диаграмма сущностей очень похожа на схему базы данных, первое отличие состоит в добавлении символов в конце каждой ассоциации, которые указывают тип ассоциации.

1. Изучите следующие типы ассоциации:

- *one-to-zero-or-one* ассоциация представляется как "1" и "0..1" (см. рисунок 1.3.1):

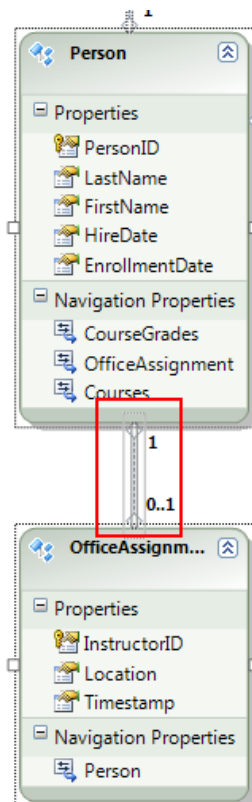


Рисунок 1.3.1 Ассоциация one-to-zero-or-one

В этом случае сущность `Person` может, как ассоциироваться с сущностью `OfficeAssignment`, так и не ассоциироваться. А вот сущность `OfficeAssignment` должна иметь отношение с сущностью `Person`. Другими словами, инструктор может быть назначен на должность, но это необязательно, а вот в офис может быть назначен только один инструктор.

- *one-to-many* ассоциация представлена на рисунке 1.3.2 как "1" и "*":

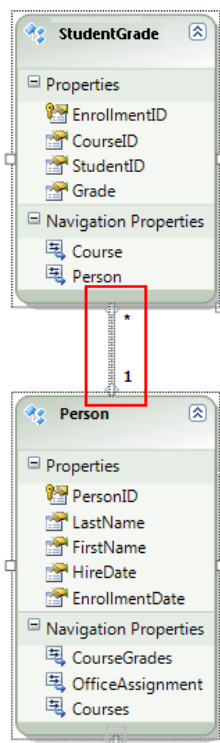


Рисунок 1.3.2 Ассоциация one-to-many

В этом случае сущность `Person` может иметь ассоциативную связь с сущностью `StudentGrade`, а в свою очередь, сущность `StudentGrade` должна быть связана только с одним экземпляром сущности `Person`. Сущность `StudentGrade` представляет обучающие курсы, и если студент поступил на обучение, но для него нет обучающего курса, то свойство `Grade` примет значение `null`. Каждый конкретный обучающий курс относится только к одному студенту.

Другими словами, студент может не быть зарегистрирован вообще ни в каком курсе, может быть обучающимся по одному курсу или может быть обучаться по нескольким курсам.

- *many-to-many* ассоциация представляется как "*" и "*" (см. рисунок 1.3.3).

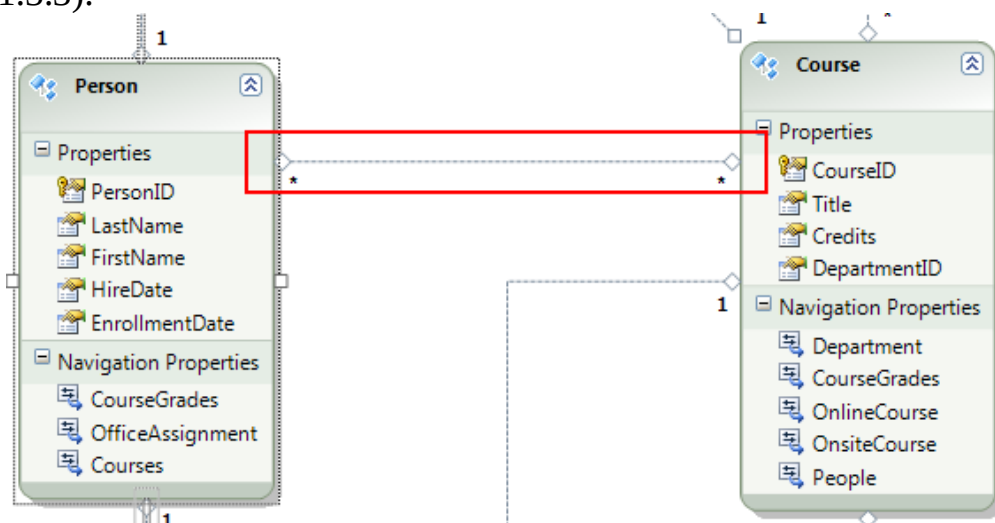


Рисунок 1.3.3 Ассоциация many-to-many

В этом случае Person может быть связан с сущностью Course, и обратное также верно: сущность Course может быть связана с сущностью Person.

Другими словами, преподаватель может вести несколько курсов, и курс может преподаваться несколькими преподавателями. В этой базе данных, эти отношения распространяются только на преподавателей и не связывают студентов с курсами. Студенты связаны с курсами таблицей StudentGrades.

2. Обратите внимание на второе отличие между схемой базы данных и моделью данных: дополнительный раздел *Свойства навигации* (**Navigation Properties**) для каждого объекта.

Свойство навигации содержит ссылки на связанные сущности. Например, свойство Courses в сущности Person содержит коллекцию всех курсов, которые связаны с этим человеком (см. рисунок 1.3.4).

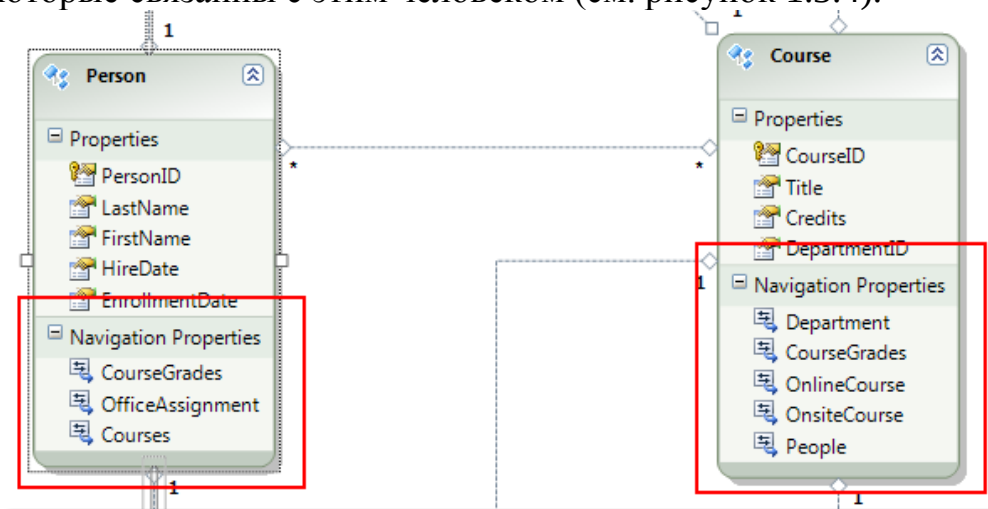


Рисунок 1.3.4 Использование свойств навигации

Еще одно отличие между базой данных и моделью данных является отсутствие таблицы ассоциации CourseInstructor, которая используется в базе данных, чтобы связать таблицы Person и Course между которыми отношение многие-ко-многим.

Навигационные свойства позволяют получить связанные объекты Course и Person, так что нет никакой необходимости представлять таблицу ассоциации в модели данных (см. рисунок 1.3.5).

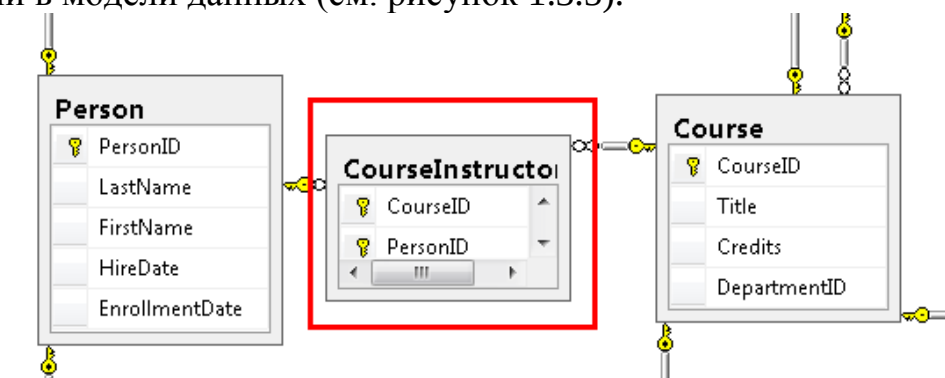


Рисунок 1.3.5 Использование таблицы ассоциации

Изменение модели данных

Созданная модель данных может быть изменена. Изменения эти, как правило, не носят существенный характер и служат для уточнения модели.

Например, предположим, что столбец `FirstName` таблицы `Person` на самом деле содержит и имя человека, и его отчество. В силу различных причин изменить саму базу данных нет возможности. Вы можете изменить имя свойства `FirstName` в модели данных, оставляя его эквивалент без изменений в базе данных.

1. В дизайнере модели откройте контекстное меню свойства **FirstName** в сущности `Person`, и выберите **Переименовать (Rename)**, как показано на рисунке 1.3.6:

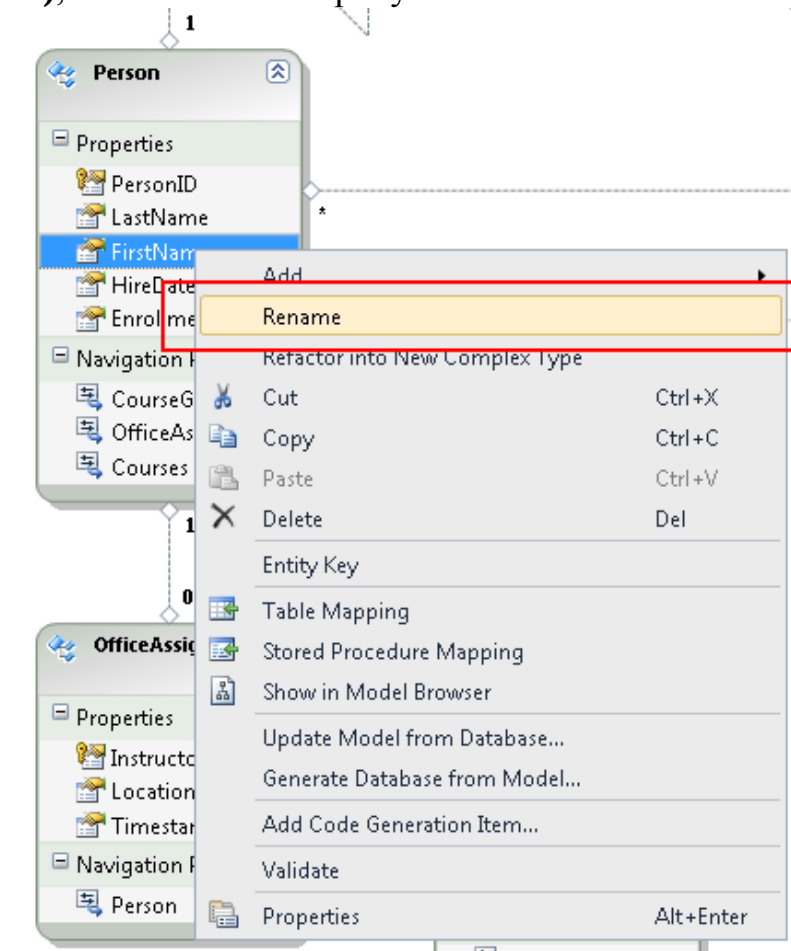


Рисунок 1.3.6 Переименование свойства сущности

2. Введите новое имя "FirstMidName".

Теперь в коде можно будет обращаться по этому имени.

Просмотр моделей

Среда разработки предоставляет удобный способ для просмотра структуры базы данных, структуры модели данных и отображение между ними.

1. Щелкните правой кнопкой мыши пустую область дизайнера, а затем нажмите **Обозреватель моделей (Model Browser)**.

На панели обозревателе моделей (**Model Browser**) отобразится дерево моделей. Узел **SchoolModel** содержит структуру модели данных, а узел **SchoolModel.Store** – структуру базы данных, как показано на рисунке 3.7:

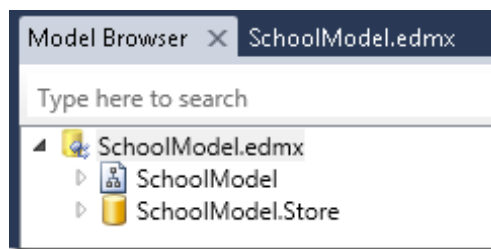


Рисунок 1.3.7 Дерево моделей

2. Раскройте узел **SchoolModel.Store**, далее раскройте **Таблицы/Представления (Tables / Views)** и просмотрите список таблиц, раскройте таблицу **Course** и изучите перечень столбцов.
3. Раскройте узел **SchoolModel**, далее раскройте **Типы Сущностей (Entity Types)**, затем раскройте узел **Course** и просмотрите его содержимое.
4. Проверьте внесенные Вами изменения в модели данных (свойство **FirstMidName**). Это можно сделать, просто просмотрев соответствующие узлы обозревателя, но удобнее для этой цели использовать таблицу сопоставления.
5. В обозревателе моделей в контекстном меню сущности **Person** выберите команду **Таблицы сопоставления (Table Mapping)** (см. рисунок 1.3.8):

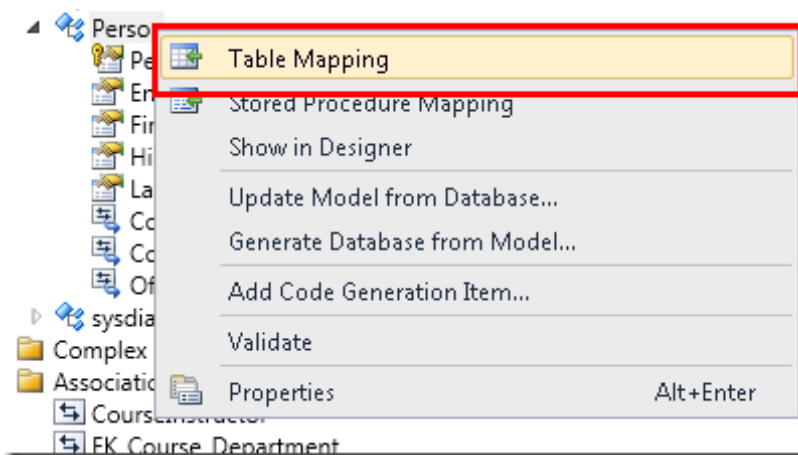


Рисунок 1.3.8 Открытие таблицы сопоставления

6. В окне панели **Сведения о сопоставлении (Mapping Details)** (см. рисунок 1.3.9) найдите в базе данных столбец **FirstName** и проверьте, что он сопоставляется со свойством **FirstMidName**, которое было переименовано в модели данных.

Column	Operator	Value / Property
Tables		
Maps to Person		
<Add a Condition>		
Column Mappings		
PersonID : int	↔	PersonID : Int32
LastName : nvarchar	↔	LastName : String
FirstName : nvarchar	↔	FirstMidName : String
HireDate : datetime	↔	HireDate : DateTime
EnrollmentDate : datetime	↔	EnrollmentDate : DateTime
<Add a Table or View>		

Рисунок 1.3.9 Таблица сопоставления

Платформа Entity Framework использует XML для хранения информации о базе данных, модели данных, и отображений между ними. Файл SchoolModel.edmx представляет собой XML-файл, который содержит эту информацию.

Дизайнер представляет модель данных в графическом формате, но вы можете также просмотреть файл как XML.

- В контекстном меню файла EDMX (в обозревателе решений) нажмите **Открыть с помощью (Open With)** и выберите **редактор (текстовый) XML (XML (Text) Editor)**.

Следует учитывать при просмотре модели, что дизайнер и редактор XML это только два различных способа открытия модели и нельзя работать с одним и тем же файлом в двух режимах открытия.

- Постройте проект. Возможные изменения в модели данных не доступны для дизайнера, пока проект не будет построен.

В результате выполненных упражнений вы создали веб-сайт, базу данных, и модель данных.

В следующей работе вы будете оперировать с данными, используя модель данных и элемент управления ASP.NET `EntityDataSource`.

Лабораторная работа 2. Применение `EntityDataSource`

В этой лабораторной работе вы познакомитесь с элементом управления `EntityDataSource`, который обеспечивает удобную работу с моделью данных Entity Framework. Вы создадите элементы `GridView` для отображения и редактирования данных, `DetailsView` для добавления новых студентов и `DropDownList` для выбора кафедр, который вы будете использовать в дальнейшем для отображения связанных с ними обучающих курсов.

Упражнение 1. Добавление и настройка элемента `EntityDataSource`

В этом упражнении вы настроите элемент `EntityDataSource` для чтения объектов `Person` из множества `People`.

1. Добавьте в проект новую web-страницу с помощью шаблона **Веб-форма, использующая главную страницу (Web Form using Master Page)**, и присвойте ей имя *Students.aspx* (см. рисунок 2.1.1.)

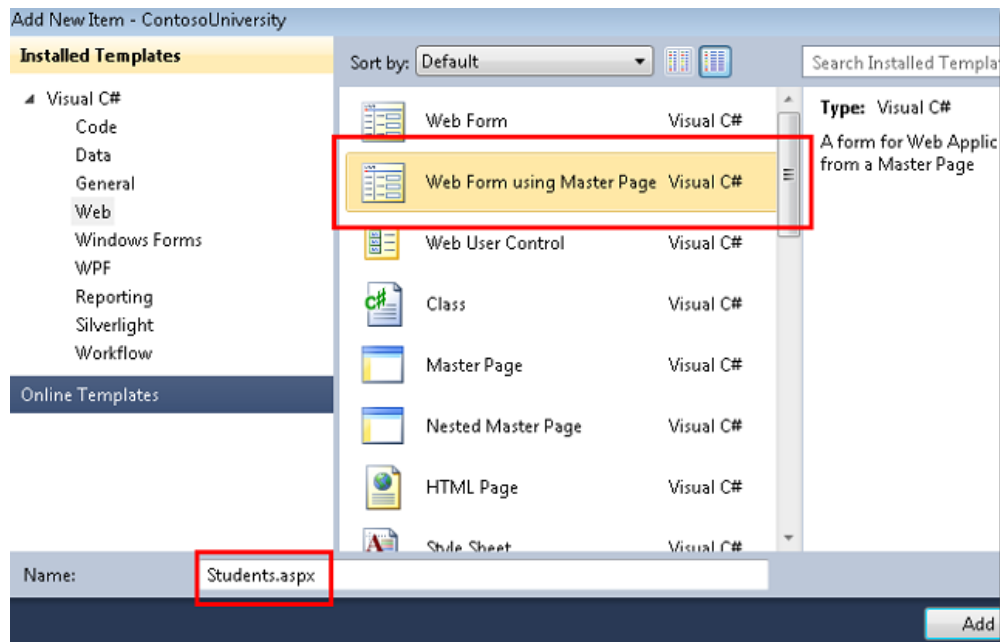


Рисунок 2.1.1 Создание нового проекта

2. В окне выбора мастер-страницы (рисунок 2.1.2) укажите *Site.Master* как мастер страницы. Все страницы вы будете создавать на основе этой мастер-страницы.

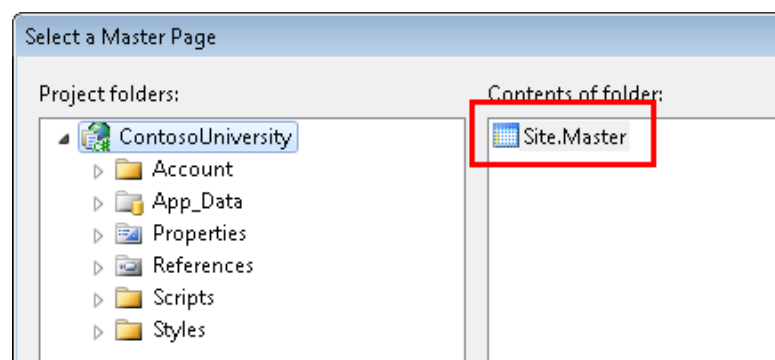


Рисунок 2.1.2 Выбор мастер-страницы

3. В коде разметки новой страницы добавьте текст **Student List** стиля заголовка **h2** в элемент содержимого **Content** с именем **Content2**:

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">
    <h2>
        Student List</h2>
</asp:Content>
```
4. Откройте страницу *Students.aspx* в обозревателе и убедитесь, что добавленная строка отображается как надо.
5. Откройте панель инструментов (**Toolbox**) и из вкладки **Данные (Data)** перенесите элемент **EntityDataSource** на страницу после

тега `</h2>` и измените значение свойства ID на `StudentsEntityDataSource`:

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

- Откройте страницу *Students.aspx* в режиме дизайнера (**Design**), кликните смарт тег элемента источника данных (рисунок 2.1.3) и выберите **Настроить источник данных (Configure Data Source)** для запуска мастера **Настройка источника данных (Configure Data Source)**.

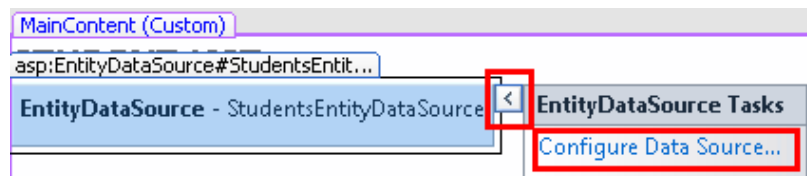


Рисунок 2.1.3 Выбор режима настройки источника данных

- На первом шаге мастера (рис.2.1.4) **НастройкаObjectContext (ConfigureObjectContext)** выберите для переключателя **Именованное соединение (Named Connection)** значение **SchoolEntities** и для **DefaultContainerName** тоже должно быть значение **SchoolEntities**. Нажмите **Далее (Next)**.

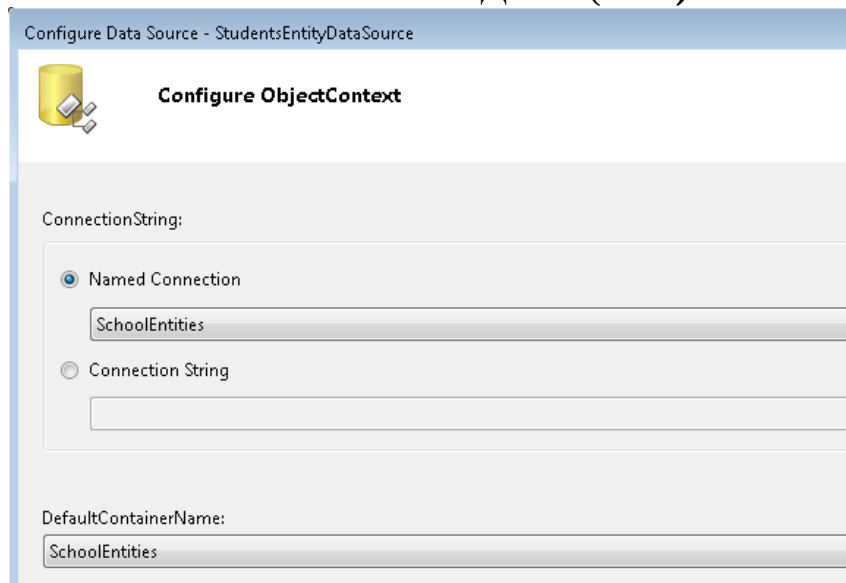


Рисунок 2.1.4 Выбор способа подключения

Замечание: Если вы получили сообщение об ошибке загрузки данных для строки соединения, то возможно забыли построить проект перед настройкой элемента источника данных.

8. На шаге **Настройка выбора данных (Configure Data Selection)** (рисунок 2.1.5) выберите для набора **EntitySetName** значение **People**. В поле **Select** поставьте флажок **Select All**. Установите также флажки автоматического обновления и удаления (enable update and delete). Нажмите **Готово (Finish)**.

Configure Data Source - StudentsEntityDataSource

Configure Data Selection

EntitySetName:
People

EntityTypeFilter:
(None)

Select:

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

☐ Enable automatic inserts

☒ Enable automatic updates

☒ Enable automatic deletes

< Previous Next > **Finish**

Рисунок 2.1.5 Настройка выбора данных

9. Сохраните изменения.
10. Откройте страницу *Students.aspx* в режиме разметки и изучите код, добавленный матером.

Упражнение 2. Настройка правила: разрешить удаление

В этом упражнении вы создадите страницу, которая позволит пользователям удалять студентов из таблицы *Person*, которая имеет связи с другими таблицами (*Course*, *StudentGrade* и *OfficeAssignment*).

По умолчанию, база данных ограничивает удаление строки в *Person*, если есть связанные строки в одной из других таблиц. Вы можете вручную удалить связанные строки, или лучше настроить базу данных так, чтобы удалять их автоматически, когда вы удалите строку в *Person*.

Для записей о студентах в этом упражнении вы настроите базу данных для автоматического удаления связанных данных. Так как студенты могут

иметь связанные строки только в таблице StudentGrade, необходимо настроить только одно из трех отношений.

1. Откройте диаграмму базы данных.
2. В контекстном меню отношения между таблицами Person и StudentGrade (рисунок 2.2.1) выберите **Свойства (Properties)**.

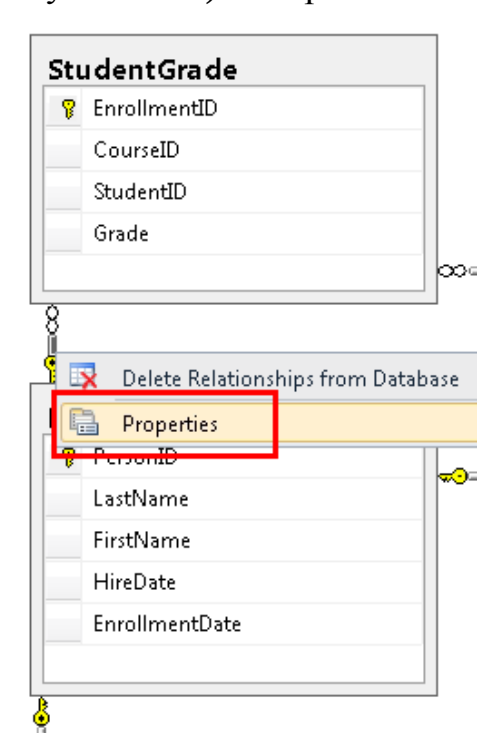


Рисунок 2.2.1 Отношение между таблицами

3. В окне свойств (рисунок 2.2.2) раскройте **Спецификация INSERT и UPDATE (INSERT and UPDATE Specification)** и установите для свойства правила удаления (**DeleteRule**) значение **Каскадом (Cascade)**.

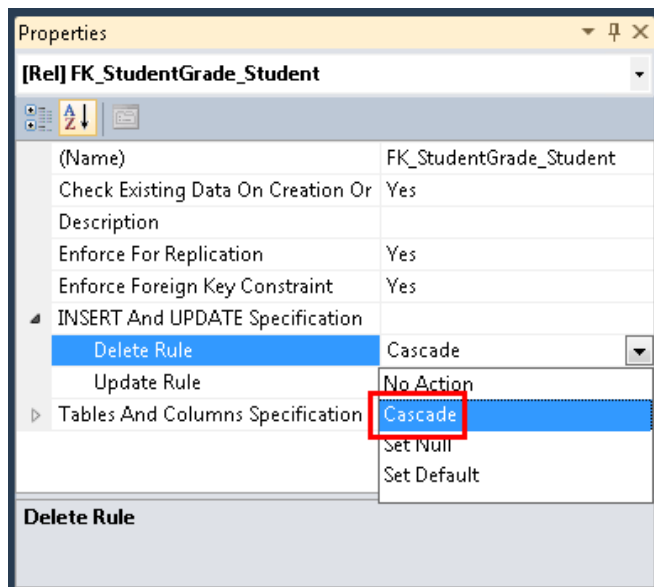


Рисунок 2.2.2 Настройка правил отношения

4. Сохраните изменения и закройте диаграмму. Если перед сохранением появится окно с вопросом - хотите ли вы обновить базу данных? - согласитесь.

Чтобы убедиться, что модель поддерживает объекты, которые находятся в памяти, синхронизированы с тем, что содержит база данных, вы должны установить соответствующие правила в модели данных.

5. Откройте модель *SchoolModel.edmx* в режиме дизайнера и в контекстном меню связи между *Person* и *StudentGrade*, выберите **Свойства (Properties)**, как показано на рисунке 2.2.3:

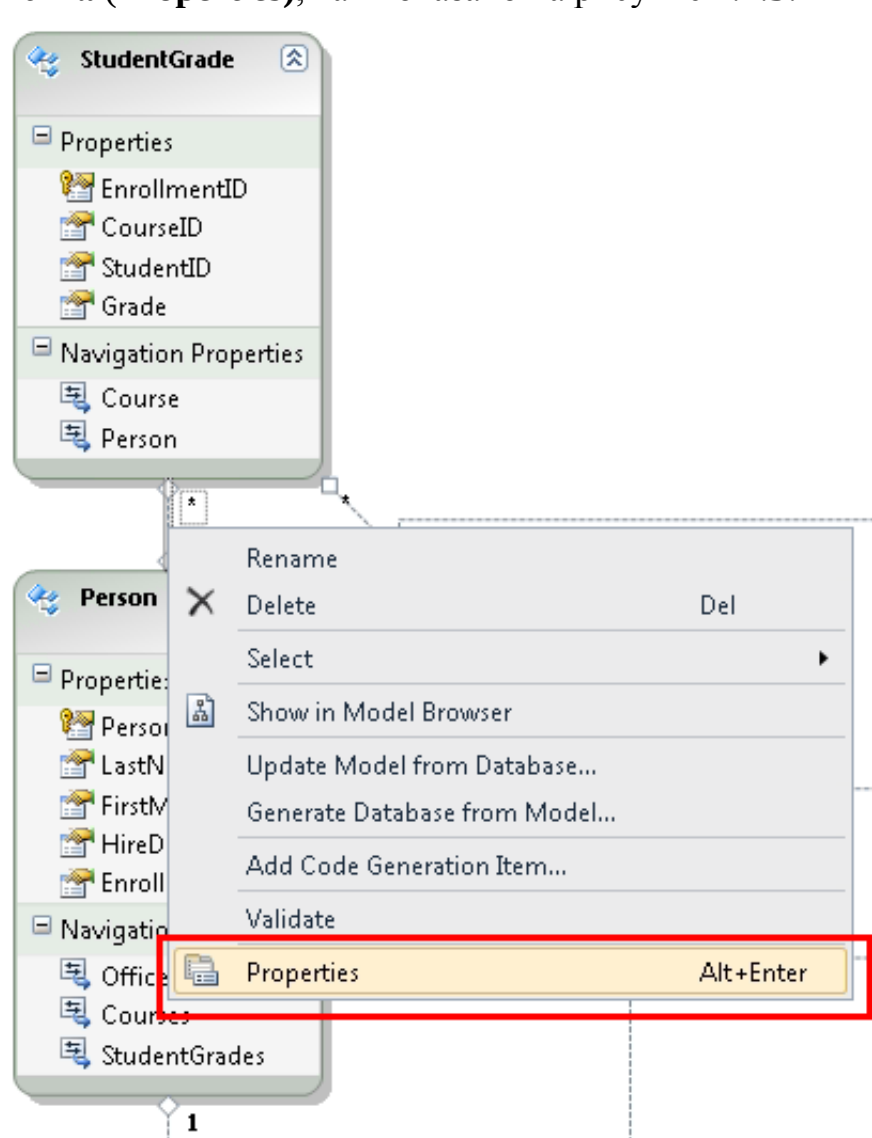


Рисунок 2.2.3 Открытие свойств связи между сущностями

6. На панели свойств для свойства **Событие OnDelete** элемента **End1 (End1 OnDelete)** установите значение **Cascade**, как на рисунке 2.2.4:

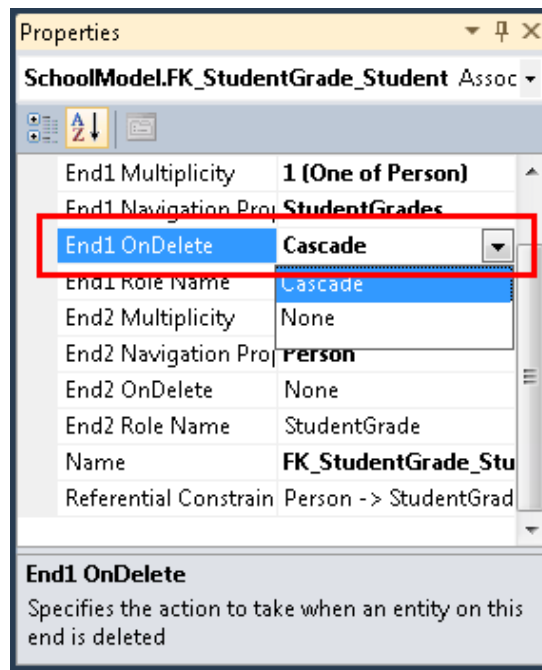


Рисунок 2.2.4. Свойства отношения

7. Сохраните и закройте файл *SchoolModel.edmx*.
8. Перестройте проект.

Упражнение 3. Применение GridView для чтения и обновления данных

В этом упражнении вы используете элемент GridView для отображения, обновления или удаления студентов.

1. Откройте файл *Students.aspx* в режиме конструктора.
2. Откройте панель инструментов (**Toolbox**) и из вкладки **Данные (Data)** перенесите элемент GridView справа от элемента *EntityDataSource*.
3. С помощью окна свойств укажите свойству (ID) значение *StudentsGridView*.
4. Кликните смарт тег элемента GridView и выберите **StudentsEntityDataSource** в качестве источника данных, как на рисунке 2.3.1:

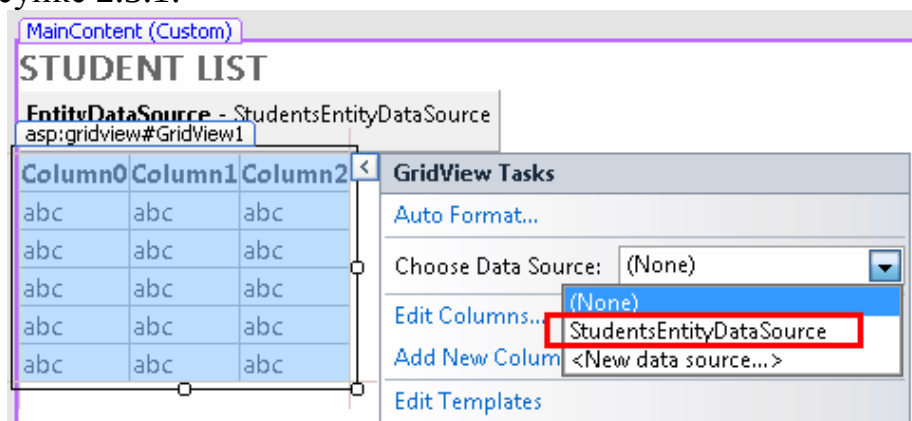


Рисунок 2.3.1 Выбор источника данных

5. Кликните **Обновить схему (Refresh Schema)** (согласитесь если появится вопрос-предупреждение).
6. Установите флажки **Включить постраничный просмотр (Enable Paging)**, **Включить сортировку (Enable Sorting)**, **Включить правку (Enable Editing)** и **Включить удаление (Enable Deleting)**.
7. Кликните **Правка столбцов (Edit Columns)**.

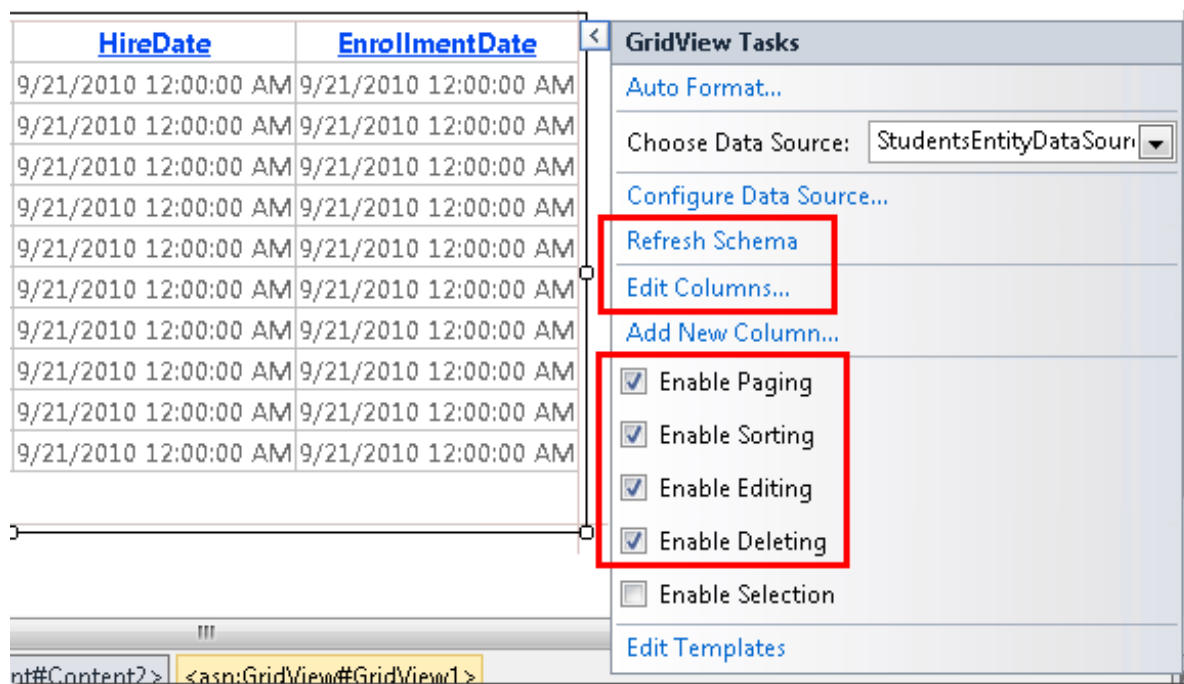


Рисунок 2.3.2 Настройка столбцов

8. В окне **Выбранные поля (Selected fields)** удалите **PersonID**, **LastName** и **HireDate** (см. рисунок 2.3.3). Как правило на практике эти столбцы не нужны.

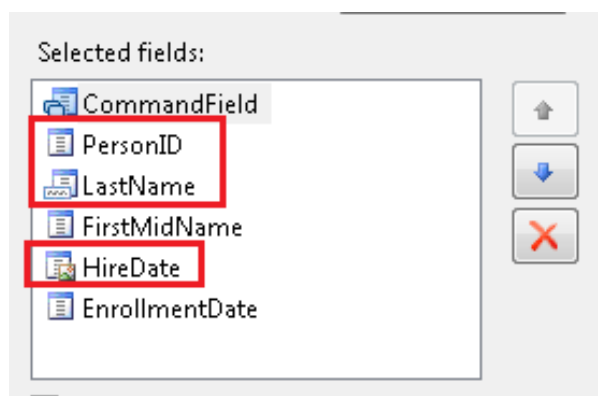


Рисунок 2.3.3 Выбор столбцов

9. Выберите поле **FirstMidName** и кликните **Преобразовать это поле в TemplateField (Convert this field into a TemplateField)** (см. рисунок 2.3.4).
10. Сделайте тоже самое и с полем **EnrollmentDate**.

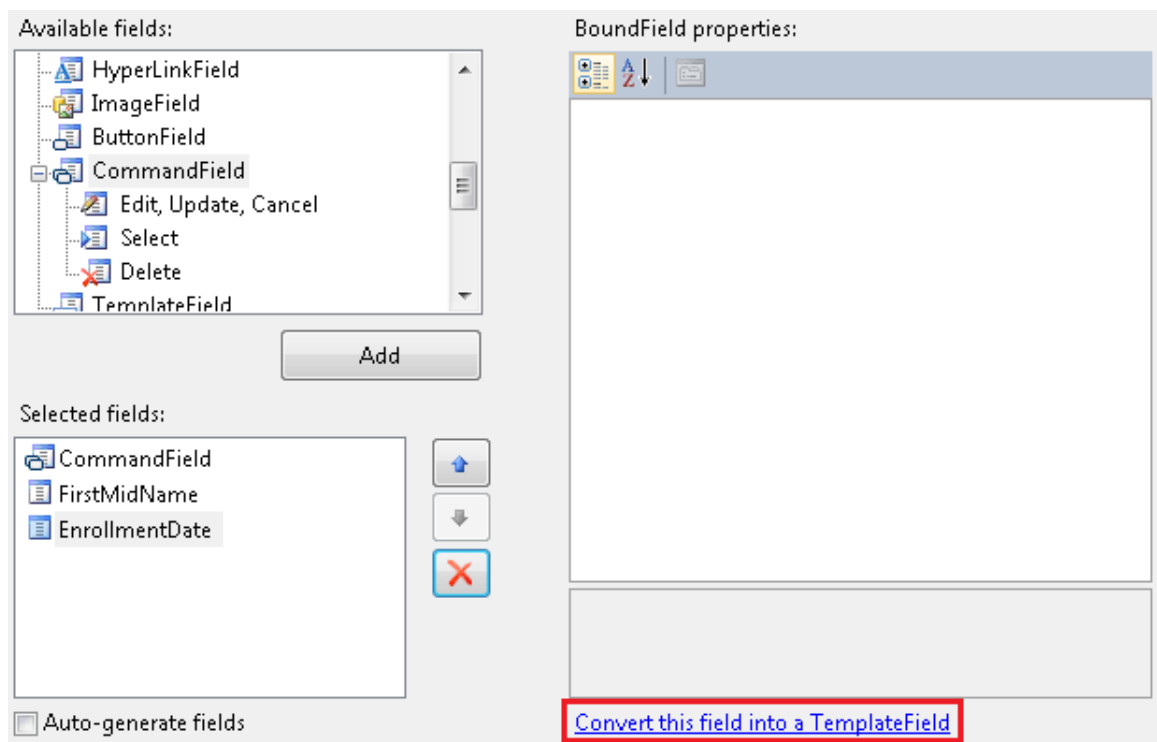


Рисунок 2.3.4 Преобразование поля

11. Нажмите **ОК** и переключитесь в режим разметки.

12. Изучите изменения, которые внесены в разметку. Элемент

GridView должен представляться следующим образом:

```
<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>
            <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>

```

13. Первый столбец после командного поля является шаблонным полем и отображает имя студента. Измените код разметки для более наглядного отображения:

```

        <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>

```

В режиме отображения два элемента Label отображают имя и фамилию. В режиме редактирования отображаются два текстовых поля для реализации возможности изменения имени и фамилии.

14. Последний столбец является шаблонным полем, который отображает дату зачисления студента на курс. Измените код разметки для более наглядного отображения:

```

        <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>

```

Обратите внимание на формат "{0,d}".

Упражнение 4. Настройка EntityDataSource для улучшения производительности

В этом упражнении вы внесете изменения в настройку элемента EntityDataSource для улучшения производительности.

`ConnectionString="name=SchoolEntities" DefaultContainerName="SchoolEntities"`

1. В коде разметки элемента EntityDataSource удалите атрибуты `ConnectionString` и `DefaultContainerName` и замените их на атрибут `ContextTypeName="ContosoUniversity.DAL.SchoolEntities"`.

Код разметки сейчас должен быть похож на следующий (порядок свойств может быть различным):

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

2. Запустите страницу в браузере и просмотрите список студентов. Имя и фамилия отображаются в одном поле.
3. Для сортировки в столбце кликните по имени столбца.
4. Кликните для правки данных строки по **Edit**. Отобразятся текстовые поля для изменения имени и фамилии.
5. Проверьте работу кнопки **Delete**. Удалите любую строку, которая имеет дату зачисления.

Удаление строк, не содержащих дату зачисления, приведет к ошибке. В дальнейшем эта ошибка будет устранена.

Упражнение 5. Отображение данных с помощью свойства навигации (Navigation Property)

Свойства навигации предоставляют способ перемещения по ассоциации между типами сущностей. Каждый объект может обладать свойством навигации для каждого отношения, в котором участвует. Свойства навигации позволяют передвигаться по связям и управлять ими в обоих направлениях, а также возвращают ссылочный объект (если кратность равна одному либо нулю или одному) или коллекцию (если кратность больше одного).

В этом упражнении вы добавите возможность вывода количества курсов, на которых обучается студент. Entity Framework предоставляет эту информацию в навигационном свойстве `StudentGrades` сущности `Person`.

1. Откройте файл `Students.aspx` в режиме конструктора.
2. Выделите элемент `StudentsEntityDataSource` и в окне свойств для свойства **Include** (см. рисунок 2.5.1) установите значение **StudentGrades** (если вы хотите получить несколько свойств

навигации, вы можете указать их имена через запятую, например, **StudentGrades, Courses.**)

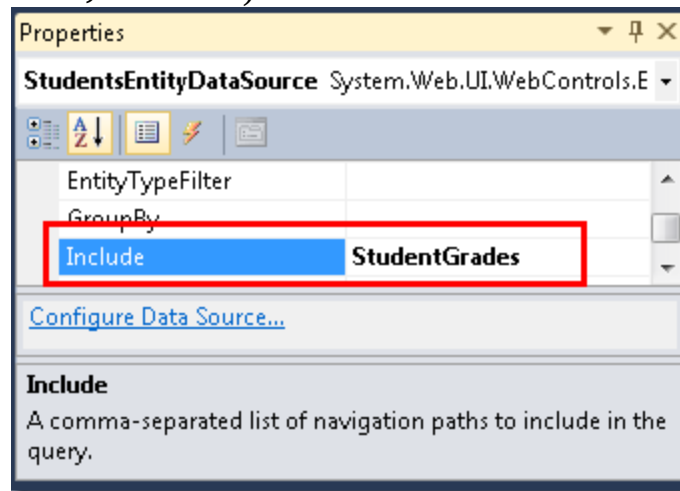


Рисунок 2.5.1 Свойства источника данных

3. Переключитесь в режим разметки кода. В элементе `StudentsGridView` после последнего элемента `asp:TemplateField` добавьте следующее поле:

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

В выражении `Eval` используется свойство `StudentGrades` поскольку оно содержит коллекцию, имеющее свойство `Count`, которое можно использовать для отображения количества курсов, в которых студент обучается.

4. Запустите приложение и проверьте, что таблица отображает количество курсов для каждого студента.

Упражнение 6. Применение `DetailsView` для вставки данных

В этом упражнении вы создадите страницу с элементом `DetailsView`, который позволит добавлять новых студентов.

1. Добавьте в проект новую веб-страницу, используя главную страницу `Site.Master`. Назовите страницу `StudentsAdd.aspx`.
2. В коде разметки новой страницы добавьте текст `Add New Students` стиля заголовка `h2` в элемент содержимого `Content` с именем `Content2`:

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">
    <h2>
        Add New Students </h2>
</asp:Content>
```

3. Откройте страницу `StudentsAdd.aspx` в обозревателе и убедитесь, что добавленная строка отображается как надо.

4. Откройте страницу *Students.aspx* в коде разметки, скопируйте код элемента `EntityDataSource` и вставьте его в файл *StudentsAdd.asp* и добавьте свойство, разрешающее операцию вставки:

`EnableInsert="True"`

5. Добавьте следующий код для использования элемента `DetailsView`:

```
<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>
```

Как и в элементе `GridView` связанные поля элемента `DetailsView` кодируются так, как будто они будут работать для управления данными из базы данных, за исключением того, что они ссылаются на свойства сущностей. В данном случае `DetailsView` используется только для вставки строк, так что включен режим по умолчанию для вставки.

6. Запустите приложение и протестируйте вставку данных студента.
7. Откройте список студентов и проверьте, что новый студент успешно добавлен.

Упражнение 7. Отображение данных в списке Drop-Down

В этом упражнении вы реализуете связь элемента `DropDownList` с данными с помощью элемента `EntityDataSource`. В следующих упражнениях вы будете использовать список, чтобы позволить пользователям выбрать кафедру и отобразить курсы, связанные с этой кафедрой.

1. Добавьте в проект новую веб-страницу, используя главную страницу *Site.Master*. Назовите страницу *Courses.aspx*.
2. В коде разметки новой страницы добавьте текст `Courses by Department` стиля заголовка `h2` в элемент содержимого `Content` с именем `Content2`:

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

3. В режиме конструктора добавьте элемент `EntityDataSource` и укажите (**ID**) как `DepartmentsEntityDataSource`.
4. Настройте источник данных: на первом шаге мастера **НастройкаObjectContext (Configure ObjectContext)** выберите для переключателя **Именованное соединение (Named Connection)** значение **SchoolEntities** и для **DefaultContainerName** тоже должно быть значение **SchoolEntities**.
5. На шаге **Настройка выбора данных (Configure Data Selection)** (см. рисунок 2.7.1) выберите для набора **EntitySetName** значение **DepartmentID**. В поле **Select** укажите только **Departments** и **Name**.

Рисунок 2.7.1 Настройка набора данных

6. Перенесите с панели инструментов элемент `DropDownList`, укажите (**ID**) как `DepartmentsDropDownList`, кликните смарт тег и выберите **Выбрать источник данных (Choose Data Source)** для запуска мастера **DataSource Configuration Wizard**.

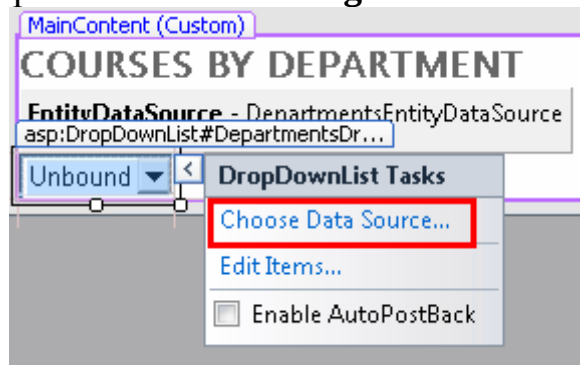


Рисунок 2.7.2 Смарт-тег

7. На шаге **Выбор источника данных (Choose a Data Source)** (см. рисунок 2.7.3) выберите **DepartmentsEntityDataSource** как источник данных, кликните **Обновить схему (Refresh Schema)**, и в первом списке укажите поле данных для отображения **Name** и во втором поле для отображения значений **DepartmentID**. Нажмите **ОК**.

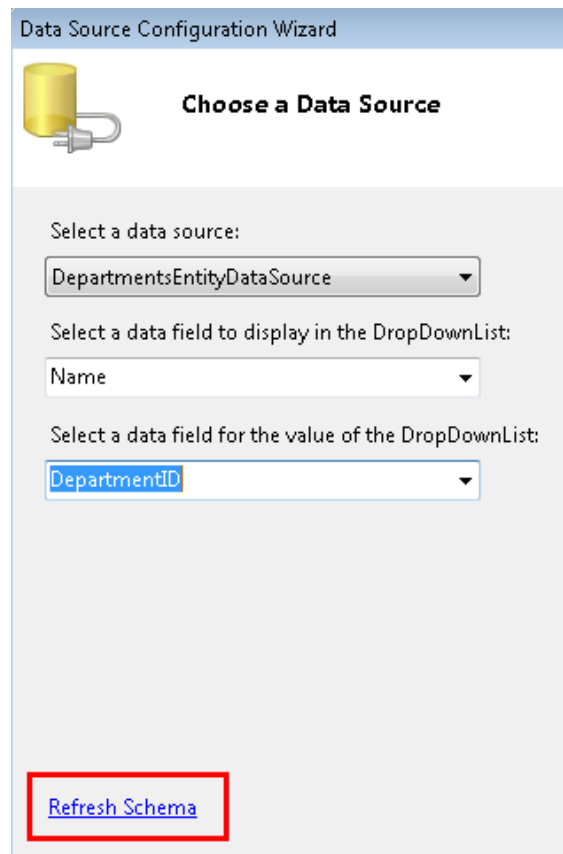


Рисунок 2.7.3. Выбор источника данных

8. В коде разметки страницы добавьте фразу "Select a department:" прямо перед элементом DropDownList:

```

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

```

9. Запустите страницу в обозревателе и протестируйте возможность выбора элемента списка.

В результате выполненных упражнений вы настроили элемент `EntityDataSource`. Работа с этим элементом управления, как правило, мало отличается от работы с другими элементами управления источника данных ASP.NET. Единственное исключение, когда вы хотите получить доступ к навигационным свойствам.

Лабораторная работа 3. Фильтрация, упорядочивание и группирование данных

В предыдущей работе вы использовали элемент `EntityDataSource` для отображения и редактирования данных. В этой работе вы реализуете фильтрацию, порядок отображения определенного свойства и группировку данных.

Вы измените страницу *Students.aspx* для фильтрации студентов, сортировки и поиска их по имени, и поиск по имени. Вы также измените страницу *Courses.aspx* для отображения курсов для выбранной кафедры и поиска курсов по названию. Наконец, вы добавите статистику по студентам к странице *About.aspx*.

Упражнение 1. Настройка свойств элемента *EntityDataSource* для более наглядного отображения данных

Использование свойства "Where" элемента *EntityDataSource* для фильтрации данных

В этой части упражнения вы внесете изменения в элемент *EntityDataSource* для того, чтобы *GridView* мог отображать только студентов, которые зачислены (т.е. имеют даты зачисления).

1. Откройте в режиме конструктора страницу *Students.asp*, созданную в предыдущей работе.
2. Выделите *EntityDataSource*. В окне свойств установите для свойства *Where* значение `it.EnrollmentDate is not null`.

Синтаксис, используемый в свойстве *Where* элемента *EntityDataSource*, соответствует *Entity SQL*. В выражении `it.EnrollmentDate IS NOT NULL`, слово *it* представляет собой ссылку на сущность, возвращаемую запросом. Таким образом, `it.EnrollmentDate` относится к свойству *EnrollmentDate* человека, что возвращает управление *EntityDataSource*.

3. Откройте страницу в обозревателе. Будут отображены только те студенты, которые имеют дату зачисления хотя бы на один курс.

Использование свойства "OrderBy" элемента *EntityDataSource* для определения порядка отображения данных

4. Определите порядок отображения студентов в списке, например, сначала фамилия, потом имя: для свойства **OrderBy** элемента *EntityDataSource* установите значение `it.LastName`.
5. Откройте страницу в обозревателе. Проверьте реализованную функциональность.

Использование параметров элемента управления в свойстве "Where"

Можно передать значения параметров в свойство *Where*. На странице *Courses.aspx*, вы можете это использовать для отображения курсов, связанных с кафедрой, которую пользователь выбирает из выпадающего списка.

6. Откройте *Courses.aspx* в режиме конструктора.
7. Добавьте на страницу второй элемент *EntityDataSource*, установите свойству **(ID)** имя `CoursesEntityDataSource`.

- Настройте источник данных: установите соединение с моделью SchoolEntities и выберите в качестве набора EntitySetName значение Courses.
8. В окне свойств элемента CoursesEntityDataSource кликните на три точки напротив свойства **Where**.
 9. В окне **Редактор выражений (Expression Editor)** (см. рисунок 3.1.1) выберите **Автоматически создавать выражение Where на основе параметров (Automatically generate the Where ...)** и кликните **Добавить параметр (Add Parameter)**.
 10. Укажите имя параметру – DepartmentID, в списке **Источник параметров (Parameter source)** выберите **Control**, в списке **ControlID** – DepartmentsDropDownList.

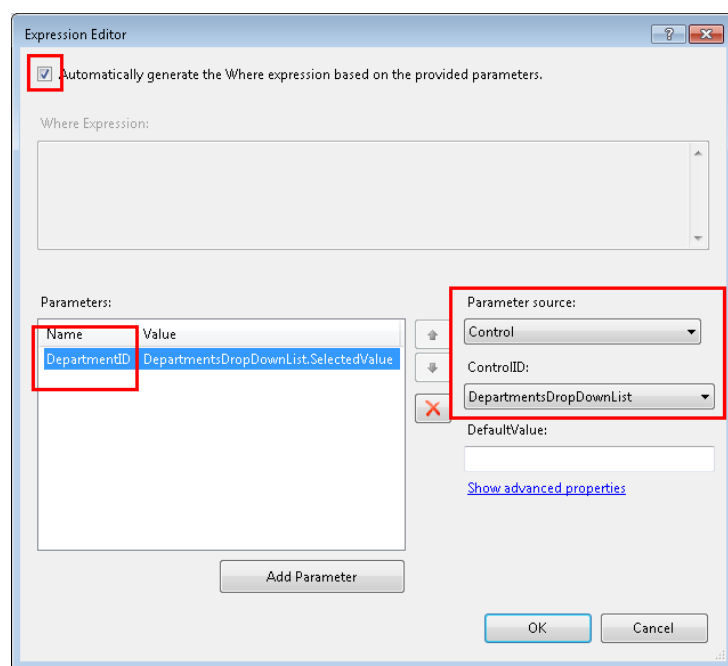


Рисунок 3.1.1 Редактор выражений

11. Кликните **Показать дополнительные (Show advanced properties)** и в окне свойств редактора выражений для свойства Type укажите Int32.
12. Нажмите **OK**.
13. Ниже выпадающего списка добавьте элемента GridView и укажите (ID) как CoursesGridView.
14. Для нового элемента укажите источник данных – CoursesEntityDataSource, кликните **Обновить схему (Refresh Schema)**, далее кликните **Правка столбцов (Edit Columns)**, удалите столбец DepartmentID. Код разметки элемента GridView должен быть следующим.

```
<asp:GridView ID="CoursesGridView" runat="server"
AutoGenerateColumns="False"
```

```

        DataKeyNames="CourseID"
DataSourceID="CoursesEntityDataSource">
    <Columns>
        <asp:BoundField DataField="CourseID"
HeaderText="CourseID" ReadOnly="True"
SortExpression="CourseID" />
        <asp:BoundField DataField="Title"
HeaderText="Title" SortExpression="Title" />
        <asp:BoundField DataField="Credits"
HeaderText="Credits"
SortExpression="Credits" />
    </Columns>
</asp:GridView>

```

Когда пользователь выберет название кафедры (department) в выпадающем списке, список курсов должен изменяться автоматически соответственно кафедре.

15. Для реализации этого выделите элемент выпадающий список и в окне свойств для свойства `AutoPostBack` установите значение `True`.

16. Перейдите в режим разметки кода и замените свойства `ConnectionString` и `DefaultContainer` элемента `CoursesEntityDataSource` на `ContextTypeName="ContosoUniversity.DAL.SchoolEntities"`.

Код разметки должен быть следующим.

```

<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>

```

17. Запустите страницу `Courses.aspx` в обозревателе. В выпадающем списке выберите любой элемент и проверьте изменение списка курсов в элементе `GridView`.

Использование свойства "GroupBy" элемента `EntityDataSource` для группирования данных

18. Откройте страницу `About.aspx` в режиме разметки, замените существующее содержимое элемента `BodyContent` на "Student Body Statistics" между тегами `h2`, а между `<p>` удалите:

```

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

```

19. После заголовка h2 добавьте элемент EntityDataSource, укажите ID, равным StudentStatisticsEntityDataSource.

20. В режиме конструктора настройте источник данных: соединение SchoolEntities, набор данных EntitySetName – people, остальные параметры оставьте по умолчанию.

21. Установите в окне свойств:

- для фильтрации только студентов свойству Where значение it.EnrollmentDate is not null.
- для группирования по дате зачисления свойству GroupBy установите значение it.EnrollmentDate.
- для отбора по дате и подсчета количества студентов свойству Select установите it.EnrollmentDate, Count(it.EnrollmentDate) AS NumberOfStudents.
- для определения порядка вывода результатов по дате свойству OrderBy установите it.EnrollmentDate.

Код разметки элемента EntityDataSource должен быть следующим.

```

<asp:EntityDataSource ID="StudentStatisticsEntityDataSource"
runat="server"
    ConnectionString="name=SchoolEntities"
DefaultContainerName="SchoolEntities"
    EnableFlattening="False" EntitySetName="People"
GroupBy="it.EnrollmentDate"
    OrderBy="it.EnrollmentDate"
    Select="it.EnrollmentDate, Count(it.EnrollmentDate) AS
NumberOfStudents"
    Where="it.EnrollmentDate is not null">
</asp:EntityDataSource>

```

22. Добавьте следующий код для создания элемента GridView для отображения данных:

```

<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>

```

23. Запустите страницу *About.aspx* в обозревателе и просмотрите результаты статистики по датам зачисления.

Упражнение 2. Реализация поиска данных

Применение элемента `QueryExtender` для фильтрации и упорядочивания

Элемент `QueryExtender` обеспечивает способ определения фильтрации и сортировки в разметке. В этой части упражнения вы будете использовать элемент `QueryExtender` для фильтрации и упорядочивания данных, используя навигационное свойство.

1. Откройте страницу *Courses.aspx* в коде разметки и ниже кода `GridView` вставьте следующий код для создания заголовка, текстового поля для ввода строки поиска и кнопку поиска, а также добавьте код настройки элемента `EntityDataSource`:

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

Обратите внимание, что свойство `Include` элемента управления `EntityDataSource` устанавливается как `Department`. В базе данных таблица `Course` не содержит название кафедры, она содержит столбец внешнего ключа `DepartmentID`. Если бы вы выполняли запросы к базе данных напрямую для получения названия кафедры вместе с данными курса, вы должны были соединить таблицы `Course` курс и `Department`.

Установив свойству `Include` значение `Department`, вы указали, что `Entity Framework` должен обеспечить получение соответствующего объекта `Department`, когда он получает объект `Course`. Объект `Department` затем хранится в навигационном свойстве объекта `Course`.

2. После элемента `EntityDataSource` вставьте следующий код разметки для создания элемента `QueryExtender`, связанного с `EntityDataSource`:

```
<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>

```

Элемент `SearchExpression` определяет выражение поиска курса. Свойство `SearchType` определяется как `StartsWith`. Это означает, что поиск должен определить, совпадает ли строка поиска с какой-либо строкой в начале поля.

Элемент `OrderByExpression` указывает, что набор результатов будет упорядочен по названию курса в названии кафедры. Обратите внимание, как указано название отдела: `Department.Name`. Поскольку связь между объектом `Course` и `Department` является один-к-одному, свойство навигации `Department` содержит сущность `Department`. Если бы это было отношения один-ко-многим, свойство содержало бы коллекцию. Чтобы получить названия кафедры, необходимо указать свойство `Name` сущности `Department`.

3. Добавьте элемент `GridView` для отображения результатов требуемого запроса:

```

<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>

```

Первым столбцом является шаблонное поле, которое отображает название кафедры. Выражение привязки данных определяет `Department.Name`, как вы видели в элементе `QueryExtender`.

4. Запустите страницу в обозревателе. Введите название курса и просмотрите результаты.
5. Введите букву "m" и кликните кнопку **Search** для просмотра всех курсов, начинающихся с этой буквы.

Применение оператора "Like" Operator to Filter Data

Вы можете достичь эффекта, аналогичного тому, который предоставляется элементом QueryExtender со свойствами startsWith, Contains и EndsWith, путем использования оператора Like в свойстве Where элемента управления EntityDataSource. В этой части упражнения вы увидите, как использовать оператор Like для поиска студента по имени.

6. Откройте файл *Students.aspx* в режиме разметки кода. После элемента GridView добавьте следующую разметку:

```
<h2>Find Students by Name</h2>
Enter any part of the name
<asp:TextBox ID="SearchTextBox" runat="server"
AutoPostBack="true"></asp:TextBox>
<asp:Button ID="SearchButton" runat="server" Text="Search" />
<br />
<br />
<asp:EntityDataSource ID="SearchEntityDataSource"
runat="server"
ConnectionString="name=SchoolEntities"
DefaultContainerName="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>

```

7. Запустите страницу в обозревателе. Initially you see all of the students because the default value for the `StudentName` parameter is "%".
8. Введите букву "g" в текстовое поле и кликните **Search**. Вы увидите список студентов, которые имеют "g" в имени или фамилии.

Лабораторная работа 4. Работа со связанными данными

В прошлой работе вы настроили элемент `EntityDataSource` для фильтрации, сортировки и группирования данных. В этой работе вы создадите страницу *Инструкторы*, которая отобразит список преподавателей (инструкторов). При выборе инструктора, вы увидите список курсов, преподаваемых этим инструктором. При выборе курса вы увидите подробности курса и список студентов, обучающихся на курсе.

Упражнение 1. Отображение и обновление связанных данных в элементе `GridView`

1. Создайте новую страницу *Instructors.aspx*, основанная на странице *Site.Master* и добавьте следующую разметку в элемент `Content` с именем `Content2`:

```

<h2>Instructors</h2>
<div>
    <asp:EntityDataSource ID="InstructorsEntityDataSource"
runat="server"
    ConnectionString="name=SchoolEntities"
DefaultContainerName="SchoolEntities"
    EnableFlattening="False"
    EntitySetName="People"
    Where="it.HireDate is not null"
Include="OfficeAssignment" EnableUpdate="True">
    </asp:EntityDataSource>
</div>

```

Эта разметка создает элемент `EntityDataSource`, который выбирает инструкторов и позволяет выполнять обновления.

2. Между разметкой `EntityDataSource` и закрывающим тегом `</div>` добавьте код разметки для создания элементов `GridView` и `Label` (этот элемент будет отображать сообщения об ошибках):

```

    <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>

```

Элемент `GridView` позволяет выбрать строку, подчеркивает выбранную строку с легким серым фоном, и определяет обработчики событий (которые вы создадите позже) `SelectedIndexChanged` и `Updating`. Он также указывает `PersonId` для свойства `DataKeyNames`, так что значение ключа выбранной строки могут быть переданы другому элементу управления, что вы будете добавлять позже.

В последнем столбце содержится значение офиса инструктора, который хранится в навигационном свойстве сущности `Person`, потому что он приходит от соответствующего экземпляра. Обратите внимание, что элемент `EditItemTemplate` определяет `Eval` вместо `Bind`, потому что `GridView` управления не может непосредственно связываться с навигационными свойствами для того, чтобы обновить их. Вы обновить назначение офис в коде. Чтобы сделать это, вам нужно ссылка элемент `TextBox`, и вы получите и сохраните, что в случае события `Init` элемента управления `TextBox`.

После элемента `GridView` добавлен элемент `Label`, который используется для сообщений об ошибках. Свойство `Visible` изначально установлено в `false` и состояние просмотра отключено, так что сообщение появится только тогда, когда произойдет ошибка.

3. Откройте файл *Instructors.aspx.cs* и добавьте выражение `using`:

```
using ContosoUniversity.DAL;
```

4. Добавьте ссылку на текстовое поле непосредственно после объявления класса:

```
private TextBox instructorOfficeTextBox;
```

5. Добавьте заглушку обработчика события `SelectedIndexChanged` (код будет добавлен позже).

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

```

6. Добавьте обработчик события `Init` элемента управления `InstructorOfficeTextBox`, в котором сохраните ссылку на элемент `TextBox`. Вы будете использовать эту ссылку, чтобы получить значение, введенное пользователем при обновлении.

```
protected void InstructorOfficeTextBox_Init(object
sender, EventArgs e)
{

```

```

        instructorOfficeTextBox = sender as TextBox;
    }

```

Вы будете использовать событие Updating элемента GridView, чтобы обновить свойство Location, связанное с сущностью OfficeAssignment.

7. Добавьте следующий обработчик для события Updating:

```

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.
    }
}

```

Этот код запускается, когда пользователь нажимает **Update** в строке элемента GridView. Код использует LINQ to Entities, чтобы получить сущность OfficeAssignment, которая связана с текущей сущностью Person, используя PersonId выбранной строки из аргумента события.

8. Запустите страницу *Instructors.aspx* в обозревателе.
9. Кликните **Edit (Правка)**, отобразятся текстовые поля.
10. Измените все значения, в том числе и поле **Office Assignment**. Кликните **Update (Обновить)** и вы увидите изменения, отраженные в списке.

Упражнение 2. Отображение связанных данных в отдельном элементе управления

Каждый преподаватель может быть назначен на один или на несколько курсов. В этом упражнении вы добавите элементы EntityDataSource и GridView для перечисления курсов, связанных с инструктором, выбранным в элементе GridView, отображающем инструкторов.

1. Для создания заголовка (подписи) и элемента EntityDataSource для курсов добавьте следующий код разметки между элементом Label (сообщающий о возможных ошибках) и закрывающим тегом </div>:

```

        <h3>Courses Taught</h3>
        <asp:EntityDataSource ID="CoursesEntityDataSource"
runat="server"
        ConnectionString="name=SchoolEntities"
DefaultContainerName="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>

```

Параметр Where содержит значение PersonID инструктора выбираемой строки в элементе InstructorsGridView.

2. Для создания элемента GridView добавьте следующий код разметки сразу после элемента CoursesEntityDataSource (до тега </div>):
- ```

 <asp:GridView ID="CoursesGridView" runat="server"
 DataSourceID="CoursesEntityDataSource"

```

```

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

```

3. Откройте страницу в обозревателе.

4. Выделите любую строку с инструктором и проверьте, что внизу отображается информация о читаемом им курсе.

Элемент `CoursesGridView` показывает только несколько полей о курсе. Для отображения всей детальной информации о курсе можно использовать элемент `DetailsView` для курса, который выбрал пользователь.

5. Добавьте следующий код разметки после закрывающего тега `</div>`:

```

<div>
 <h3>Course Details</h3>
 <asp:EntityDataSource
ID="CourseDetailsEntityDataSource" runat="server"
ConnectionString="name=SchoolEntities"
DefaultContainerName="SchoolEntities"
EnableFlattening="False"
EntitySetName="Courses"
AutoGenerateWhereClause="False" Where="it.CourseID
= @CourseID"
Include="Department,OnlineCourse,OnsiteCourse,StudentGrades.Per
son"

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>

```

Этот код создает элемент EntityDataSource, который привязан к набору Courses. Свойство Where выбирает курс, используя значение CourseID выбранной строки в элементе GridView, отображающий курсы.

Разметка также определяет обработчик для выбранного события, которое вы будете использовать в дальнейшем для отображения оценки студентов.

6. В файле *Instructors.aspx.cs* создайте заглушку для метода CourseDetailsEntityDataSource\_Selected

```

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

```

7. Откройте страницу в обозревателе.
8. Выберите инструктора и затем выберите курс для просмотра подробной о нем информации.

### ***Упражнение 3. Применение события "Selected" EntityDataSource для отображения связанных данных***

В этом упражнении вы отобразите всех зачисленных студентов для определенного курса. Чтобы сделать это, вы будете использовать событие Selected элемента управления EntityDataSource связанного с курсом в DetailsView.

1. В файл *Instructors.aspx* добавьте код разметки после элемента DetailsView:

```
<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 id="Tr1" runat="server">
 <th id="Th1" runat="server">
 Name
 </th>
 <th id="Th2" 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>
```

Этот код разметки создает элемент ListView, который отображает список студентов и их оценки для выбранного курса. Источник данных не



задан, потому что вы выполните привязку в коде. Элемент `EmptyDataTemplate` обеспечивает сообщение для отображения, когда ничего не выбрано. Элемент `LayoutTemplate` создает HTML таблицу для отображения списка, а `ItemTemplate` определяет столбцы для отображения. ID студента и оценка студента определяются в сущности `StudentGrade`, имя студента выбирается из сущности `Person`, которую Entity Framework делает доступным в свойстве навигации `Person` сущности `StudentGrade`.

2. В файле `Instructors.aspx.cs` реализуйте метод `CourseDetailsEntityDataSource_Selected` следующим образом:

```
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();
 }
}
```

Аргумент этого события обеспечивает выбранные данные в виде коллекции, которая будет иметь нулевые элементы, если ничего не выбрано, или один пункт, если выбран объект `Course`. Если выбран объект `Course`, код использует первый метод, чтобы преобразовать коллекцию к одному объекту. Затем она получает `StudentGrade` объекты из свойства навигации, преобразует их в коллекции, и связывает элемент `GradesListView` в коллекцию.

3. Для очистки отображения в классе `Instructors` создайте следующий метод:

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

4. Вызовите этот метод в обработчике события `Page_Load` страницы `Instructors.aspx` для отображения пустого шаблона при первой ее загрузке.

```
protected void Page_Load(object sender, EventArgs e)
{
```

```

 if (!IsPostBack)
 {
 ClearStudentGradesDataSource();
 }
 }

```

5. Также вызовите этот метод в обработчике события `InstructorsGridView_SelectedIndexChanged` – это событие происходит при выборе инструктора:

```

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

```

6. Откройте страницу в обозревателе.  
 7. Выберите инструктора, который назначен на курс, а затем выберите курс.

### **Лабораторная работа 5. Работа со связанными данными (продолжение)**

В предыдущей работе вы начали использовать элемент `EntityDataSource` для работы со связанными данными. В этой работе вы продолжите работать со связанными данными путем добавления и удаления отношения и добавления нового объекта, который имеет отношение к существующему объекту.

Вы создадите страницу, которая добавляет курсы, распределенные по кафедрам. Кафедры уже существуют, и, когда вы создаете новый курс, в то же время вы будете устанавливать отношения между ним и существующей кафедрой.

Вы также создадите страницу, которая работает с отношением многие-ко-многим, назначая инструктора курса (добавление отношения между двумя сущностями, которые вы выберете) или удаление инструктора из курса (удаление отношения между двумя сущностями). В базе данных, добавив, отношения между преподавателем и курсом в новой строке добавляется к таблице ассоциации `CourseInstructor`; удаление отношения включает в себя удаление строки из таблицы ассоциации `CourseInstructor`. В Entity Framework, настроив свойства навигации, это делается без явного обращения к таблице `CourseInstructor`.

## Упражнение 1. Добавление сущности с отношением к другой сущности

1. Создайте новую страницу *CoursesAdd.aspx*, которая основана на мастер странице *Site.Master* и добавьте код разметки элемента

```
Content с именем Content2:
<h2>Add Courses</h2>
<asp:EntityDataSource ID="CoursesEntityDataSource"
runat="server"
 ConnectionString="name=SchoolEntities"
DefaultContainerName="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>
```

Этот код разметки создает элемент `EntityDataSource`, который выбирает курсы и позволяет добавить строки. Вы будете использовать обработчик события вставки, чтобы обновить свойство навигации `Department`, когда новый курс будет создан.

Далее создается элемент `DetailsView` для добавления новых объектов `Course`. Используются связанные поля для свойств `Course`. Вы будете должны ввести значение `CourseID`, потому что это поле не генерируется системой.

2. В файле *CoursesAdd.aspx.cs* после объявления класса добавьте поле класса, ссылающееся на элемент `DepartmentsDropDownList`:

```
private DropDownList departmentDropDownList;
```

3. Далее добавьте обработчик события `Init` элемента `DepartmentsDropDownList` в котором сохраните ссылку на элемент управления.

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

4. Добавьте обработчик события `Inserting` элемента `DetailsView`:

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

Когда пользователь нажимает `Insert`, то событие `Inserting` вызывается, прежде чем будет вставлена новая запись. Обработчик получает `DepartmentID` от элемента `DropDownList` и использует его, чтобы установить значение, которое будет использоваться для свойства `DepartmentID` сущности `Course`.

5. Запустите страницу *CoursesAdd.aspx* в обозревателе.
6. Введите ID, название и количество кредита и выберите кафедру, затем кликните **Insert (Вставка)**.
7. Откройте страницу *Courses.aspx* и выберите кафедру, ведущую новый курс. Проверьте, что новый курс успешно добавлен.

## **Упражнение 2. Применение отношения многое ко многим**

Отношения между `Courses` и `People` многие-ко-многим. В этом упражнении вы будете добавлять и удалять отношения между `Person` Человек и `Course` курс лиц, обновляя свойства навигации соответствующих сущностей.

1. Создайте новую страницу *InstructorsCourses.aspx* которая основана на мастер странице Site.Master и добавьте код разметки элемента Content с именем Content2:

```
<h2>Assign Instructors to Courses or Remove from
Courses</h2>

<asp:EntityDataSource ID="InstructorsEntityDataSource"
runat="server"
 ConnectionString="name=SchoolEntities"
DefaultContainerName="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_SelectedIndexCh
anged"
 OnDataBound="InstructorsDropDownList_DataBound">
</asp:DropDownList>
<h3>
 Assign a Course</h3>

Select a Course:
<asp:DropDownList ID="UnassignedCoursesDropDownList"
runat="server"
 DataTextField="Title" DataValueField="CourseID">
</asp:DropDownList>

<asp:Button ID="AssignCourseButton" runat="server"
Text="Assign" OnClick="AssignCourseButton_Click" />

<asp:Label ID="CourseAssignedLabel" runat="server"
Visible="false" Text="Assignment successful"></asp:Label>

<h3>
 Remove a Course</h3>

Select a Course:
<asp:DropDownList ID="AssignedCoursesDropDownList"
runat="server"
 DataTextField="title" DataValueField="courseid">
</asp:DropDownList>


```

```

 <asp:Button ID="RemoveCourseButton" runat="server"
Text="Remove" OnClick="RemoveCourseButton_Click" />

 <asp:Label ID="CourseRemovedLabel" runat="server"
Visible="false" Text="Removal successful"></asp:Label>

```

Это код разметки создает элемент `EntityDataSource`, который извлекает имя и `PersonId` объекта `Person` для инструкторов. Элемент `DropDownList` связан с элементом `EntityDataSource`. `DropDownList` определяет обработчик для события `DataBound` с привязкой к данным. Вы будете использовать этот обработчик привязки двух раскрывающихся списков, которые отображают курсы.

Разметка также создает следующую группу элементов управления, чтобы использовать их для присвоения курса к выбранному инструктору:

- `DropDownList` для выбора назначаемого курса. Этот элемент будет заполнен курсами, которые в настоящее время не назначенные выбранному инструктору.
- `Button` для начала выполнения действия.
- `Label` для отображения сообщения об ошибке.

Наконец, разметка также создает группу элементов управления, чтобы использовать удаление курса для выбранного инструктора.

2. В файл `InstructorsCourses.aspx.cs` добавьте выражение `using`:

```
using ContosoUniversity.DAL;
```

3. Добавьте метод для заполнения двух выпадающих списков, отображающих курсы:

```

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;
 }
}

```

В методе выполняется запрос получения всех курсов. Затем определяется, какие курсы назначаются для инструктора и заполняются соответствующие раскрывающиеся списки,

4. Добавьте обработчик события Click для кнопки Assign:

```

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;
 }
}

```

В методе реализовывается получение объекта Person для выбранного инструктора, получение объекта Course для выбранного курса, и добавление выбранного курса для свойства навигации Courses для сущности Person. Затем сохраняются изменения в базу данных и повторно заполняются раскрывающиеся списки, так что результаты можно увидеть сразу.

5. Добавьте обработчик события Click для кнопки Remove:

```
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;
 }
}
```

В методе реализовывается получение объекта Person для выбранного инструктора, получение объекта Course для выбранного курса, и удаление выбранного курса для свойства навигации Courses для сущности Person. Затем сохраняются изменения в базу данных и повторно заполняются раскрывающиеся списки, так что результаты можно увидеть сразу.

6. Добавьте реализацию обработчика события загрузки страницы Page\_Load, в котором реализуется скрывание сообщений об ошибках в случае их отсутствия

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



7. Добавьте обработчики событий `DataBound` и `SelectedIndexChanged` выпадающего списка инструкторов:

```
protected void InstructorsDropDownList_DataBound(object sender, EventArgs e)
{
 PopulateDropDownLists();
}

protected void InstructorsDropDownList_SelectedIndexChanged(object sender, EventArgs e)
{
 PopulateDropDownLists();
}
```

8. Запустите страницу в обозревателе.  
9. Выберите инструктора. В выпадающем списке **Assign a Course** выберите курс и нажмите кнопку **Assign**.  
10. Перейдите на страницу списка инструкторов. Выберите инструктора и убедитесь, что курс добавился для этого инструктора.

## Лабораторная работа 6. Реализация наследования *Table-per-Hierarchy*

В этой работе будет показано, как реализуется наследование в модели данных путем создания наследуемых сущностей, хранящихся в одной таблице базы данных. Вы добавите в модель данных производные сущности и внесете изменения в существующие страницы для их применения.

### Варианты наследования *Table-per-Hierarchy* и *Table-per-Type*

База данных может хранить информацию о связанных объектах в одной таблице или в нескольких таблицах. Например, в базе данных `School`, таблица `Person` включает в себя информацию о студентах и преподавателях в одной таблице. Некоторые из столбцов применяются только к инструкторам (`HireDate`), некоторые только к студентам (`EnrollmentDate`), а некоторые как (`LastName`, `FirstName`) применяются для обеих сущностей.

Вы можете создать новые сущности `Instructor` и `Student`, которые будут наследоваться от сущности `Person`. Этот паттерн создания наследуемых сущностей, хранящихся в одной таблице базы данных называется *table-per-hierarchy* (ТРН) наследование.

Для курсов база `School` применяет другой паттерн (см. рисунок 6.1.1). Online курсы и onsite курсы хранятся в разных таблицах, каждая из которых имеет внешний ключ, связанный с ключом таблицы `Course`.

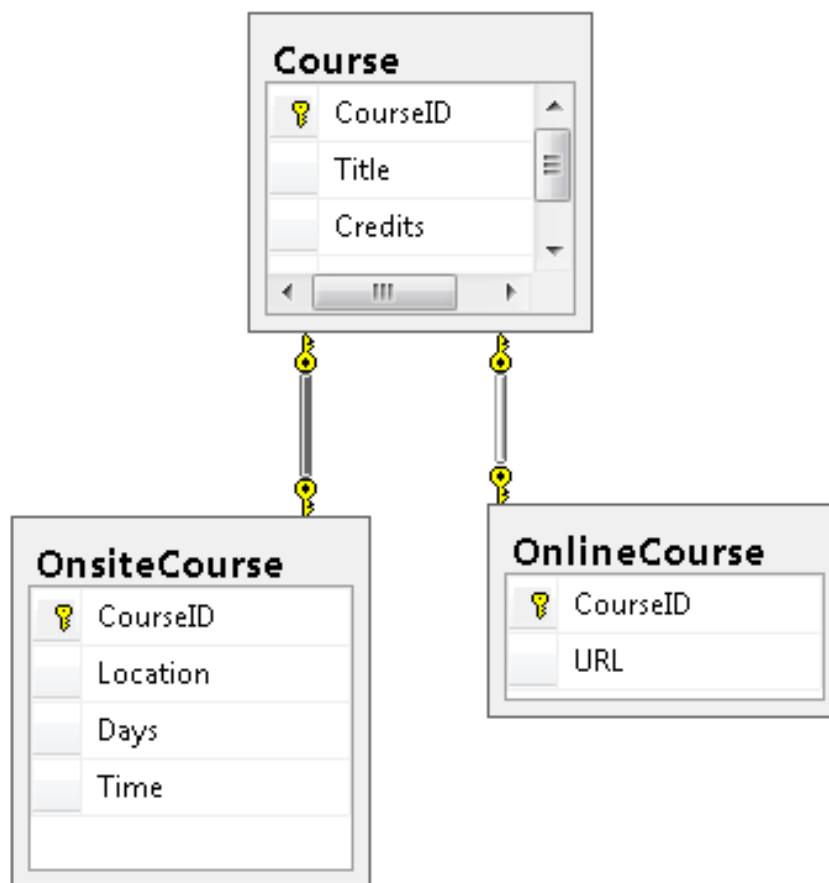


Рисунок 6.1.1 Связь между классами

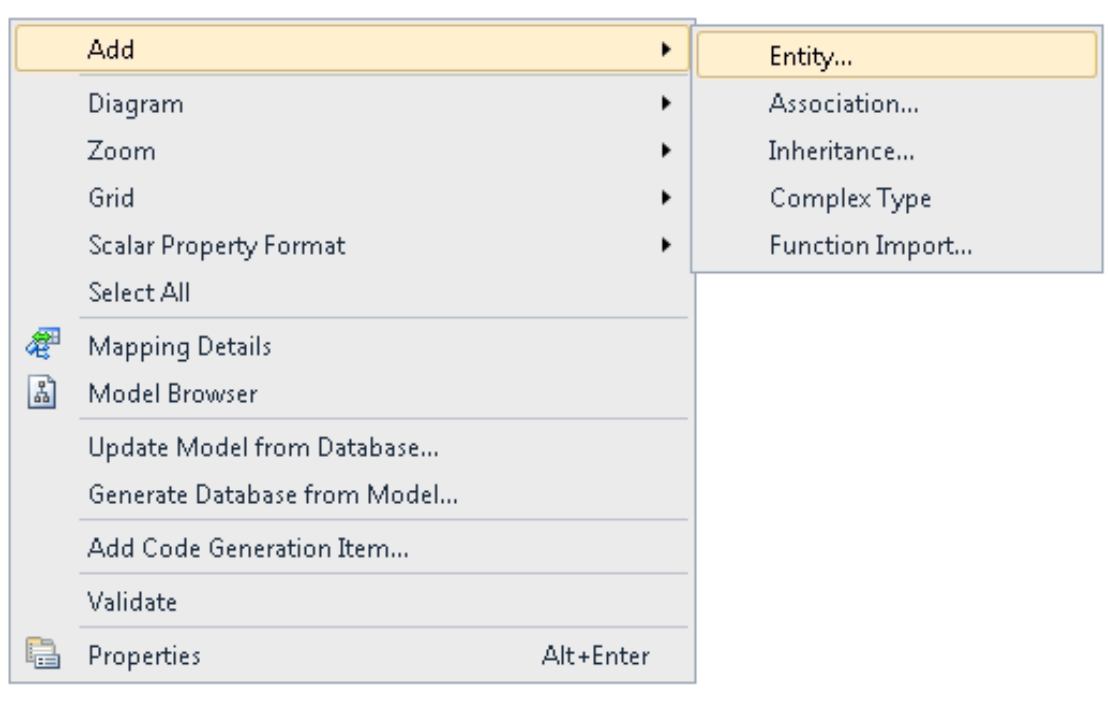
Вы можете создать в модели данных сущности `OnlineCourse` и `OnsiteCourse`, наследуемые от сущности `Course`. Этот паттерн создает наследуемые сущности, находящихся в разных таблицах, а в базовой сущности хранятся общие данные. Такое паттерн называется *table per type* (TPТ) наследование.

Паттерн ТРН обеспечивает, как правило, более высокую производительность в Entity Framework, чем ТРТ, потому что ТРТ может привести к сложным запросам слияния. Для реализации наследования ТРН следует выполнить следующие действия:

1. Создать типы сущностей `Instructor` и `Student` которые наследуются от `Person`.
2. Переместить свойства из базовой сущности, которые относятся к производным сущностям в производные сущности.
3. Установить ограничения на свойства в производных типах.
4. Сделать базовую сущность абстрактной.
5. Сопоставить каждый полученный объект в таблицу `Person` с условием, которое определяет, как определить, представляет строка объект `Person` или производный тип.

## **Упражнение 1. Добавление производных сущностей Instructor и Student**

1. Откройте файл *SchoolModel.edmx* и на пустом месте дизайнера кликните правую кнопку мыши, выберите **Add (Добавить)**, далее (см. рисунок 6.1.2) – **Entity (Сущность)**.



*Рисунок 6.1.2 Контекстное меню добавления элемента*

2. В окне **Add Entity (Добавление сущности)** (см. рисунок 6.1.3) укажите имя – `Instructor` и базовый тип (**Base type**) `Person`. Нажмите **ОК**.

**Add Entity**

**Properties**

Entity name:  
Instructor

Base type:  
Person

Entity Set:  
People

**Key Property**

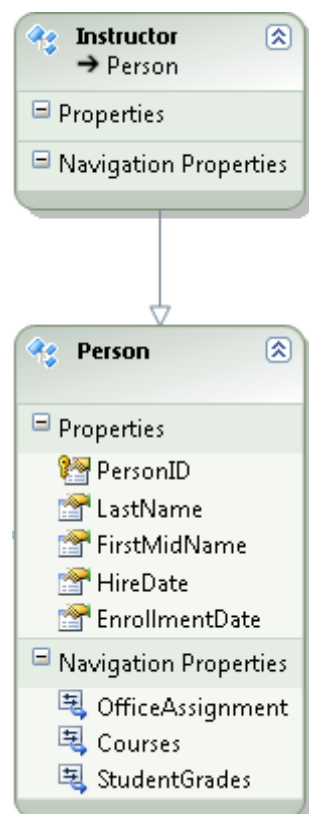
☐ Create key property

Property name:  
Id

Property type:  
Int32

*Рисунок 6.1.3 Добавление сущности*

3. Проверьте (см. рисунок 6.1.4), что дизайнер создал сущность **Instructor**, которая наследуется от сущности **Person**. Новая сущность не имеет свойств.



*Рисунок 6.1.4 Созданные сущности*

4. Повторите аналогичные действия для создания сущности `Student`, которая также наследуется от `Person`.
5. В данной задаче только инструкторы имеют дату приема на работу, поэтому перенесите свойство `HireDate` из сущности `Person` в сущность `Instructor`, используя операции **Cut и Paste**, как показано на рисунке 6.1.5:

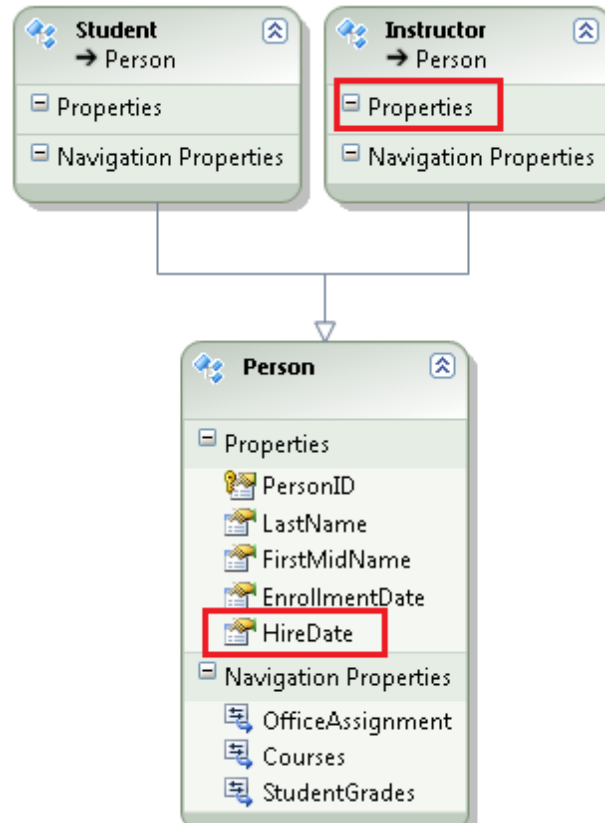


Рисунок 6.1.6 Перенос свойства в базовую сущность

6. Дата приема на работу для сущности `Instructor` не может быть пустой (`null`), поэтому установите свойству `HireDate` сущности `Instructor`, свойство `Nullable` (допускает `Null`) значение `False`.
7. Переместите свойство `EnrollmentDate` из сущности `Person` в сущность `Student`.
8. Для свойства `Nullable` (Допускает `Null`) установите `False` свойству `EnrollmentDate`.

Теперь, когда сущность `Person` имеет только свойства, которые являются общими для инструкторов и студентов, сущность `Person` может быть использована только в качестве базового объекта в структуре наследования. Таким образом, необходимо гарантировать, что эта базовая сущность никогда не будет рассматривается как самостоятельный субъект, т.е. сделать ее абстрактной.

9. В окне свойств сущности `Person` для свойства **Abstract** (Абстрактный) установите **True**.

Теперь вы должны показать Entity Framework, как отличить сущности `Instructor` и `Student` в базе данных.

10. В контекстном меню сущности **Instructor** выберите команду **Table Mapping (Таблица сопоставления)**.
11. В окне **Mapping Details (Сведения о сопоставлении)** (см. рисунок 6.1.7) кликните **Add a Table or View (Добавление таблицы или представления)** и выберите **Person**.

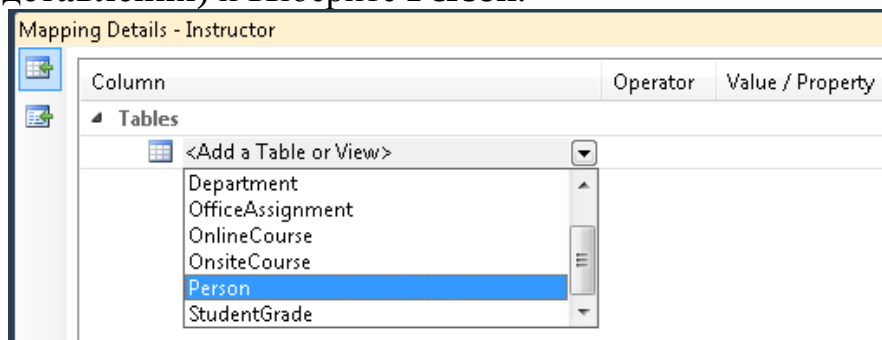


Рисунок 6.1.7 Добавление таблицы

12. Кликните **Add a Condition** и выберите **HireDate** (см. рисунок 6.1.8).

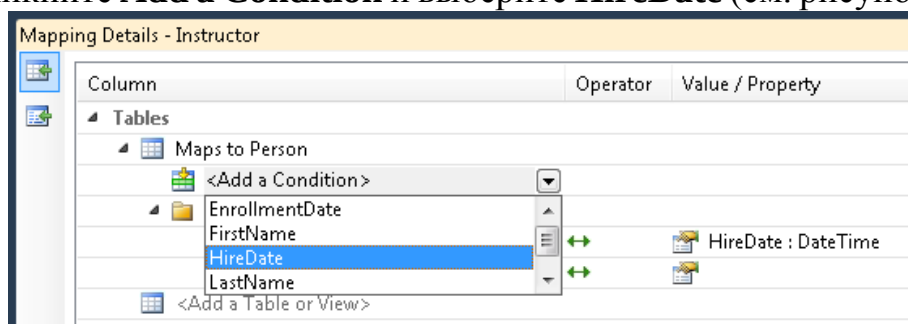


Рисунок 6.1.8 Добавление состояния

13. Измените столбец **Operator** на **Is** и в столбце **Value/Property** установите **Not Null** как показано на рисунке 6.1.8:

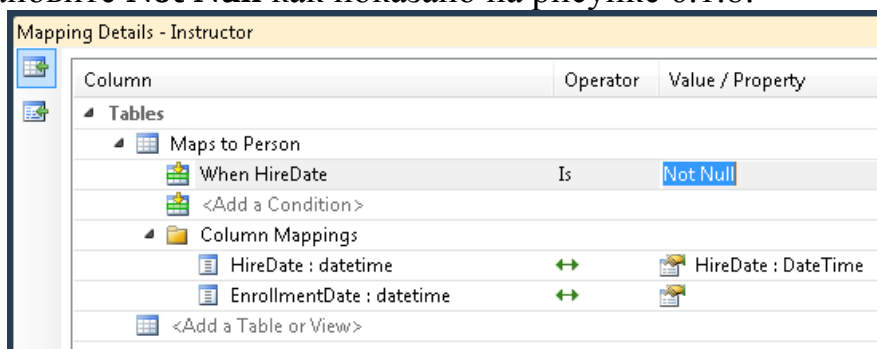


Рисунок 6.1.9 Настройка свойства

14. Повторите процедуру для сущности **Students**, указав, что эта сущность отображает в таблице **Person**, когда **EnrollmentDate** не ноль.
15. Сохраните и закройте модель данных.
16. Постройте проект, чтобы создать новые объекты и сделать их доступными в конструкторе.

## Упражнение 2. Применение сущностей *Instructor* и *Student*

Когда вы создали веб-страницы, которые работают с данными о студентах и инструкторах, вы привязали данные сущности *Person* и отфильтровали свойство *HireDate* или *EnrollmentDate* для ограничения возвращаемых данных для студентов или преподавателей.

Тем не менее, теперь, когда связали каждый элемент источника данных с сущностью *Person*, вы можете указать, что только типы объектов *Student* или *Instructor* должны быть выбраны. Поскольку Entity Framework знает, как отличить студентов и преподавателей в наборе *Person*, вы можете удалить настройки свойства *Where*.

Вы можете это сделать в мастере Visual Studio Designer, указать тип объекта в элементе *EntityTypeDataSource*, как показано, например, на рисунке 6.2.1:

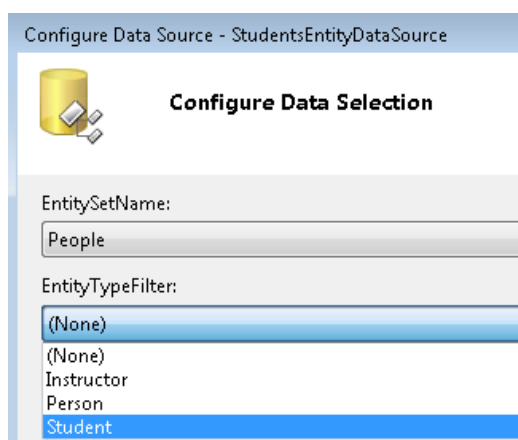


Рисунок 6.2.1 Настройка сущности

В окне свойств **Properties** вы можете удалить значение *Where*, как показано, например на рисунке 6.2.2:

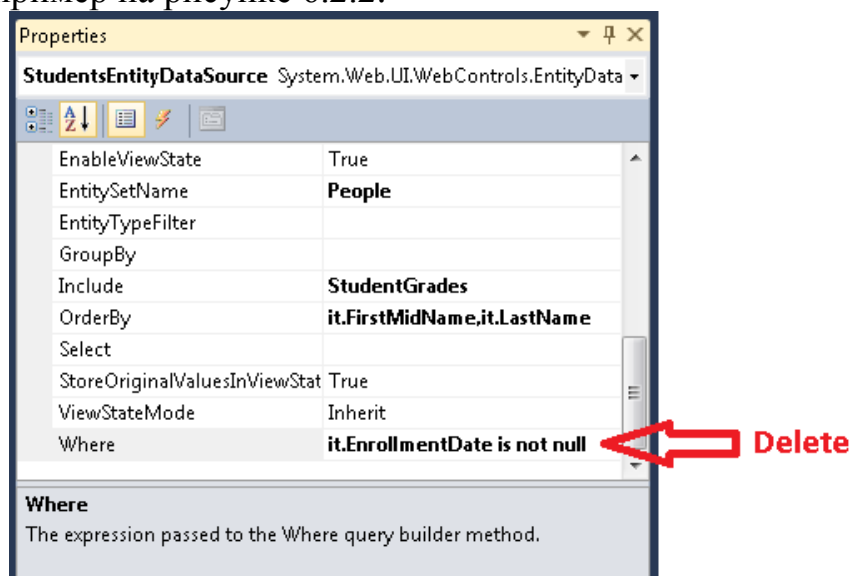


Рисунок 6.2.2 Удаление значения фильтрации

Далее необходимые изменения вы выполните в коде разметки.

1. Откройте страницу *Students.aspx* в режиме разметки.

2. В элементе `StudentsEntityDataSource` удалите атрибут `Where` и добавьте атрибут `EntityTypeFilter="Student"`. Код разметки должен быть следующим:

```
<asp:EntityDataSource ID="StudentsEntityDataSource"
runat="server"
 ConnectionString="name=SchoolEntities"
 DefaultContainerName="SchoolEntities"
 EnableDelete="True" EnableFlattening="False"
 EnableUpdate="True"
 EntitySetName="People" Include="StudentGrades"
 OrderBy="it.LastName" EntityTypeFilter="Student">
</asp:EntityDataSource>
```

Установка атрибута `EntityTypeFilter` гарантирует, что `EntityDataSource` будет выбирать только указанный тип. Если вы хотите, чтобы получить оба типа `Student` и `Instructor`, вы должны не устанавливать этот атрибут.

3. Для элемента `SearchEntityDataSource` добавьте атрибут `EntityTypeFilter="Student"` и измените атрибут `Where` выбирает студентов. Код разметки должен быть следующим:

```
<asp:EntityDataSource ID="SearchEntityDataSource"
runat="server"
 ConnectionString="name=SchoolEntities"
 DefaultContainerName="SchoolEntities" EnableFlattening="False"
 EntitySetName="People" EntityTypeFilter="Student"
 Where=" it.FirstMidName Like '%' + @StudentName + '%' or
it.LastName Like '%' + @StudentName + '%" >
 <WhereParameters>
 <asp:ControlParameter ControlID="SearchTextBox"
Name="StudentName" PropertyName="Text"
Type="String" DefaultValue="%" />
 </WhereParameters>
</asp:EntityDataSource>
```

4. Запустите страницу, чтобы проверить, что все работает как раньше.  
5. Обновите следующие страницы, которые вы создали в предыдущих упражнениях, так чтобы они использовали новые сущности `Student` и `Instructor` вместо сущности `Person`, а затем запустите их, чтобы убедиться, что они работают как раньше:

- на странице `StudentsAdd.aspx` в элемент `StudentsEntityDataSource` добавьте `EntityTypeFilter="Student"`:

```
<asp:EntityDataSource ID="StudentsEntityDataSource" runat="server"
 ConnectionString="name=SchoolEntities"
 DefaultContainerName="SchoolEntities"
 EnableDelete="True" EnableFlattening="False"
 EnableUpdate="True"
 EnableInsert="True" EntityTypeFilter="Student"
 EntitySetName="People" Include="StudentGrades">
</asp:EntityDataSource>
</asp:EntityDataSource>
```



- на странице *About.aspx* в элемент `StudentStatisticsEntityDataSource` добавьте `EntityTypeFilter="Student"` и удалите `where="it.EnrollmentDate is not null"`. Код разметки должен быть таким:

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

- на страницах *Instructors.aspx* и *InstructorsCourses.aspx* в элемент `InstructorsEntityDataSource` добавьте `EntityTypeFilter="Instructor"` и удалите `Where="it.HireDate is not null"`.

Код страницы *Instructors.aspx*:

```
<asp:EntityDataSource ID="InstructorsEntityDataSource"
runat="server"
 ConnectionString="name=SchoolEntities"
DefaultContainerName="SchoolEntities" EnableFlattening="False"
 EntitySetName="People" EntityTypeFilter="Instructor"
Include="OfficeAssignment">
</asp:EntityDataSource>
```

Код разметки *InstructorsCourses.aspx*:

```
<asp:EntityDataSource ID="InstructorsEntityDataSource"
runat="server"
 ConnectionString="name=SchoolEntities"
DefaultContainerName="SchoolEntities"
 EnableFlattening="False"
 EntitySetName="People"
 EntityTypeFilter="Instructor"
 Select="it.LastName + ', ' + it.FirstMidName AS Name,
it.PersonID">
</asp:EntityDataSource>
```

В результате этих изменений, вы улучшили стиль приложения Contoso в несколькими способами.

Вы перенесли выборку данных и логику проверки из слоя интерфейса (.aspx разметки) и сделали его неотъемлемой частью слоя доступа к данным. Это помогает изолировать код приложения от изменений, которые вы могли бы сделать в будущем в схеме базы данных или модели данных. Например, вы могли бы решить, что студенты могут быть наняты в качестве помощников преподавателей и, следовательно, получить дату найма. Затем можно добавить новое свойство, определяющее студентов-инструкторов, и

обновить модель данных. Код веб-приложения не нужно будет изменять исключением случаев, когда вы хотите, чтобы показать дату для студентов. Еще одно преимущество добавления классов инструкторов и студентов состоит в том, что код становится более понятным, чем когда использовались только объекты `Person`, которые на самом деле студенты или преподаватели.

## **Лабораторная работа 7. Использование хранимых процедур**

В этой работе вы узнаете, как использовать хранимые процедуры, чтобы получить больше контроля над доступом к базе данных.

Платформа `Entity Framework` поддерживает возможность использования хранимых процедур для доступа к базе данных. Для любой сущности вы можете указать хранимую процедуру, чтобы использовать ее для создания, обновления или удаления объектов этого типа. Тогда в модели данных вы можете добавить ссылки на хранимые процедуры, которые можно использовать для выполнения таких задач, как получение наборов данных.

Использование хранимых процедур является общим требованием для доступа к базе данных. В некоторых случаях администратор базы данных может потребовать, чтобы весь доступ к базе данных проходил через хранимые процедуры в целях безопасности. В других случаях вы можете построить бизнес-логику в некоторых процессах, которые `Entity Framework` использует при обновлении базы данных. Например, всякий раз, когда объект удаляется, возможно, вы хотите скопировать его в архивной базе данных. Или, когда ряд обновляется, вы можете написать строку таблицы регистрации, которая записывает, кто сделал изменения. Вы можете выполнить эти виды задач в хранимой процедуре, которая вызывается, когда `Entity Framework` удаляет или обновляет объект.

В этой работе вы измените способ обращения к базе данных для некоторых уже созданных страниц. Вы создадите хранимые процедуры в базе данных для вставки объектов `Student` и `Instructor`. Вы добавите их в модели данных, укажите, что `Entity Framework` должен использовать их для добавления объектов `Student` и `Instructor` в базу данных. Вы также создадите хранимую процедуру, которую можно использовать для извлечения объектов `Course`.

### **Упражнение 1. Создание хранимых процедур в базе данных**

1. В **Server Explorer** (Обозревателе серверов) раскройте узел *School.mdf*, далее раскройте узел **Stored Procedures (Хранимые процедуры)** и просмотрите список уже существующих процедур в базе данных *School*.

2. Выделите папку **Stored Procedures (Хранимые процедуры)** и в контекстном меню выберите команду **Add New Stored Procedure (Добавить новую хранимую процедуру)**.
3. Скопируйте следующие SQL-выражения и вставьте в окно редактирования хранимой процедуры, заменив шаблон:

```
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 имеет четыре свойства: PersonID, LastName, FirstName и EnrollmentDate. В базе данных генерируется значение идентификатора автоматически, и хранимая процедура принимает параметры для трех других свойств. Хранимая процедура возвращает значение ключа записи новой строки так, чтобы платформа Entity Framework могла отслеживать версию объекта в памяти.

4. Сохраните и закройте хранимую процедуру.
5. Создайте хранимую процедуру InsertInstructor, используя следующее SQL выражение:

```
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;
```

6. Создайте две хранимые процедуры обновления объектов Student и Instructor. (база данных имеет хранимую процедуру удаления DeletePerson, которая может применяться и для Instructor и для Student).

```
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;

```

7. Создайте хранимую процедуру, которая будет считывать данные о курсах:

```

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

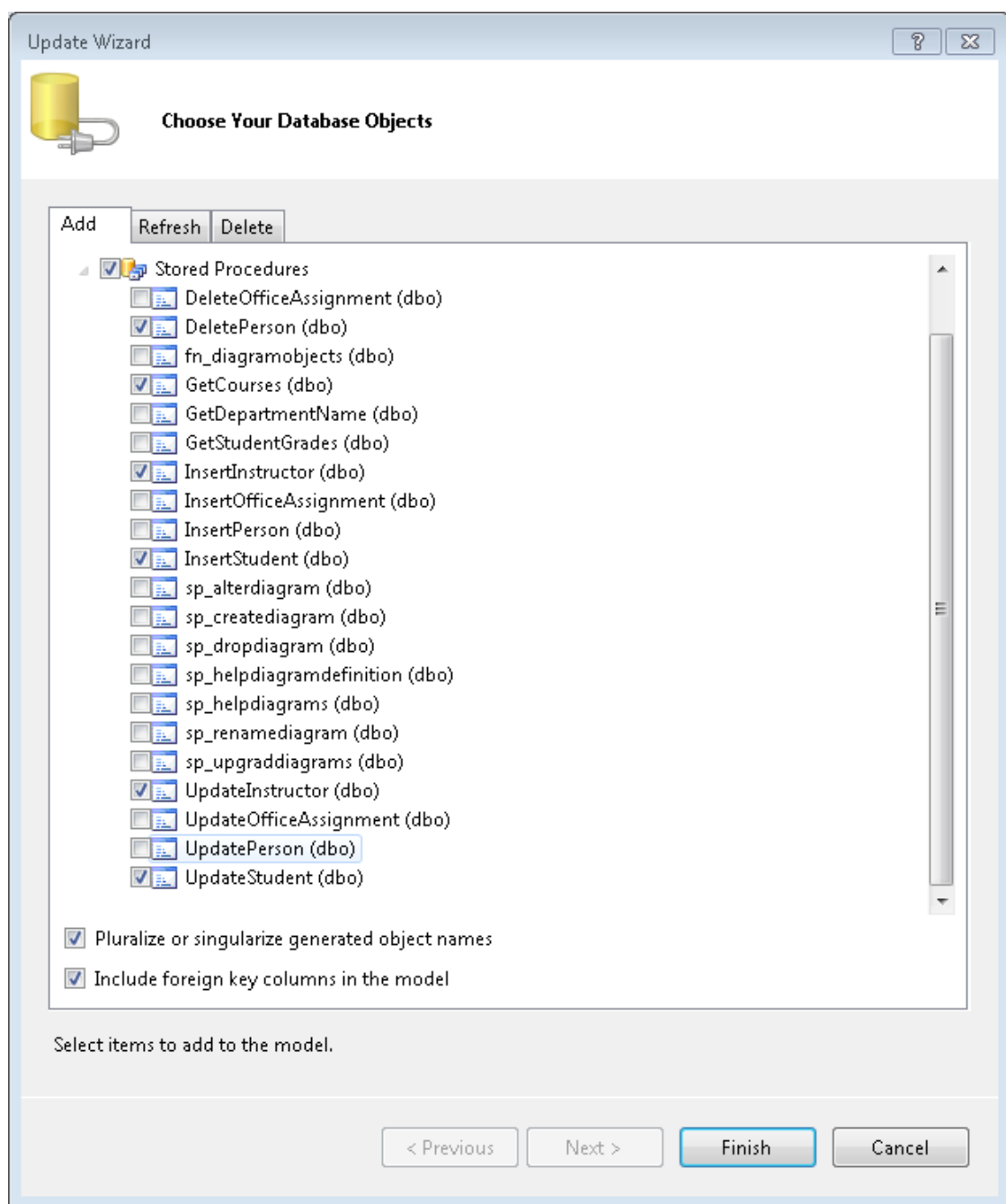
```

## ***Упражнение 2. Размещение хранимых процедур в модели данных***

### **Добавление хранимых процедур в модель данных**

Хранимые процедуры, которые определены в базе данных, должны быть добавлены к модели данных, чтобы сделать их доступными для Entity Framework.

1. Откройте модель *SchoolModel.edmx* в режиме дизайнера.
2. В контекстном меню поверхности модели выберите **Update Model from Database (Обновить модель из базы данных)**.
3. На вкладке **Add (Добавить)** окна **Choose Your Database Objects (Выбор объектов базы данных)** (см. рисунок 7.2.1) раскройте **Stored Procedures (Хранимые процедуры)**, выберите вновь добавленные процедуры и процедуру `DeletePerson`.
4. Включите оба флажка генерации имен и включения внешних ключей, нажмите **Finish (Готово)**.



*Рисунок 7.2.1 Добавление процедур*

### Сопоставление хранимых процедур

5. В дизайнере модели в контекстном меню `Student` выберите **Stored Procedure Mapping (Сопоставление хранимых процедур)** (см. рисунок 7.2.2).

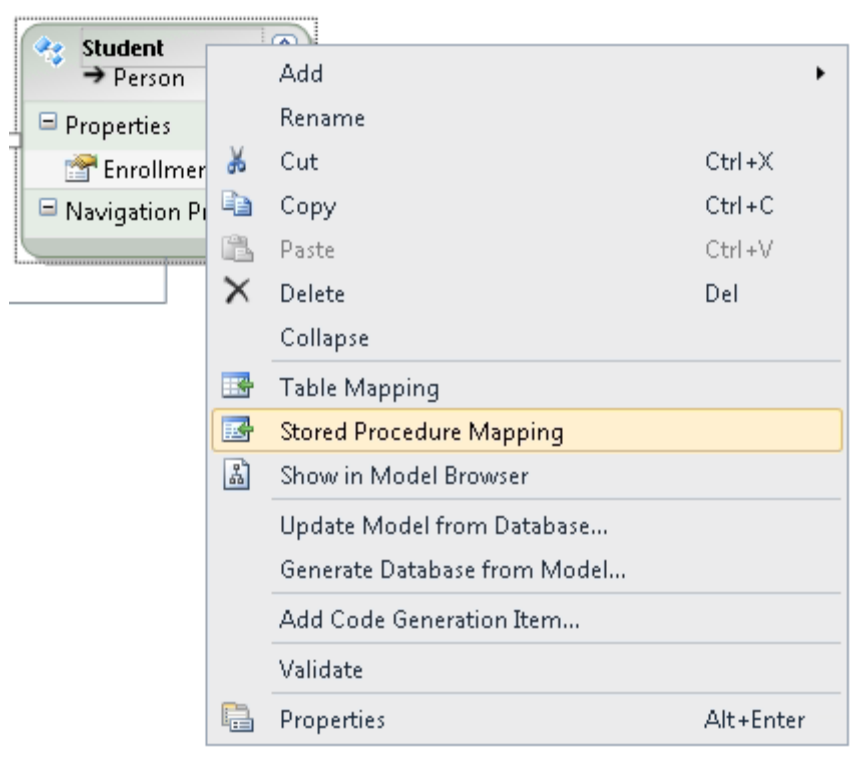


Рисунок 7.2.2 Открытие окна сопоставления процедур

Появится окно **Mapping Details (Сведения о сопоставлении)** (см. рисунок 7.2.3), в котором можно указать хранимые процедуры, которые Entity Framework должен использовать для вставки, обновления и удаления объектов данного типа.

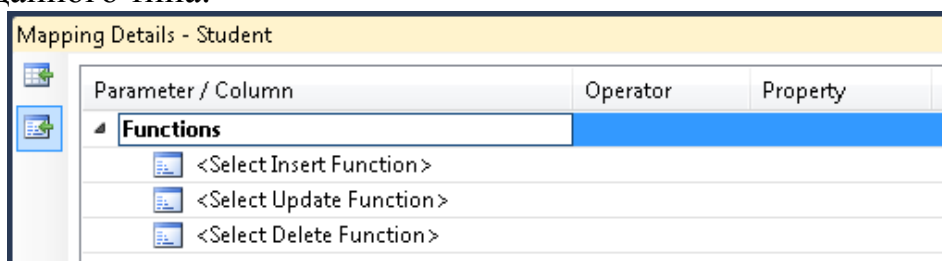


Рисунок 7.2.3 Окно сопоставления процедур

6. Выберите в списке **<Select Insert Function> (Вставка)** функцию **InsertStudent**.

Окно, показанное на рисунке 7.2.4 отображает список параметров хранимой процедуры, каждый из которых должен быть сопоставлен со свойствами сущности. Обратите внимание, что `LastName` и `EnrollmentDate` отображаются автоматически, потому что имена совпадают, а вот свойства `FirstName` нет (потому, что в вы изменили имя свойства `FirstName` на `FirstMidName`).

7. Выберите `FirstMidName:String` из выпадающего списка напротив свойства `FirstName`, который показывает доступные свойства сущностей.

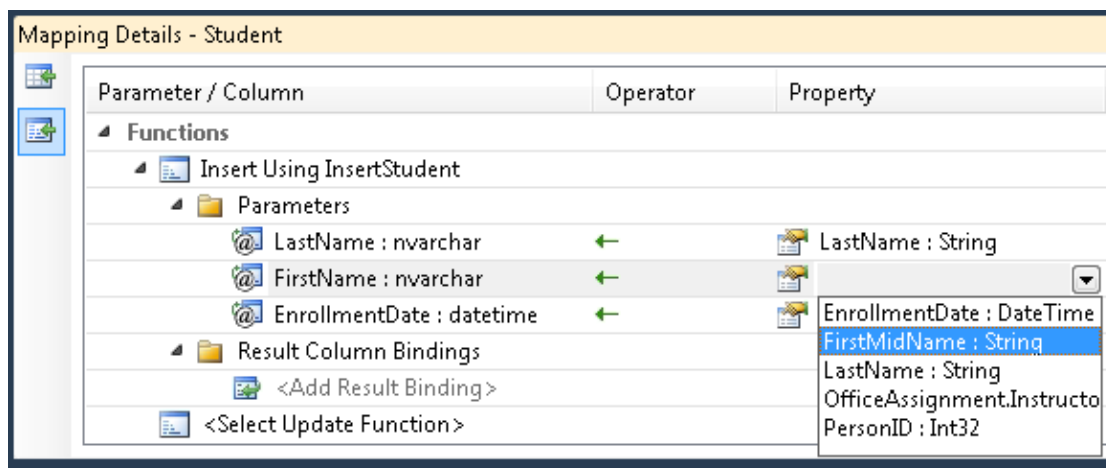


Рисунок 7.2.4 Настройка сопоставления

8. Ниже в списке выбора функции обновления укажите хранимую процедуру UpdateStudent.
9. И для функции удаления укажите процедуру DeletePerson.
10. Проверьте параметры в соответствии с рисунком 7.2.5.

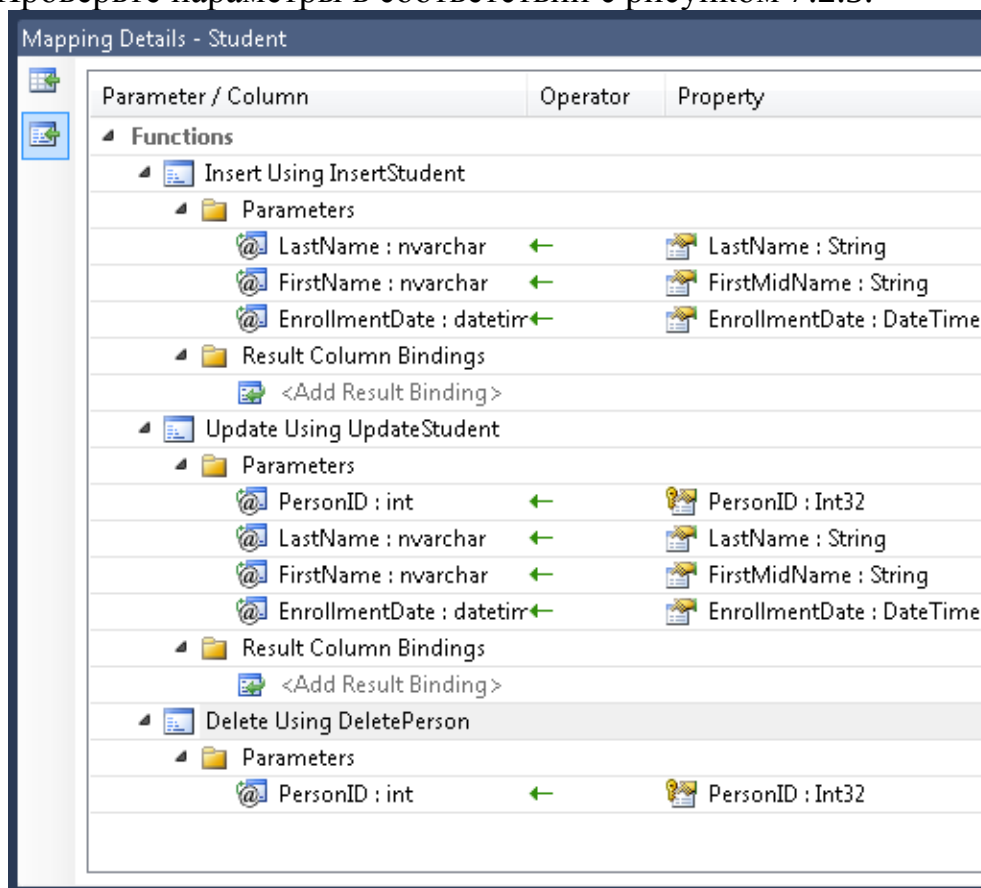


Рисунок 7.2.5 Параметры сопоставления сущности Student

11. В контекстном меню Instructor выберите **Stored Procedure Mapping (Сопоставление хранимых процедур)** и укажите соответствующие хранимые процедуры вставки, обновления и удаления для сопоставления с моделью, согласно рисунка 7.2.6:

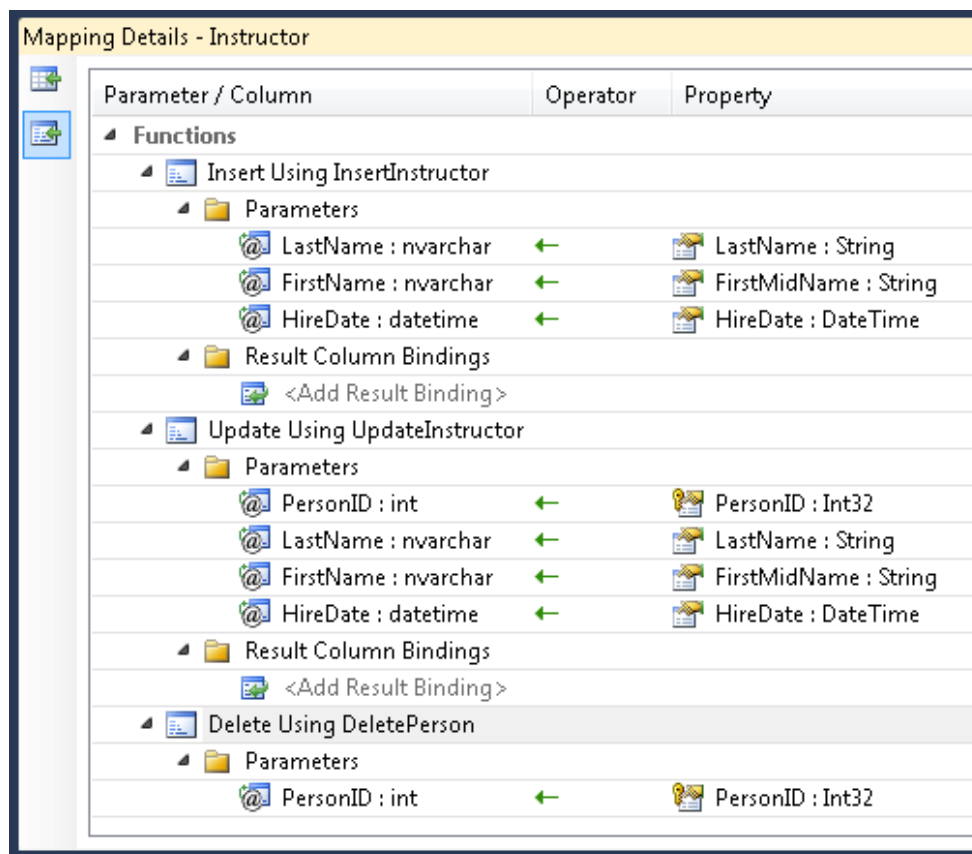


Рисунок 7.2.6 Параметры сопоставления сущности Instructor

12. Откройте **Model Browser** (Обозреватель моделей), раскройте последовательно узлы **SchoolModel.Store** и **Stored Procedures** (Хранимые процедуры).
13. В контекстном меню процедуры **GetCourses** выберите команду **Add Function Import** (Добавить импорт функций), как показано на рисунке 7.2.7:

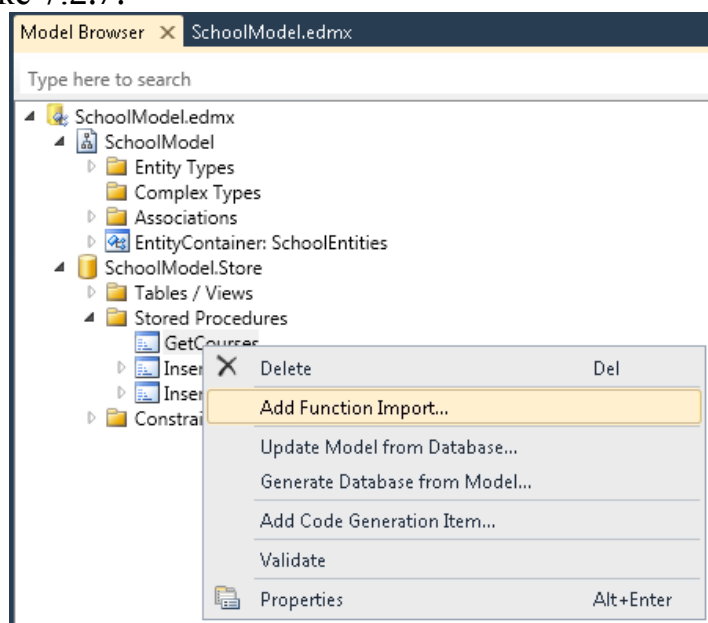


Рисунок 7.2.7 Меню добавления функций



14. В окне добавления импорта **Add Function Import** (см. рисунок 7.2.8) для группы **Returns a Collection Of** (Возвращает коллекцию) выберите переключатель **Entities** (Сущности), и затем выберите **Course** в качестве типа возвращаемой сущности. Нажмите **ОК**, сохраните и закройте файл модели с расширением **edmx**.

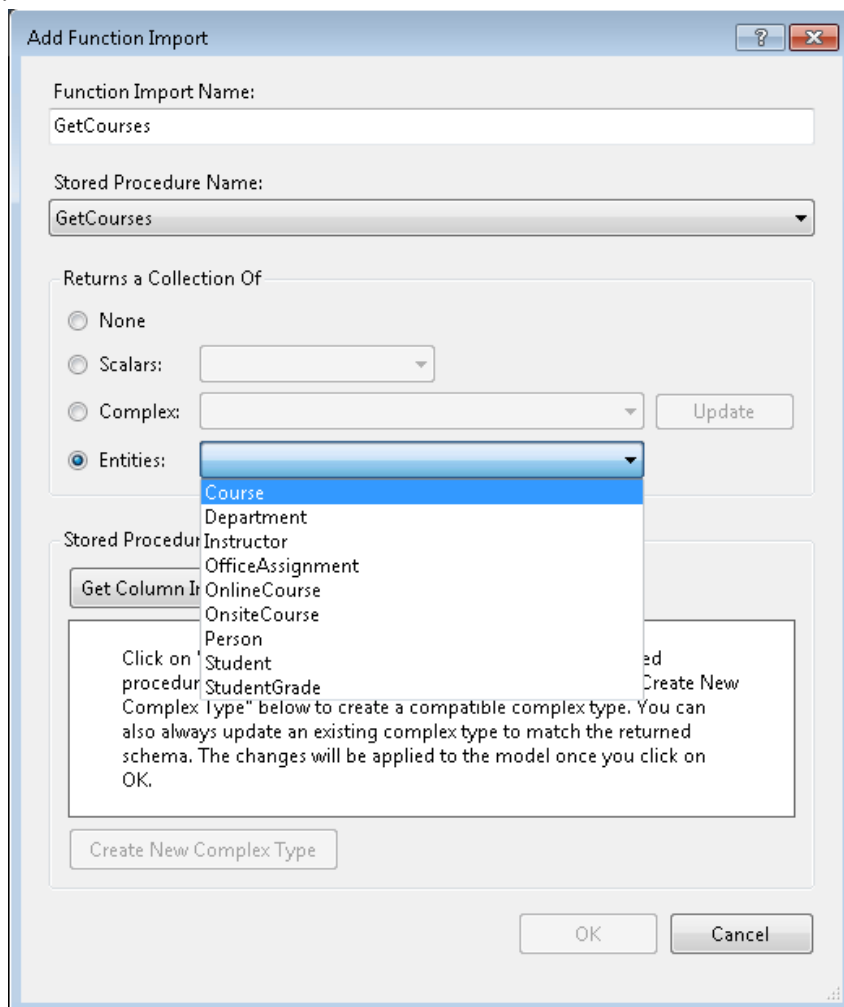


Рисунок 7.2.8 Импорт функции

15. Постройте приложение.

### Упражнение 3. Применение хранимых процедур

#### Применение процедур Insert, Update, и Delete

1. Откройте страницу *StudentsAdd.aspx* в обозревателе и добавьте нового студента, запустится хранимая процедура *InsertStudent* и запись добавится в таблицу *Student*.
2. Откройте страницу *Students.aspx* в обозревателе и проверьте, что новый студент добавлен в список.
3. Измените имя для проверки функции обновления, а затем удалите студента для проверки функции удаления.

## Применение хранимой процедуры Select

Entity Framework не может автоматически запустить такую хранимую процедуру как `GetCourses`, и в этом случае нет возможности использовать элемент `EntityDataSource`. Запуск процедуры вы должны выполнить непосредственно в коде.

4. Откройте файл *InstructorsCourses.aspx.cs*.

5. Найдите метод `PopulateDropDownLists`, который применяет запрос LINQ-to-Entities для получения всех курсов:

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

6. Замените этот код следующим:

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

Теперь страница использует хранимую процедуру `GetCourses` для получения списка курсов.

7. Откройте данную страницу в обозревателе и проверьте правильность ее работы.

## Лабораторная работа 8. Применение функциональности Dynamic Data для форматирования и валидации данных

В этой работе вы изучите возможности динамических данных (Dynamic Data) для обеспечения следующих преимуществ:

- поля автоматически форматируются при отображении согласно типу,
- поля автоматически проверяются на соответствие типа,
- вы можете добавить в модель данных дополнительные возможности по форматированию и проверке данных.

В этой работе вы внесете изменения в элементы управления для отображения и редактирования полей на странице *Students.aspx*, и добавите форматирование и проверку полей имени и даты объектов `Student`.

### Упражнение 1. Применение элементов *DynamicField* и *DynamicControl*

1. Откройте страницу *Students.aspx* и в элементе `StudentsGridView` замените элементы `TemplateField` с именами **Name** и **Enrollment Date** следующим кодом:

```
<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" />

```

Этот код использует элементы `DynamicControl` в элементах управления `TextBox` и `Label` в шаблоне поля отображения имени студента, а также применяет элемент `DynamicField` для поля даты (`enrollment date`).

- Добавьте элемент `ValidationSummary` после компонента `StudentsGridView`:

```

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

```

- В элементе `SearchGridView` замените код столбцов **Name** and **Enrollment Date** как это было сделано в элементе `StudentsGridView`. Элементы `Columns` компонента `SearchGridView` должны выглядеть следующим образом:

```

 <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" />

```

- Откройте файл `Students.aspx.cs` и добавьте выражение `using`:  
`using ContosoUniversity.DAL;`

- Добавьте обработчик события `Init` этой страницы:

```

protected void Page_Init(object sender, EventArgs e)
{

```

```

 StudentsGridView.EnableDynamicData(typeof(Student));

```

```

 SearchGridView.EnableDynamicData(typeof(Student));
}

```

Это код определяет, как `Dynamic Data` будут обеспечивать форматирование и проверку полей элементов объекта `Student`.

- Откройте страницу в обозревателе.

- В столбце **Enrollment Date** время отображается в длинном формате в соответствии со свойством `DateTime`.

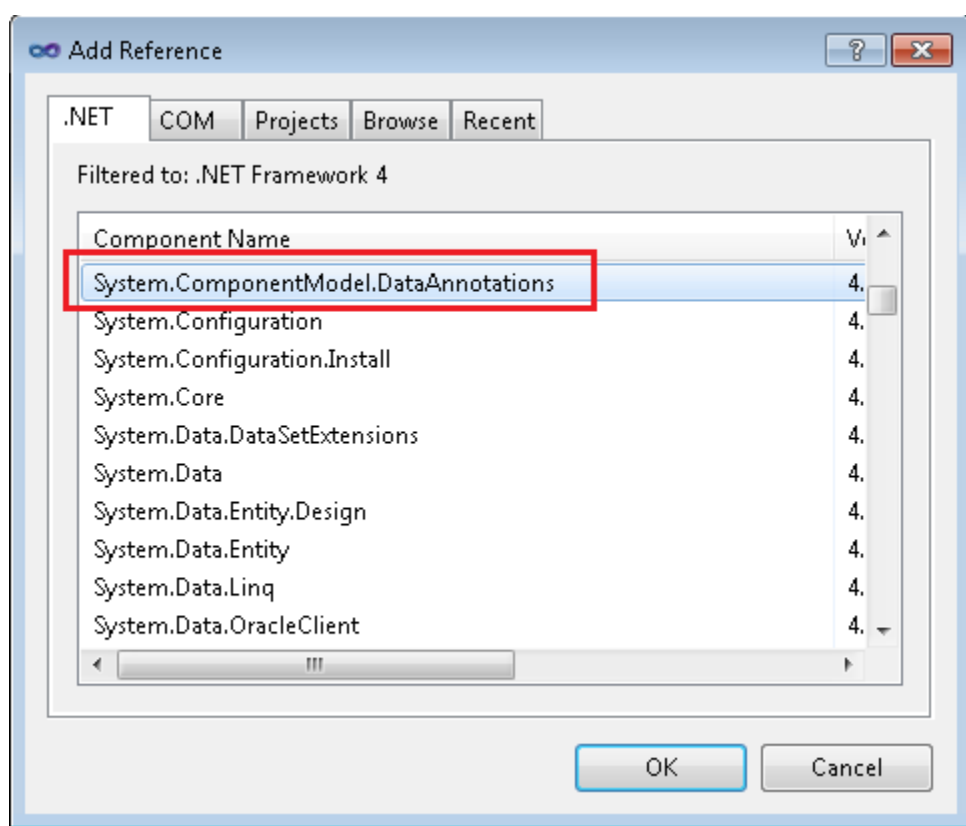
Обратите внимание как Dynamic Data автоматически обеспечивает валидацию данных.

8. Например, кликните **Edit (Правка)** для первого студента, очистите поле даты и кликните **Update (Обновить)**, проверьте, что Dynamic Data автоматически пометил, что требуется данные в это поле, так как модель не допускает, чтобы это поле было пустым. На странице должно отобразиться сообщение об этом в элементе `ValidationSummary`.
9. Проверьте наличие всплывающей подсказки при наведении мыши на метку ошибки поля даты.
10. Введите неправильную дату, например 1/32/2010, проверьте, что Dynamic Data также среагировал на эту ошибку.

## ***Упражнение 2. Добавление правил валидации и форматирования***

В этом упражнении вы реализуете дополнительные возможности по форматированию и проверки данных к тем, которые предоставляются механизмом Dynamic Data.

1. В контекстном **ContosoUniversity** выберите команду **Add Reference (Добавить ссылку)** и в окне добавления ссылки на вкладке **.NET** (см. рис. 8.2.1) найдите компонент `System.ComponentModel.DataAnnotations` и нажмите **OK**.



*Рисунок 8.2.1 Добавление ссылки на компонент*

2. В папку *DAL* добавьте новый файл – класс *Student.cs*

### 3. Добавьте класс метаданных с указанием требуемых свойств класса и атрибутов валидации:

```
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; }
 }
}
```

В этом коде создается частичный (partial) класс для сущности Student. Атрибут `MetadataType`, добавленный к этому частичному классу определяет класс как с возможностью применять специфические метаданные. Класс метаданных может иметь любое имя, но лучше применять суффикс "Metadata" как часть имени.

Атрибуты применяются для свойств в классе метаданных, определяющих форматирование, валидацию, правила и сообщения об ошибках

Атрибуты обеспечивают следующие результаты:

- `EnrollmentDate` будет отображаться как дата без времени,
  - оба поля имени должны быть не больше 25 символов и сообщение об ошибке,
  - оба поля имени требуют обязательного ввода и отображают сообщение об соответствующей ошибке.
4. Откройте страницу *Students.aspx* в обозревателе и проверьте, что на ней отображаются студенты.
  5. Выполните правку строки: удалите содержимое поля имени и обновите строку. Проверьте, что отобразились предупреждения об ошибке.
  6. Введите в поле имени более 25 символов, проверьте появление сообщения об ошибке.

## Список литературы

1. Джеффри Рихтер CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. – СПб.: Питер, 2016. – 896 с.
2. Джон Скит C# для профессионалов. Тонкости программирования. – М.:Вильямс, 2014. – 408 с.
3. Фленов М. Библия C#/3-е издание. – СПб.: БХВ-Петербург, 2016. – 544 с.
4. Троелсен Э., Джепикс Ф. Язык программирования C# 6.0 и платформа .NET 4.6. – М.:Вильямс, 2016. – 1440 с.
5. Джозеф Албахари, Бен Албахари C# 6.0. Справочник. Полное описание языка– М.:Вильямс, 2016. – 1040 с.
6. Фримен А. ASP.NET 4.5 с примерами на C# 5.0 для профессионалов. – М.:Вильямс, 2014. – 1120 с.

**Миссия университета** – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

---

## КАФЕДРА ПРОГРАММНЫХ СИСТЕМ

<http://ps.ifmo.ru>

**Кафедра Программных систем** входит в состав факультета Инфокоммуникационных технологий. На кафедре обеспечена возможность обучения студентов по современным образовательным стандартам в области программного обеспечения ИКТ. Для этого на кафедре работает высококвалифицированный преподавательский состав, имеется современная техническая база и специализированные лаборатории, оснащенные необходимым оборудованием и программным обеспечением; качественная методическая поддержка образовательных программ. Наши студенты принимают активное участие в российских и зарубежных исследованиях и конференциях, имеют возможность публикации результатов своих исследований в ведущих российских и зарубежных реферируемых изданиях. Существенным преимуществом является возможность выпускников продолжить научную деятельность в аспирантуре Университета ИТМО и в других передовых российских и зарубежных научных Центрах.

Кафедра обеспечивает подготовку бакалавров и магистров по образовательным программам:

- Интеллектуальные инфокоммуникационные системы - бакалавры;
- Программное обеспечение в инфокоммуникациях – магистры.

На кафедре реализуется международная образовательная программа DD Master Program, в рамках которой выпускники имеют возможность получить два диплома: Диплом Университета ИТМО с присвоением магистерской степени по направлению «Программное обеспечение в инфокоммуникациях» и Международный диплом - Master of Science in

Technology Lappeenranta University of Technology in the field of Computer Science majoring in Software Engineering.

**Выпускники** кафедры обладают компетенциями:

- проектирования и создания рациональных структур ИКС;
- разработки алгоритмов решения задач и их программной реализации на основе современных платформ;
- моделирования процессов функционирования сложных систем;
  - обеспечения безопасности работы ИКС;
- реализации сетевых услуг и сервисов в ИКС;
- проектирования и разработки баз данных;
- разработки клиент-серверных приложений ИКС;
- проектирования, создания и поддержки Web-приложений;
- управления проектами перспективных направлений развития ИКС.

Трудоустройство выпускников кафедры возможно на предприятиях: ООО «Digital Design»; ООО «Аркадия»; ОАО «Ростелеком»; ООО «ЭПАМ Системз»; ООО «Т-Системс СиАйЭс» и многие другие.

**Мы готовим квалифицированных инженеров в области инфокоммуникационных технологий с новыми знаниями, образом мышления и способностями быстрой адаптации к современным условиям труда.**



Осипов Никита Алексеевич

**Разработка приложений ASP.NET  
с применением Entity Framework**

**УЧЕБНОЕ ПОСОБИЕ**

В авторской редакции

Редакционно-издательский отдел Университета ИТМО

Зав. РИО

Н.Ф. Гусарова

Лицензия ИД № 00408 от 05.11.99

Подписано к печати

Заказ №

Тираж

Отпечатано на ризографе

**Редакционно-издательский отдел**  
**Университета ИТМО**  
197101, Санкт-Петербург, Кронверкский пр., 49