



Seri Belajar  
**ASP.NET**  
Implementasi Entity Framework pada ASP.NET MVC  
M Reza Faisal, Microsoft MVP ASP.NET/IIS

# Implementasi Entity Framework pada ASP.NET MVC

Tom Dykstra

Diterjemahkan oleh: M Reza Faisal

**Ringkasan Isi:** Contoh aplikasi web Universitas Contoso memberikan demonstrasi bagaimana membuat aplikasi web ASP.NET MVC dengan menggunakan Entity Framework. Ebook ini akan menerangkan langkah-langkah yang dilakukan untuk membangun seperti contoh web aplikasi Universtas Contoso.

**Kategori:** Seri Belajar

**Teknologi:** ASP.NET

**Source:** ASP.NET site (<http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc>)

**Tanggal Publikasi:** Februari 2013



Original © 2012 Microsoft; Translation ©2013 M Reza Faisal. All rights reserved. This translation has not been reviewed for accuracy.

## Contents

Pendahuluan .....	7
Membuat Entity Framework Data Model untuk Aplikasi ASP.NET MVC .....	9
Aplikasi Web Universitas Contoso .....	9
Pendekatan Pengembangan dengan Entity Framework .....	12
Database First .....	13
Model First .....	13
Code First .....	14
POCO (Plain Old CLR Object).....	14
Membuat Aplikasi Web MVC.....	14
Style Website .....	16
Membuat Data Model.....	19
Student Entity .....	21
Course Entity.....	22
Membuat Database Context.....	23
Membuat Connection String.....	24
Inisialisasi Database .....	24
Membuat Student Controller.....	28
Convention .....	35
Implementasi Fungsi CRUD dengan menggunakan Entity Framework pada Aplikasi ASP.NET MVC.....	37
Membuat Halaman Detail.....	40
Membuat Halaman Create.....	43
Membuat Halaman Edit .....	49
Entity State dan Method Attach dan SaveChanges .....	50
Membuat Halaman Untuk Delete Data .....	53
Memastikan Koneksi Database Tidak Ditinggalkan Terbuka .....	58
Sorting, Filtering dan Pagging dengan Entity Framework pada Aplikasi ASP.NET MVC .....	59
Menambahkan Link Sorting Kolom Pada Halaman Index Student .....	60
Menambahkan Fungsionalitas Sorting pada Index Method .....	60
Menambahkan Hyperlink di Header Kolom pada Student Index View .....	62
Menambahkan Kolom Pencarian pada halaman Student Index .....	64
Menambahkan Fungsionalitas Filter pada Index Method .....	64

Menambahkan Kolom Pencarian pada Student Index View .....	66
Menambahkan Pagging pada halaman Student Index .....	67
Installasi Paket PagedList NuGet.....	68
Menambahkan Fungsionalitas Pagging pada Index Method .....	69
Menambahkan Link Pagging pada Halaman Student Index .....	72
Membuat halaman About yang berfungsi untuk menampilkan data Statistic Student .....	77
Membuat View Model .....	78
Modifikasi Home Controller.....	78
Modifikasi About View .....	79
Membuat Data Model Yang Lebih Rumit untuk Aplikasi ASP.NET MVC.....	82
Menggunakan Attribute untuk Formatting Control, Validasi dan Mapping Database.....	83
Attribute DisplayFormat .....	83
Attribute MaxLength.....	85
Attribute Column .....	88
Membuat Instructor Entity .....	90
Attribute Required dan Attribute Display.....	92
Calculated Property, FullName .....	92
Navigation Property, Courses dan OfficeAssignment.....	92
Membuat Entity OfficeAssignment.....	93
Attribute Key .....	94
Instructor Navigation Property .....	94
Modifikasi Course Entity .....	95
Attribute DatabaseGenerated .....	96
Foreign Key dan Property Navigation .....	96
Membuat Entity Department .....	97
Attribute Column .....	98
Foreign Key dan Property Navigation .....	99
Modifikasi Entity Student.....	100
Modifikasi Entity Enrollment .....	102
Foreign Key dan Property Navigation .....	103
Relasi Many-to-Many.....	103
Attribute DisplayFormat .....	106

Menampilkan Relasi pada Entity Diagram .....	107
Modifikasi Database Context .....	108
Inisialisasi Database dan Data Testing .....	110
Menghapus dan Membuat Ulang Database .....	115
Membaca Data Terkait dengan Entity Framework pada Aplikasi ASP.NET MVC .....	119
Lazy, Eager, dan Explicit Loading Data .....	121
Membuat Halaman Course Index untuk Menampilkan Nama Departement.....	123
Membuat Halaman Instructors Index untuk Menampilkan Course dan Enrollment .....	127
Membuat View Model untuk View dari Instructor Index .....	129
Penambahan Style pada Row yang dipilih .....	130
Membuat Controller dan View untuk Instructor .....	130
Modifikasi View pada Instructor Index .....	134
Menambahkan Explicit Loading .....	144
Update Data Terkait dengan Entity Framework pada Aplikasi ASP.NET MVC.....	147
Mengubah Halaman Create dan Edit untuk Course .....	150
Menambah Halaman Edit untuk Instructor .....	159
Menambah Course Assignment pada Halaman Edit Instructor.....	166
Penanganan Concurrency dengan Entity Framework pada Aplikasi ASP.NET MVC.....	178
Konflik Concurrency .....	180
Pessimistic Concurrency (Locking) .....	180
Optimistic Concurrency.....	180
Mendeteksi Konflik Concurrency .....	184
Menambahkan Property Tracking pada Entity Department.....	185
Membuat Controller Department.....	185
Testing Penanganan Optimistic Concurrency .....	191
Menambahkan Halaman Delete .....	197
Implementasi Inheritance dengan Entity Framework pada Aplikasi ASP.NET MVC.....	208
Table-per-Hierarchy vs Table-per-Type Inheritance .....	208
Membuat Class Person .....	210
Menambah Tipe Entity Person pada Model .....	213
Changing InstructorID and StudentID menjadi PersonID .....	213
Menyesuaikan Nilai Primary Key pada Initializer.....	214

Mengubah OfficeAssingment menjadi Lazy Loading .....	215
Testing.....	215
Implementasi Pattern Repository dan Unit of Work pada Aplikasi ASP.NET MVC.....	217
Pattern Repository dan Unit of Work .....	217
Membuat Class Student Repostory .....	219
Mengubah Student Controller untuk Menggunakan Repository .....	223
Implementasi Class Generic Repository dan Unit of Work.....	233
Membuat Generic Repository.....	234
Membuat Class Unit of Work.....	240
Mengubah Course Controller untuk Menggunakan Class UnitOfWork dan Repositories.....	243
Kasus-Kasus Lanjutan Entity Framework untuk Aplikasi Web MVC .....	251
Melakukan query dengan Raw SQL. ....	253
Memanggil Query yang Mengembalikan Entity .....	253
Memanggil Query yang Mengembalikan Tipe Object .....	256
Memanggil Query untuk Update .....	257
Melakukan query no-tracking.....	264
Memeriksa query yang dikirim ke database.....	269
Bekerja dengan class-class proxy.....	273
Menonaktifkan deteksi otomatis perubahan. ....	274
Menonaktifkan validasi saat menyimpan perubahan.....	274
Materi-Materi Entity Framework.....	274

# Pendahuluan

---

Contoh aplikasi web Universitas Contoso memberikan demonstrasi bagaimana membuat aplikasi web ASP.NET MVC dengan menggunakan Entity Framework. Contoh aplikasi ini adalah sebuah website untuk Universitas Contoso yang keberadaannya hanyalah fiksi belaka. Pada aplikasi web ini terdapat fungsi-fungsi pengelolaan pelayar, pengelolaan mata kuliah dan pengelolaan penugasan pengajar.

Seri tutorial ini akan menerangkan langkah-langkah yang dilakukan untuk membangun seperti contoh web aplikasi Universtas Contoso. Anda dapat mengunduh source code dari contoh aplikasi web ini di <http://go.microsoft.com/fwlink/?LinkId=215669> atau dapat juga mengikuti tutorial ini untuk membuat sendiri sample aplikasi web Universitas Contonso secara utuh. Pada tutorial ini akan dicontohkan pembangunan dengan menggunakan bahasa C#. pada source code pada link di atas dapat ditemui contoh aplikasi web dalam bahasa C# dan Visual Basic. Jika masih terdapat pertanyaan yang mungkin tidak berhubungan langsung dengan tutorial yang telah diberikan maka pertanyaan dapat disampaikan forum ASP.NET Entity Framework di <http://forums.asp.net/1227.aspx> atau pada forum Entity Framework and LINQ to Entities di <http://social.msdn.microsoft.com/forums/en-US/adodonetentityframework/threads/>.

Tutorial ini membutuhkan pengetahuan dasar bagaimana bekerja dengan ASP.NET MVC pada Visual Studio. Bagi pembaca yang belum mempunyai pengetahuan tersebut maka Anda dapat mendapatkan pengetahuan dasar dari tutorial ASP.NET MVC di <http://www.asp.net/mvc/tutorials/getting-started-with-MVC3-part1-cs>. Selain itu jika Anda ingin membuat aplikasi ASP.NET Web Form maka tersedia tutorial Getting Started with the Entity Framework di <http://www.asp.net/entity-framework/tutorials#Getting Started> dan tutorial Continuing with the Entity Framework di <http://www.asp.net/entity-framework/tutorials#Continuing>.

Sebelum mengikuti tutorial ini, perlu disiapkan instalasi software-software berikut ini pada komputer :

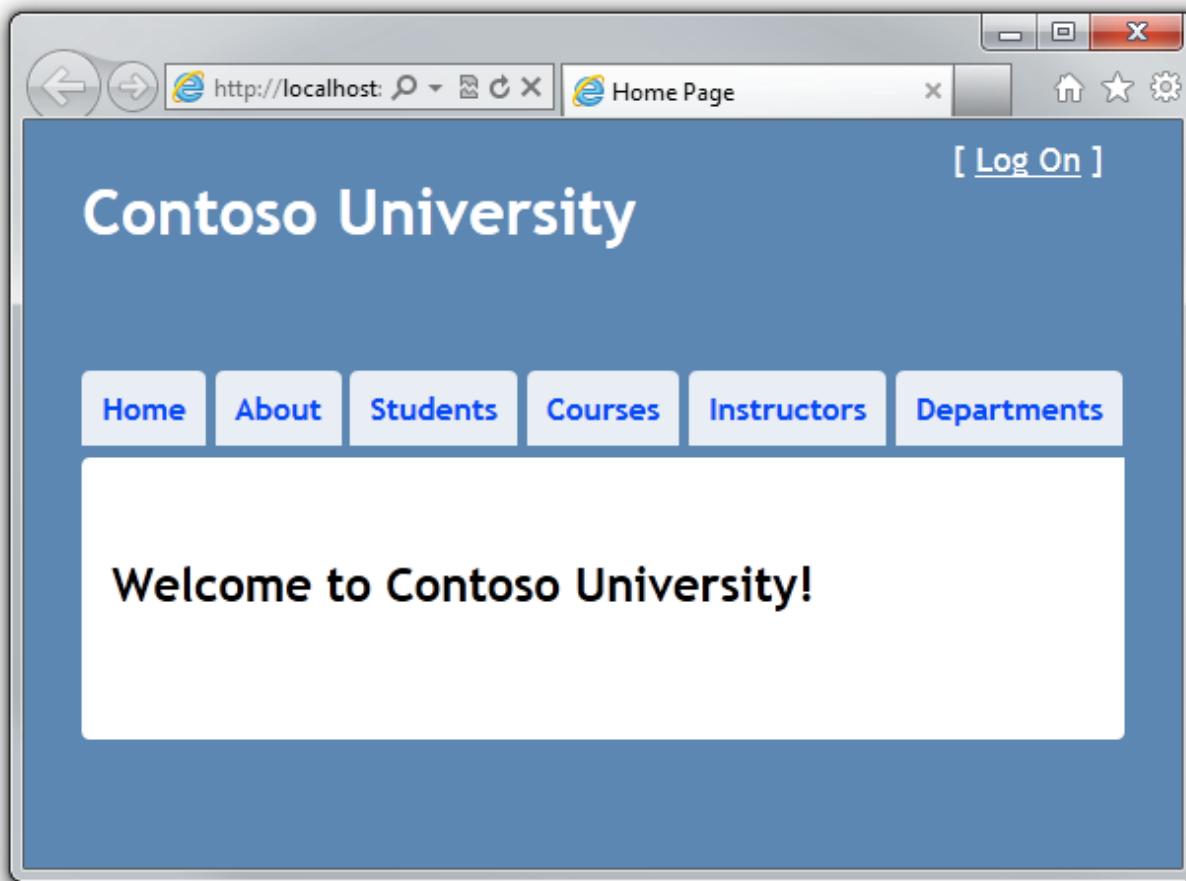
1. Visual Studio 2010 SP1  
(<http://www.microsoft.com/web/gallery/install.aspx?appxml=&appid=VS2010SP1Pack>) atau Visual Web Developer Express 2010 SP1  
(<http://www.microsoft.com/web/gallery/install.aspx?appxml=&appid=VWD2010SP1Pack>).
2. ASP.NET MVC 3 Tool Update  
(<http://www.microsoft.com/web/gallery/install.aspx?appid=MVC3>).
3. Microsoft SQL Server Compact 4.0  
(<http://www.microsoft.com/web/gallery/install.aspx?appid=SQLCE>).
4. Microsoft Visual Studio 2010 SP1 Tools for SQL Server Compact 4.0  
(<http://www.microsoft.com/web/gallery/install.aspx?appid=SQLCEVSTools>).

*This page intentionally left blank*

# Membuat Entity Framework Data Model untuk Aplikasi ASP.NET MVC

## Aplikasi Web Universitas Contoso

Berikut ini adalah website universitas sederhana yang akan kita bangun.



Pengguna dapat melihat dan melakukan update informasi Student, Course and Instructors. Berikut adalah beberapa antarmuka dari fungsional tersebut.

The screenshot shows a web browser window with the following details:

- Address Bar:** http://localhost
- Title Bar:** Students
- Header:** [ Log On ]
- Main Content:** Contoso University
- Navigation Bar:** Home, About, Students, Courses, Instructors, Departments
- Section:** Students
- Link:** Create New
- Table:** A grid of student data with columns: Last Name, First Name, Enrollment Date. Each row has Edit, Details, and Delete links.

	Last Name	First Name	Enrollment Date
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alexander	Carson	9/1/2005 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alonso	Meredith	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Anand	Arturo	9/1/2003 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Barzdukas	Gytis	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Li	Yan	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Justice	Peggy	9/1/2001 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Norman	Laura	9/1/2003 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Olivetto	Nino	9/1/2005 12:00:00 AM

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title bar says "Create". The main content area displays the "Contoso University" logo and navigation links for Home, About, Students, Courses, Instructors, and Departments. Below this, a "Create" section is shown for a "Student". It contains fields for Last Name, First Name, and Enrollment Date, each with an associated input box. A "Create" button is at the bottom of the form. A link "Back to List" is located below the form. The browser status bar at the bottom shows the URL <http://localhost:35639/>.

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Create

**Student**

Last Name

First Name

Enrollment Date

**Create**

[Back to List](#)

http://localhost:35639/

The screenshot shows a web application interface for Contoso University. At the top, there is a navigation bar with links for Home, About, Students, Courses, Instructors, and Departments. The current page is 'Instructors'. A sub-navigation menu under 'Instructors' includes 'Create New', 'Last Name', 'First Name', 'Hire Date', 'Office', and 'Courses'. Below this is a table listing instructors with columns for ID, Last Name, First Name, Hire Date, Office, and Courses. The table contains five rows of data. Below the table, there is a section titled 'Courses Taught by Selected Instructor' with a table showing courses taught by selected instructors. Finally, there is a section titled 'Students Enrolled in Selected Course' with a table showing students enrolled in selected courses.

	Last Name	First Name	Hire Date	Office	Courses
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	Smith 17	1050 Chemistry
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Gowan 27	1050 Chemistry
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304	4022 Microeconomics 4041 Macroeconomics
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001		1045 Calculus 2021 Composition 2042 Literature 3141 Trigonometry
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004		

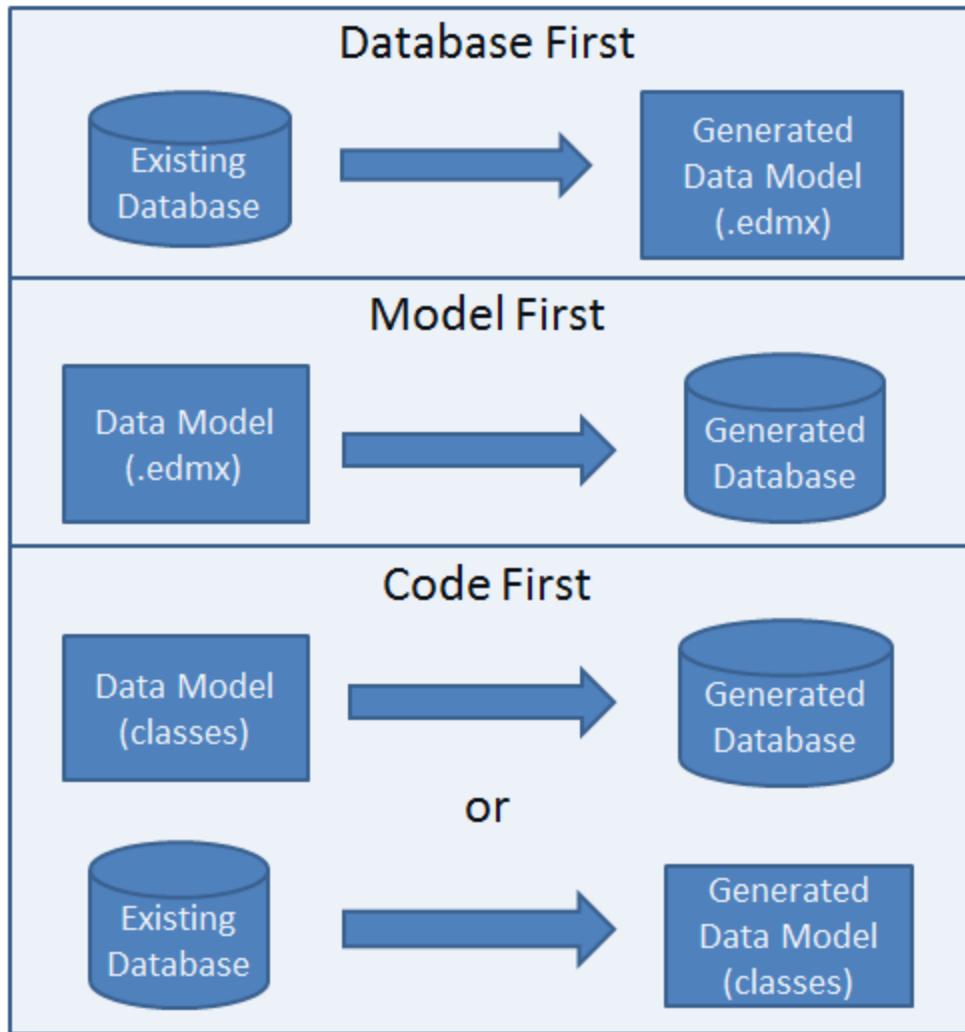
ID	Title	Department
1045	Calculus	Mathematics
2021	Composition	English
2042	Literature	English
3141	Trigonometry	Mathematics

Name	Grade
Alonso, Meredith	4.00
Norman, Laura	2.00

Antarmuka yang dilihat diatas menggunakan template built-in yang telah ada, sehingga pada tutorial ini akan lebih berkonsentrasi pada penggunaan Entity Framework.

## Pendekatan Pengembangan dengan Entity Framework

Berdasarkan diagram berikut di bawah, terdapat tiga cara yang dapat digunakan untuk bekerja dengan data pada Entity Framework yaitu Database First, Model First dan Code First.



## Database First

Jika database sudah tersedia maka Entity Framework akan secara otomatis membuat data model yang terdiri atas class-class property-property yang sesuai dengan tabel-tabel dan kolumn-kolumn pada database yang dipilih. Informasi tentang struktur database (store schema), data model (conceptual model) dan mapping akan disimpan dalam file berformat XML dan berextension .edmx. Pada bagian Getting Started with the Entity Framework di [http://www.asp.net/entity-framework/tutorials#Getting Started](http://www.asp.net/entity-framework/tutorials#GettingStarted) dan tutorial Continuing with the Entity Framework di <http://www.asp.net/entity-framework/tutorials#Continuing> untuk tutorial seri Web Form juga menggunakan pendekatan pengembangan Database First.

## Model First

Jika database belum tersedia maka pengembangan dapat dilakukan dengan membuat sebuah model dengan memanfaatkan Entity Framework designer pada Visual Studio. Setelah model selesai dibuat maka designer dapat membuat DDL (data definition language) secara otomatis yang nantinya dapat digunakan untuk membuat database. Pendekatan ini juga menggunakan file .edmx untuk menyimpan

informasi model dan mapping. Pada tutorial What's New in the Entity Framework 4 (<http://www.asp.net/entity-framework/tutorials/what-s-new-in-the-entity-framework-4>) memuat contoh pengembangan dengan Model First.

## Code First

Jika database dan model belum dibuat maka dapat ditulis kode class dan property yang sesuai dengan tabel dan kolom kemudian menggunakan pada Entity Framework tanpa bantuan file .edmx. Pendekatan seperti ini kadang ditemui, dan sering disebut sebagai pendekatan pengembangan *code only*, walaupun secara resmi nama pendekatan pengembangan Code First. Mapping yang terdapat antara store scheme dan conceptual model pada kode yang dibuat akan ditangani oleh API yang secara khusus menangani convention dan mapping. Jika database dan model belum dibuat maka Entity Framework akan membuatkan database, dan secara otomatis akan dilakukan operasi menghapus dan membuat ulang database jika ternyata ada perubahan pada model. Pada tutorial ini akan digunakan pengembangan dengan cara Code First.

API untuk akses data yang dibuat pada pengembangan Code First ini berdasarkan dari class DbContext. API ini juga dapat dimanfaatkan ketika melakukan pengembangan dengan pendekatan Database First dan Model First. Untuk informasi lebih jauh tentang ini dapat mengunjungi posting When is Code First not code first? (<http://blogs.msdn.com/b/adonet/archive/2011/03/07/when-is-code-first-not-code-first.aspx>) pada blog dari team Entity Framework.

## POCO (Plain Old CLR Object)

Secara umum ketika menggunakan pengembangan dengan pendekatan Database First atau Model First maka class entity yang terdapat pada data model merupakan turunan dari class EntityObject (<http://msdn.microsoft.com/en-us/library/system.data.objects.dataclasses.entityobject.aspx>), yang memberikan kemampuan sesuai dengan fungsi-fungsi Entity Framework. Dengan begitu artinya class-class tersebut secara teknis bukan “Persistence Ignorant” (<http://msdn.microsoft.com/en-us/magazine/dd882510.aspx#id0420053>) dan tidak sepenuhnya sesuai dengan kebutuhan dari Domain-Driver Design (<http://msdn.microsoft.com/en-us/magazine/dd419654.aspx>). Semua pendekatan pengembangan pada Entity Framework dapat juga bekerja dengan class POCO (plain old CLR object), yang pada dasarnya class tersebut adalah “Persistence Ignorant” karena tidak mewarisi dari class EntityObject. Pada tutorial ini akan digunakan class POCO.

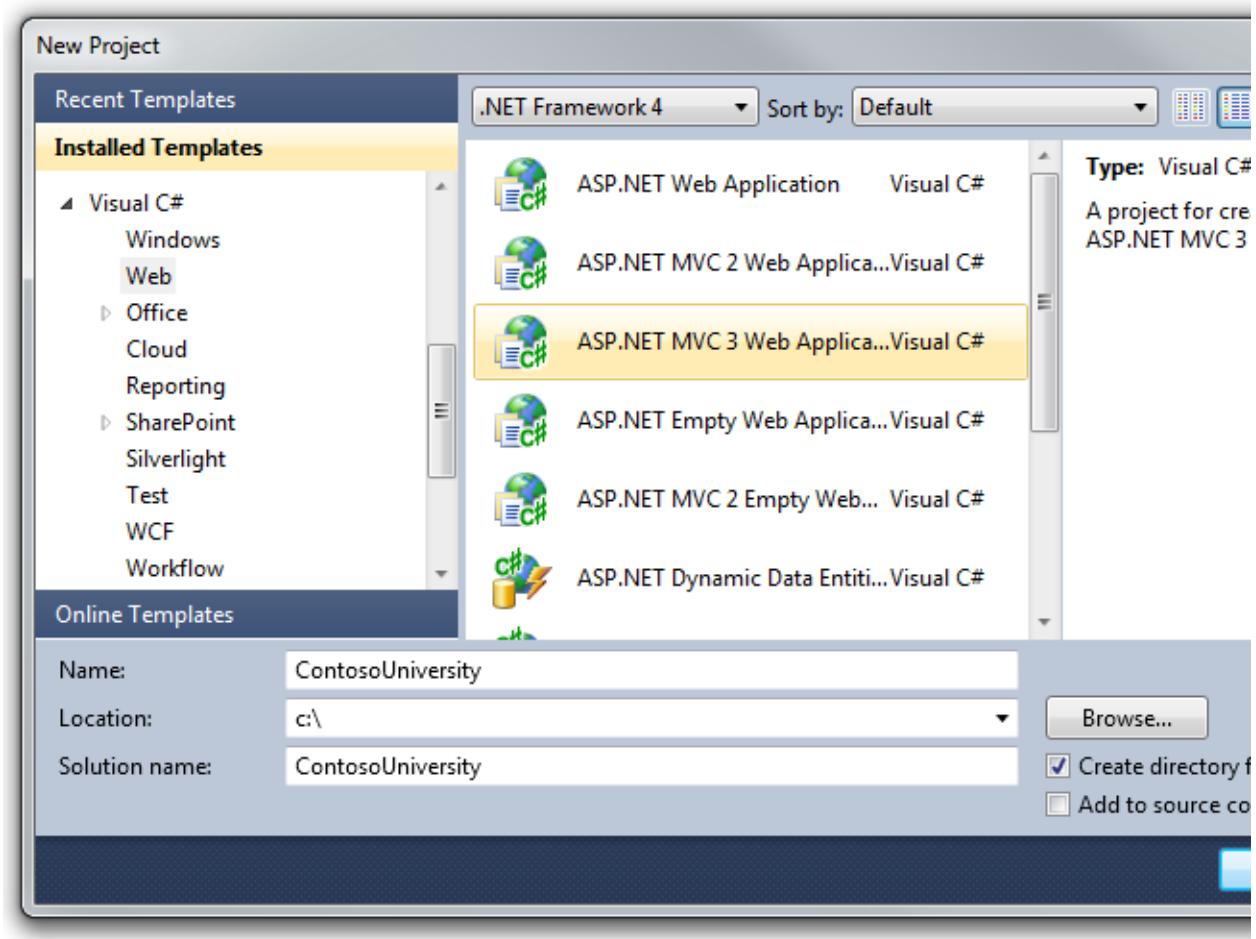
## Membuat Aplikasi Web MVC

Sebelum memulai mengikuti langkah-langkah pada tutorial ini maka terlebih dahulu pastikan software berikut ini sudah terinstall pada komputer :

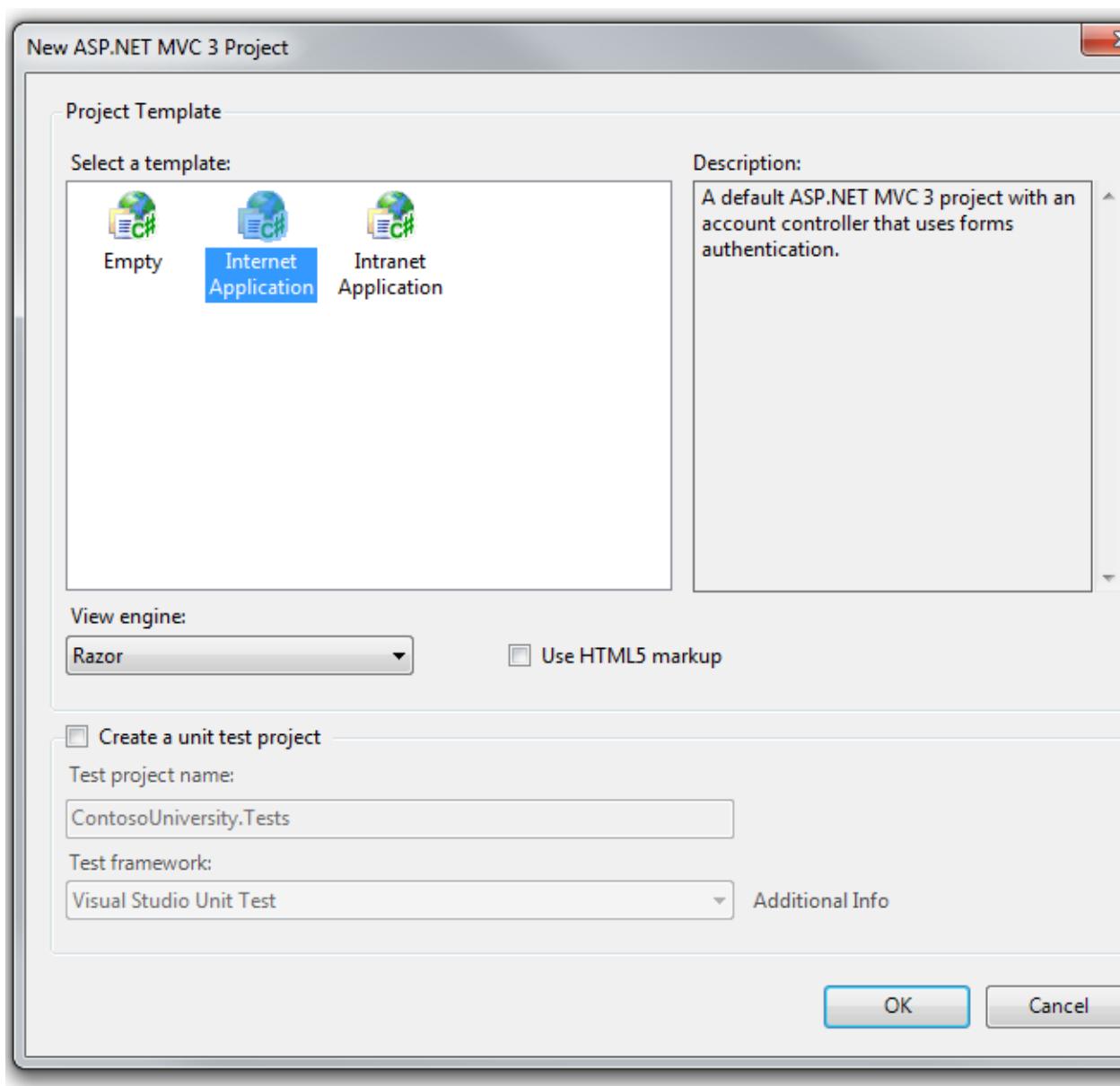
- Visual Studio 2010 SP1  
(<http://www.microsoft.com/web/gallery/install.aspx?appsxml=&appid=VS2010SP1Pack>) atau Visual Web Developer Express 2010 SP1  
(<http://www.microsoft.com/web/gallery/install.aspx?appsxml=&appid=VWD2010SP1Pack>).

- ASP.NET MVC 3 Tool Update (<http://www.microsoft.com/web/gallery/install.aspx?appid=MVC3>).
- Microsoft SQL Server Compact 4.0 (<http://www.microsoft.com/web/gallery/install.aspx?appid=SQLCE>).
- Microsoft Visual Studio 2010 SP1 Tools for SQL Server Compact 4.0 (<http://www.microsoft.com/web/gallery/install.aspx?appid=SQLCEVSTools>).

Langkah pertama adalah membuka Visual Studio and membuat project baru dengan nama “ContosoUniversity” dengan menggunakan template **ASP.NET MVC 3 Web Application**.



Pada jendela dialog **New ASP.NET MVC 3 Project** pilih template **Internet Application** and gunakan **Razor** sebagai view engine, kemudian hilangkan centang pada **Create a unit test project** dan lanjutkan dengan klik tombol **OK**.



## Style Website

Pada langkah selanjutnya akan dilakukan perubahan sederhana pada menu website, layout dan halaman home.

Untuk melakukan set up menu pada website Contoso University dapat dilakukan dengan mengedit file `Views\Shared\_Layout.cshtml`, jika diinginkan dapat dengan mengubah text pada heading `h1` dan link pada menu, seperti yang ditunjukkan pada contoh di bawah ini.

```
<!DOCTYPE html>  
<html>
```

```
<head>

    <title>@ViewBag.Title</title>

    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript">
    </script>

</head>

<body>

    <div class="page">

        <div id="header">

            <div id="title">

                <h1>Contoso University</h1>

            </div>

            <div id="logindisplay">

                @Html.Partial("_LogOnPartial")

            </div>

            <div id="menucontainer">

                <ul id="menu">

                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Students", "Index", "Student")</li>
                    <li>@Html.ActionLink("Courses", "Index", "Course")</li>
                    <li>@Html.ActionLink("Instructors", "Index", "Instructor")</li>
                    <li>@Html.ActionLink("Departments", "Index", "Department")</li>

                </ul>

            </div>

        </div>

        <div id="main">
```

```

@RenderBody()

</div>

<div id="footer">

</div>

</div>

</body>

</html>

```

Pada file *Views\Home\Index.cshtml* , hapus semua yang berada di bawah heading *h2*.

Pada file *Controllers\HomeController.cs*, cari kalimat "Welcome to ASP.NET MVC!" dan ganti dengan kalimat berikut "Welcome to Contoso University!".

Pada file *Content\Site.css*, perubahan pada file ini untuk membuat tab menu menjadi rapat kiri.

- Pada bagian #main tambahkan clear : both; seperti pada contoh di bawah ini.

```

#main
{
    clear: both;
    padding: 30px 30px 15px 30px;
    background-color: #fff;
    border-radius: 4px 0 0 0;
    -webkit-border-radius: 4px 0 0 0;
    -moz-border-radius: 4px 0 0 0;
}

```

- Pada bagian nav and #menucontainer, tambahkan clear: both; float: left; seperti pada contoh berikut.

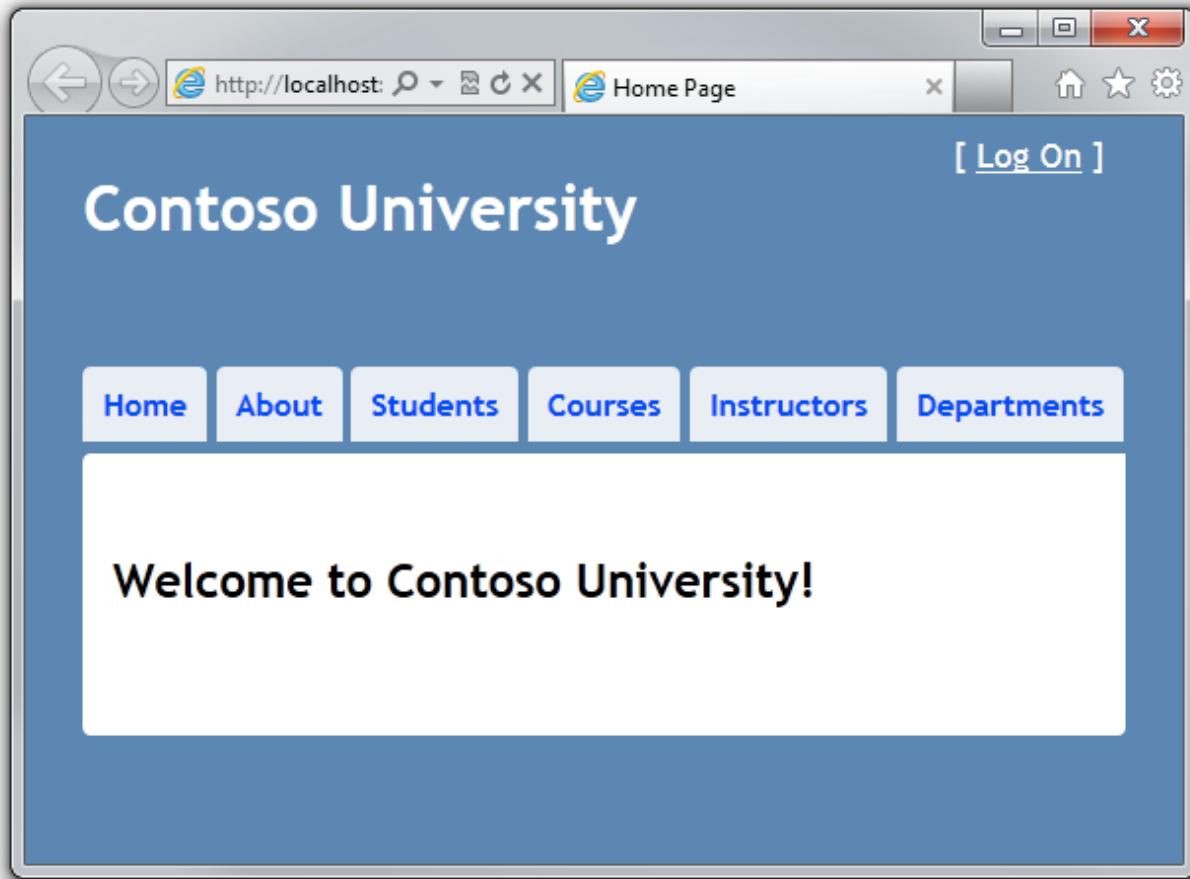
```

nav, #menucontainer
{
    margin-top: 40px;
}

```

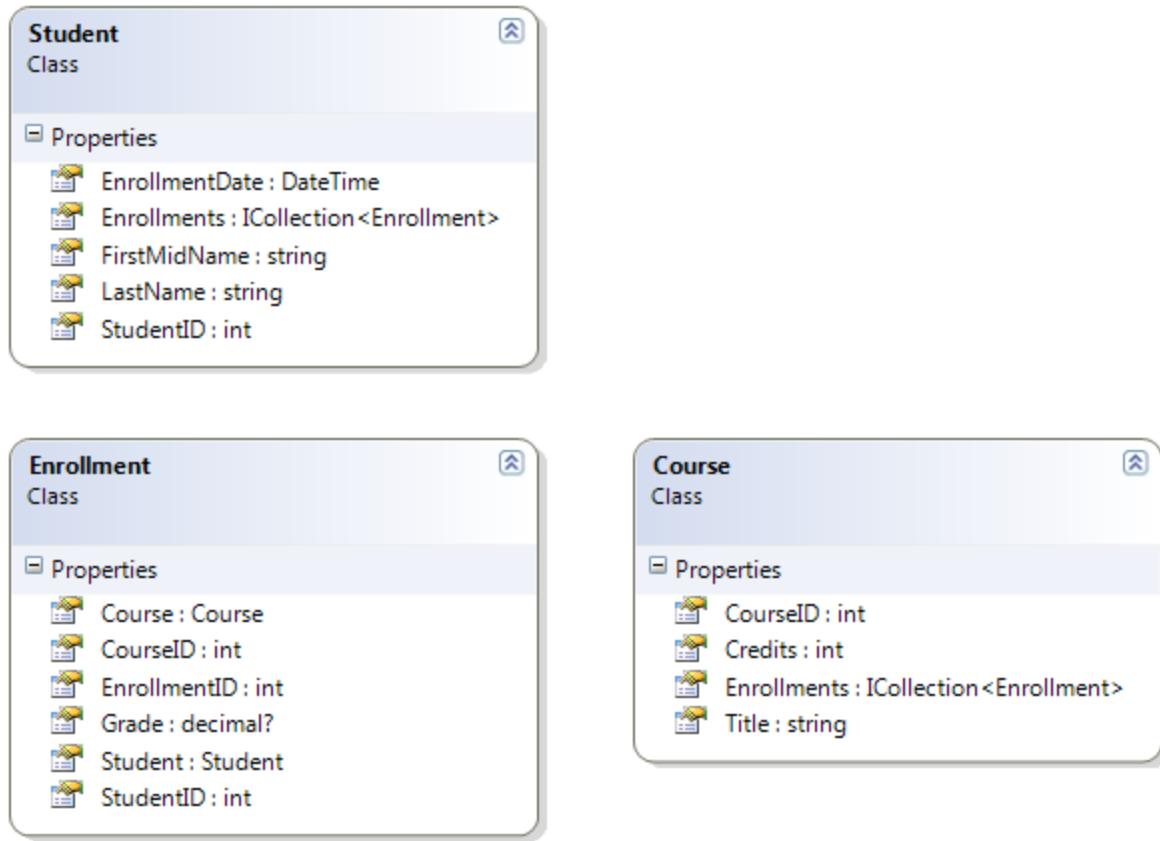
```
clear: both;  
float: left;  
}
```

Berikut adalah antarmuka muka halaman muka website setelah hasil perubahan di atas.



## Membuat Data Model

Langkah selanjutnya adalah membuat class entity untuk aplikasi web Universitas Contoso. Berikut adalah tiga class entity awal untuk aplikasi web ini.



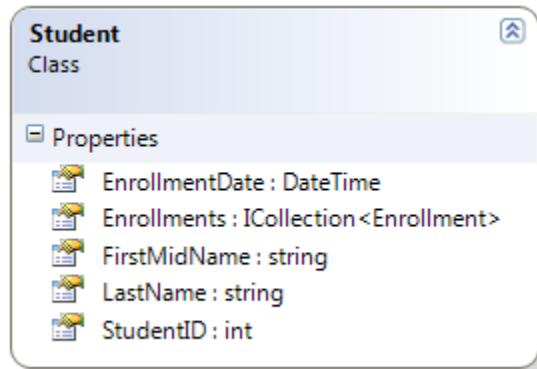
Terdapat relasi one-to-many antara entity **Student** dan **Enrollment**, dan terdapat relasi one-to-many antara entity **Course** dan **Enrollment**. Dengan kata lain Student dapat mendaftarkan dirinya ke lebih dari satu Course dan Course dapat memiliki banyak Student.

Berikutnya akan diperlihatkan langkah untuk membuat class untuk setiap entity yang telah disebutkan di atas.

#### Note

Jika kompilasi project dilakukan sebelum semua class entity dibuat maka akan didapati pesan error saat kompilasi.

## Student Entity



Pada folder *Models*, buat *Student.cs* dan gunakan kode dibawah ini sebagai isinya.

```
using System;
using System.Collections.Generic;

namespace ContosoUniversity.Models

{
    public class Student
    {
        public int StudentID { get; set; }

        public string LastName { get; set; }

        public string FirstMidName { get; set; }

        public DateTime EnrollmentDate { get; set; }

        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

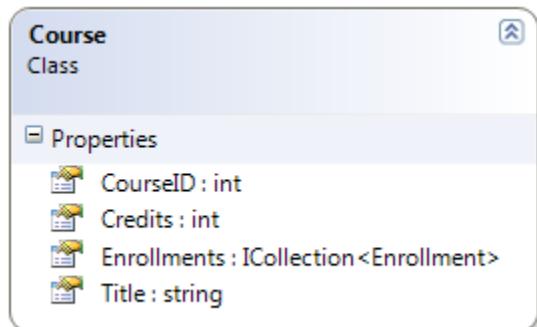
Property `StudentID` akan menjadi kolom primary key pada tabel dalam database yang bersesuaian dengan class ini. Secara umum Entity Framework akan menafsikan property dengan nama `ID` atau `classnameID` sebagai primary key.

Property `Enrollment` adalah navigation property. Navigation property bertugas untuk mempertahankan entity lain yang terkait dengan entity ini. Pada kasus ini property `Enrollment` pada

entity `Student` akan menghubungkan semua entity `Enrollment` yang berhubungan dengan entity `Student`. Dengan kata lain jika pada database terdapat row `Student` yang mempunya dua hubungan dengan row `Enrollment` maka artinya navigation property `Enrollment` milik entity `Student` akan memiliki dua entity `Enrollment`.

Navigation property didefinisikan sebagai `virtual` hal ini menjadi kelebihan sendiri pada fungsi Entity Framework yang disebut *lazy loading*. (Lazy loading akan dijelaskan kemudian, pada tutorial Reading Related Data <http://www.asp.net/entity-framework/tutorials/reading-related-data-with-the-entity-framework-in-an-asp-net-mvc-application> pada seri ini.) Jika navigation property dapat menangani lebih dari satu entity (relasi many-to-many atau one-to-many) maka tipenya yang digunakan untuk property ini adalah `Icollection`.

## Course Entity



Pada folder Models, buat Course.cs dan gunakan kode di bawah ini sebagai isinya.

```
using System;
using System.Collections.Generic;

namespace ContosoUniversity.Models
{
    public class Course
    {
        public int CourseID { get; set; }

        public string Title { get; set; }

        public int Credits { get; set; }

        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

```
}
```

Property Enrollment adalah navigation property. Entity Course dapat berhubungan dengan lebih dari satu entity Enrollment.

## Membuat Database Context

class utama yang mengkoordinasi fungsi Entity Framework pada data model yang ada disebut class database context. Class ini dibuat dengan berasal dari class `System.Data.Entity.DbContext`. Pada kode yang dibuat ditentukan entity yang termasuk dalam data model dan melakukan penyesuaian perilaku tertentu Entity Framework. Pada project ini, class tersebut diberi nama `SchoolContext`.

Pertama buat folder *DAL*, dalam folder tersebut buat file class dengan nama `SchoolContext.cs` dan gunakan kode di bawah ini.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using ContosoUniversity.Models;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace ContosoUniversity.Models
{
    public class SchoolContext : DbContext
    {
        public DbSet<Student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Course> Courses { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
}
```

```
    }  
}
```

Kode tersebut membuat property `DbSet` untuk setiap entity set. Pada terminologi Entity Framework, sebuah entity set akan sesuai dengan tabel pada database dan setiap property akan sesuai dengan row pada tabel.

Statement yang terdapat di dalam method `OnModelCreating` untuk menjaga agar nama tabel menjadi jamak. Jika hal ini tidak dilakukan maka nama tabel yang dibuat akan menjadi `Students`, `Courses`, dan `Enrollments`. Developer mungkin tidak terlalu perduli tentang apakah nama tabel dalam bentuk jamak atau tidak. Tetapi dalam tutorial ini digunakan nama tabel dalam bentuk tunggal, tetapi developer bebas untuk memilih yang diinginkan cukup dengan menggunakan baris statement tersebut atau tidak.

(Class di atas dikelompokkan dalam namespace `Models`, karena dalam beberapa kasus Code First mempunyai asumsi bahwa class entity dan class context berada dalam satu namespace.)

## Membuat Connection String

Bila *connection string* belum dibuat, maka secara otomatis Entity Framework akan membuat sebuah connection string secara otomatis. Pada tutorial ini walau digunakan SQL Server Compact tetap diperlukan pembuatan *connection string* untuk melakukan koneksi.

Pertama buka file `Web.config` dan tambahkan connection string pada bagian `connectionString`, seperti pada contoh di bawah ini. (Pastikan file `Web.config` yang diupdate adalah yang terletak pada root folder. Pada subfolder `Views` juga terdapat file `Web.config`, file ini tidak perlu diupdate).

```
<add name="SchoolContext" connectionString="Data Source=|DataDirectory|School.sdf"  
providerName="System.Data.SqlServerCe.4.0"/>
```

Biasanya Entity Framework akan mencari connection string dengan nama yang sesuai nama class object context. Connection string di atas memberikan informasi bahwa nama database dari SQL Server Compact adalah `School.sdf` yang terletak pada folder `App_Data`.

## Inisialisasi Database

Entity Framework dapat melakukan pembuatan (menghapus dan membuat ulang) database secara otomatis saat aplikasi dijalankan. Hal ini dapat didefinisikan sendiri oleh developer, apakah hal

tersebut selalu dilakukan saat aplikasi dijalankan atau dilakukan hanya apabila terjadi perubahan pada model dan perlu dilakukan sinkronisasi dengan database yang ada.

Pada folder *DAL*, buat class dengan nama file *SchoolInitializer.cs* dan ganti kode yang ada dengan contoh berikut ini. kode ini berfungsi untuk membuat database ketika diperlukan dan mengisi database tersebut dengan data test.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public class SchoolInitializer : DropCreateDatabaseIfModelChanges<SchoolContext>
    {
        protected override void Seed(SchoolContext context)
        {
            var students = new List<Student>
            {
                new Student { FirstMidName = "Carson", LastName = "Alexander",
EnrollmentDate = DateTime.Parse("2005-09-01") },
                new Student { FirstMidName = "Meredith", LastName = "Alonso",
EnrollmentDate = DateTime.Parse("2002-09-01") },
                new Student { FirstMidName = "Arturo", LastName = "Anand",
EnrollmentDate = DateTime.Parse("2003-09-01") },
                new Student { FirstMidName = "Gytis", LastName = "Barzdukas",
EnrollmentDate = DateTime.Parse("2002-09-01") },
                new Student { FirstMidName = "Yan", LastName = "Li",
EnrollmentDate = DateTime.Parse("2002-09-01") },
                new Student { FirstMidName = "Peggy", LastName = "Justice",
EnrollmentDate = DateTime.Parse("2002-09-01") }
            };
            context.Students.AddRange(students);
            context.SaveChanges();
        }
    }
}
```

```

EnrollmentDate = DateTime.Parse("2001-09-01") },
    new Student { FirstMidName = "Laura",      LastName = "Norman",
EnrollmentDate = DateTime.Parse("2003-09-01") },
    new Student { FirstMidName = "Nino",       LastName = "Olivetto",
EnrollmentDate = DateTime.Parse("2005-09-01") }

};

students.ForEach(s => context.Students.Add(s));

context.SaveChanges();

var courses = new List<Course>
{
    new Course { Title = "Chemistry",      Credits = 3, },
    new Course { Title = "Microeconomics", Credits = 3, },
    new Course { Title = "Macroeconomics", Credits = 3, },
    new Course { Title = "Calculus",        Credits = 4, },
    new Course { Title = "Trigonometry",   Credits = 4, },
    new Course { Title = "Composition",    Credits = 3, },
    new Course { Title = "Literature",     Credits = 4, }
};

courses.ForEach(s => context.Courses.Add(s));

context.SaveChanges();

var enrollments = new List<Enrollment>
{
    new Enrollment { StudentID = 1, CourseID = 1, Grade = 1 },
    new Enrollment { StudentID = 1, CourseID = 2, Grade = 3 },
    new Enrollment { StudentID = 1, CourseID = 3, Grade = 1 },
    new Enrollment { StudentID = 2, CourseID = 4, Grade = 2 },
    new Enrollment { StudentID = 2, CourseID = 5, Grade = 4 },
}

```

```

        new Enrollment { StudentID = 2, CourseID = 6, Grade = 4 },
        new Enrollment { StudentID = 3, CourseID = 1 },
        new Enrollment { StudentID = 4, CourseID = 1 },
        new Enrollment { StudentID = 4, CourseID = 2, Grade = 4 },
        new Enrollment { StudentID = 5, CourseID = 3, Grade = 3 },
        new Enrollment { StudentID = 6, CourseID = 4 },
        new Enrollment { StudentID = 7, CourseID = 5, Grade = 2 },
    };

    enrollments.ForEach(s => context.Enrollments.Add(s));

    context.SaveChanges();

}
}

}

```

Method **Seed** memiliki inputan dari objek database context, dan kode didalamnya mempunyai fungsi untuk menambah entity-entity baru ke dalam database. Untuk setiap tipe entity, kode membuat koleksi untuk entity-entity baru, menambahkan koleksi tersebut ke property **DbSet**, dan kemudian menyimpan perubahannya ke dalam database. Pada kode di atas dapat dilihat terdapat pemanggilan method **SaveChanges** pada setiap group dari entity, hal seperti itu sebenarnya tidak perlu karena bisa dilakukan sekali saja. Tetapi hal diatas dilakukan untuk menghindari jika terjadi masalah saat kode melakukan penulisan ke database.

Lakukan penambahan kode pada file Global.asax.cs yang berfungsi sebagai kode yang akan dijalankan ketika aplikasi dimulai.

- Menambahkan kata kunci using :

```

using System.Data.Entity;
using ContosoUniversity.Models;
using ContosoUniversity.DAL;

```

- Memanggil kode yang akan dipanggil setiap aplikasi dijalankan pada method Application\_Start

```
Database.SetInitializer<SchoolContext>(new SchoolInitializer());
```

Sekarang saat aplikasi dijalankan untuk pertama kali, aplikasi akan melakukan perbandingan antara model dengan database, jika terjadi perbedaan maka aplikasi akan melakukan penghapusan dan pembuatan ulang database.

#### Note

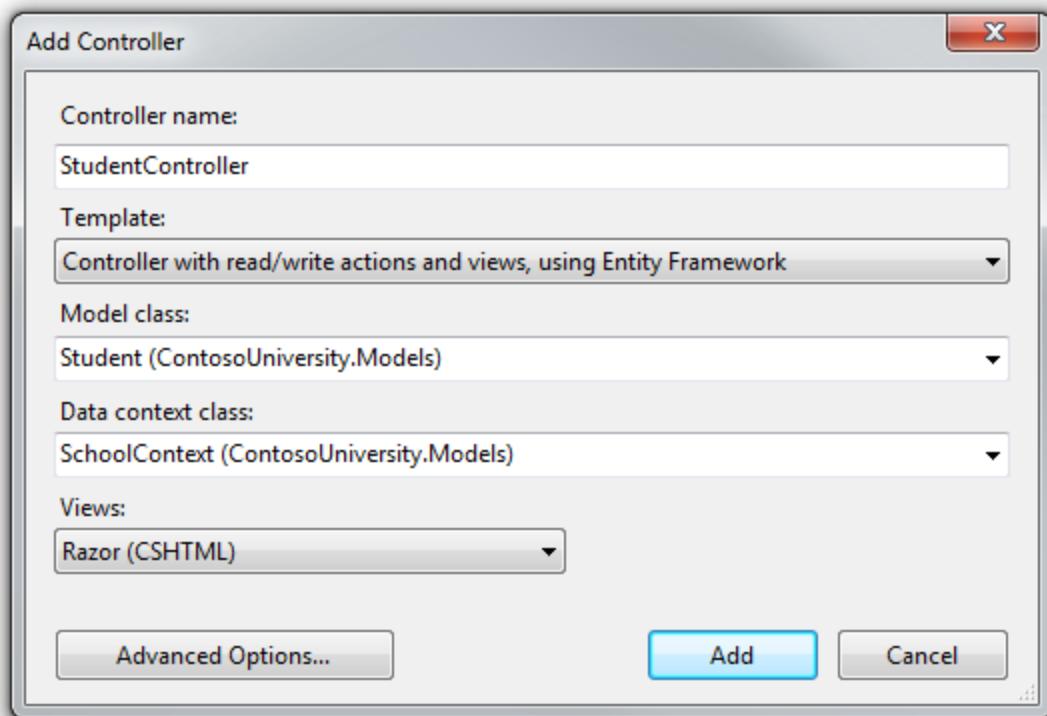
Jika aplikasi akan dideploy pada server produksi, maka kode method Seed harus dihilangkan.

Langkah selanjutnya ada dibuat halaman untuk menampilkan data dan proses request data akan secara otomatis melakukan pembuatan database. Langkah yang pertama dilakukan untuk bagian ini adalah membuat controller. Tetapi sebelum hal itu dilakukan, terlebih dahulu build project agar model dan class context tersedia untuk *MVC controller scaffolding*.

## Membuat Student Controller

Untuk membuat **Student** controller, klik kanan pada folder **Controller** di **Solution Explorer**, pilih **Add** dan kemudian klik **Controller**. Pada dialog box **Add Controller**, ikuti petunjuk di bawah ini kemudian klik **Add** :

- Nama controller : **StudentController**.
- **Template : Controller with read/write actions and views, using Entity Framework.**
- Class model : Student (ContosoUniversity.Models). (Jika pilihan ini tidak terdapat pada dropdown list, build project sekali lagi).
- Class data context : **SchoolContext (ContosoUniversity.Models)**.
- Views : Razor (CSHTML).



Buka file `Controllers\StudentController.cs`. Pada kode di bawah ini dapat dilihat terdapat variable pada class yang menginisiasi object database context.

```
private SchoolContext db = new SchoolContext();
```

Method action `Index` berfungsi untuk mengambil data siswa dari property `Students` pada instan database context.

```
public ViewResult Index()
{
    return View(db.Students.ToList());
}
```

Scaffolding otomatis juga telah membuat view untuk `Student`. Untuk melakukan kostumisasi pada heading dan kolom pada view `Index` dapat dilakukan dengan cara mengedit file `Views\Student\Index.cshtml` dan ganti kode yang ada dengan kode berikut ini.

```
@model IEnumerable<ContosoUniversity.Models.Student>

@{
    ViewBag.Title = "Students";
}



## Students



@Html.ActionLink("Create New", "Create")

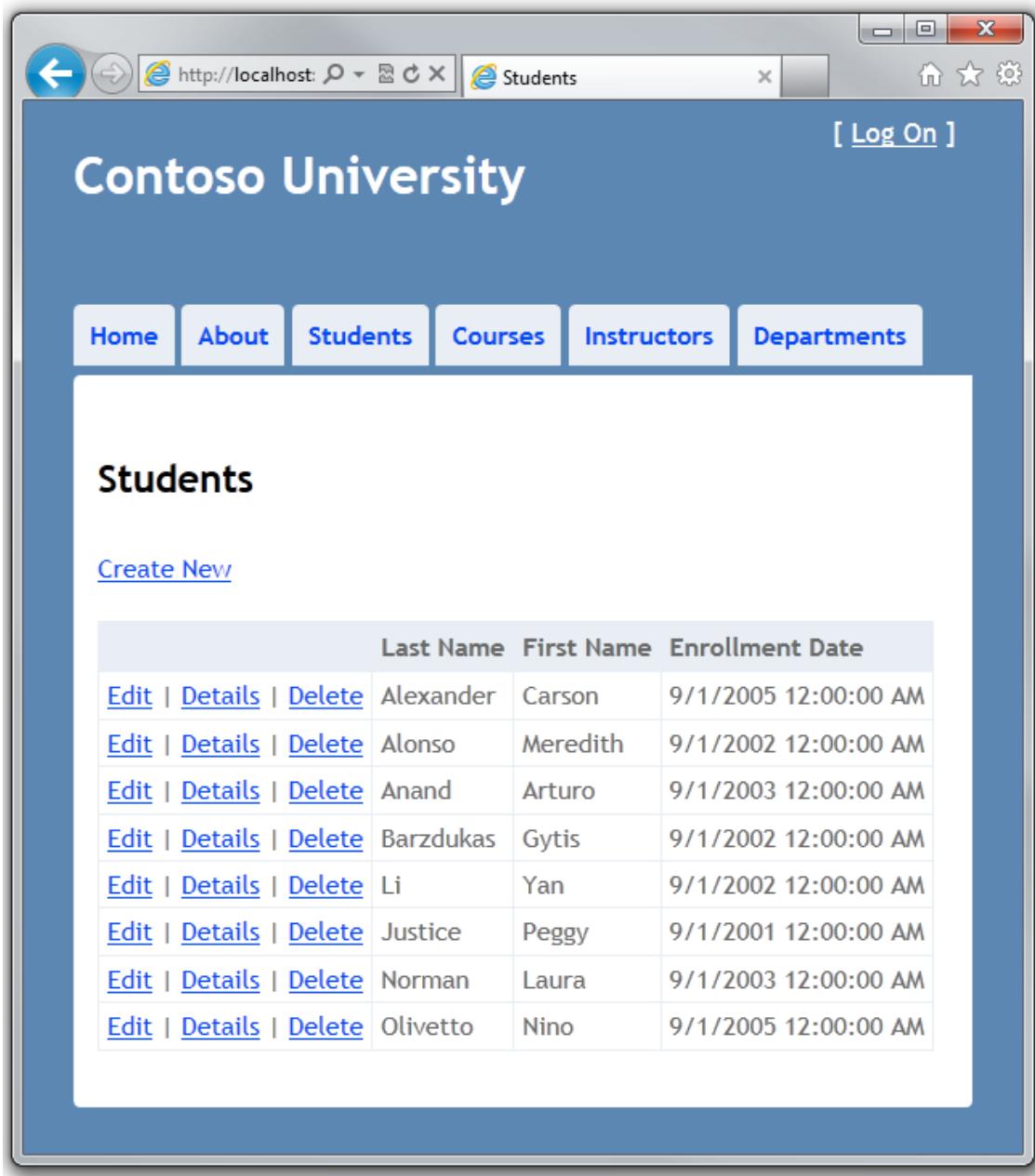


| </th> | Last Name | First Name | Enrollment Date |
|-------|-----------|------------|-----------------|
|-------|-----------|------------|-----------------|

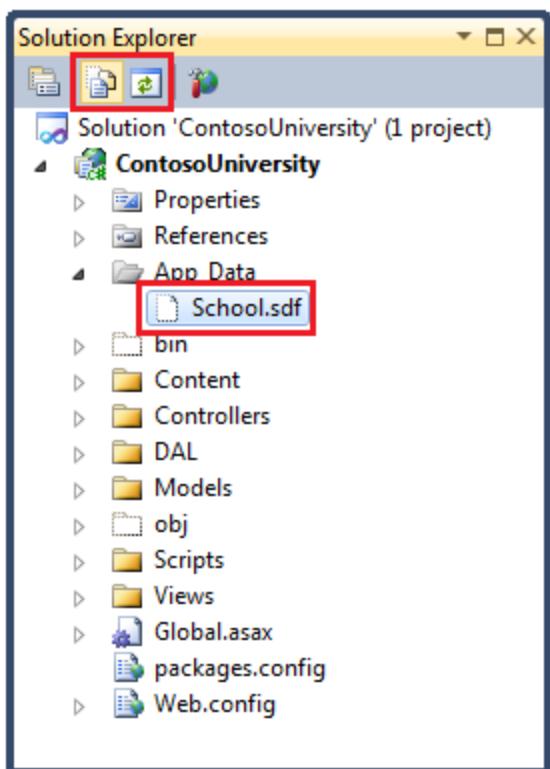

```

```
</td>
<td>
    @Html.DisplayFor(modelItem => item.LastName)
</td>
<td>
    @Html.DisplayFor(modelItem => item.FirstMidName)
</td>
<td>
    @Html.DisplayFor(modelItem => item.EnrollmentDate)
</td>
</tr>
}
</table>
```

Jalankan website dan klik tab **Student** maka akan dapat dilihat antarmuka seperti berikut ini.



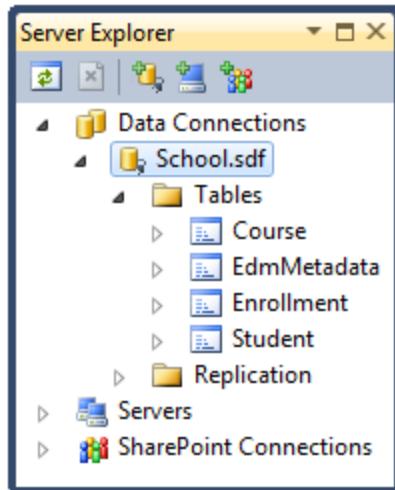
Tutup broser. Pada **Solution Explorer**, pilih project **ContosoUniversity** (pastikan yang terpilih adalah project, bukan solution). Klik tombol **Show all Files** dan klik **Refresh** dan kemudian buka folder **App\_Data**, maka akan dapat dilihat file **School.sdf**.



Klik dua kali pada file *School.sdf* untuk membuka file tersebut pada **Server Explorer**. Kemudian buka folder **Tables** maka dapat dilihat bahwa tabel telah dibuat pada database.

Note

Jika terjadi error saat melakukan klik dua kali pada file *School.sdf* maka lakukan pemeriksaan apakah **Visual Studio 2010 SP1 Tools for SQL Server Compact 4.0** sudah diinstall.

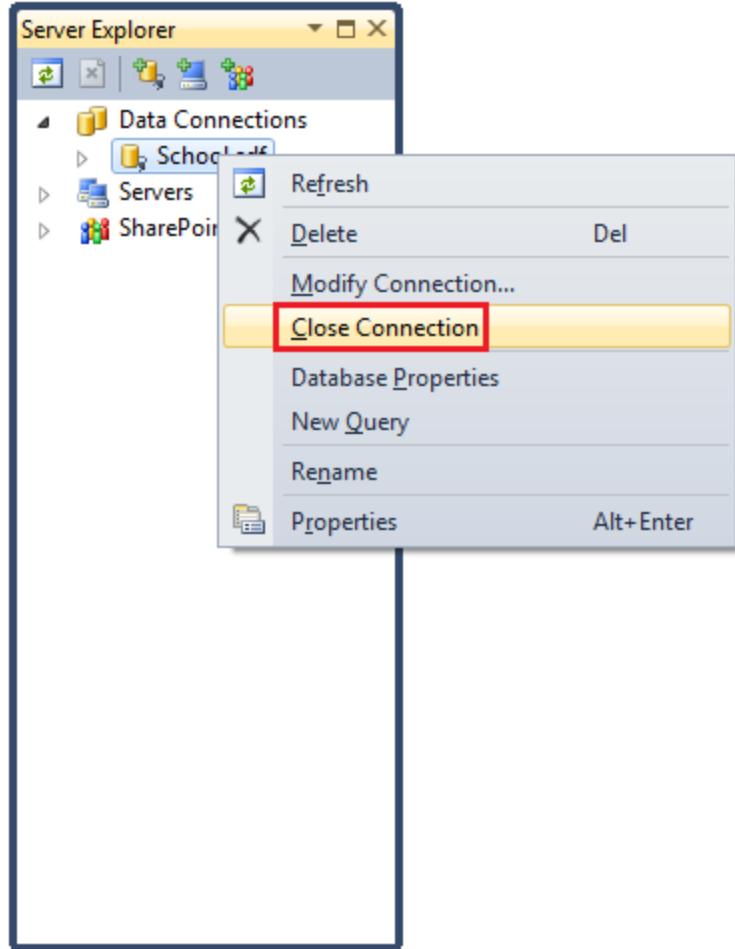


Terdapat satu tabel untuk setiap entity, ditambah sebuah tabel tambahan. Tabel EdmMetadata digunakan oleh Entity Framework untuk menentukan kapan model dan database tidak sama.

Klik kanan pada salah satu tabel dan pilih **Show Table Data** untuk melihat data yang telah disimpan ke dalam tabel hasil dari class `SchoolInitializer`.

	StudentID	LastName	FirstMidName	EnrollmentDate
▶	1	Alexander	Carson	9/1/2005 12:00:...
	2	Alonso	Meredith	9/1/2002 12:00:...
	3	Anand	Arturo	9/1/2003 12:00:...
	4	Barzdukas	Gytis	9/1/2002 12:00:...
	5	Li	Yan	9/1/2002 12:00:...
	6	Justice	Peggy	9/1/2001 12:00:...
	7	Norman	Laura	9/1/2003 12:00:...
	8	Olivetto	Nino	9/1/2005 12:00:...
*	NULL	NULL	NULL	NULL

Tutup koneksi jika telah selesai melakukan hal tersebut di atas.



## Convention

Kode yang telah ditulis agar Entity Framework dapat membuat database lengkap adalah minimal karena penggunaan convention atau asumsi yang dibuat oleh Entity Framework. Beberapa hal-hal tersebut adalah :

- Bentuk jamak dari class entity digunakan sebagai nama tabel.
- Nama property entity digunakan untuk nama kolom.
- Property entity yang bernama `ID` atau `classnameID` dikenali sebagai property primary key.
- Entity Framework melakukan koneksi ke database dengan mencari connection string yang mempunyai nama sama dengan nama class context (dalam kasus ini adalah `SchoolContext`).

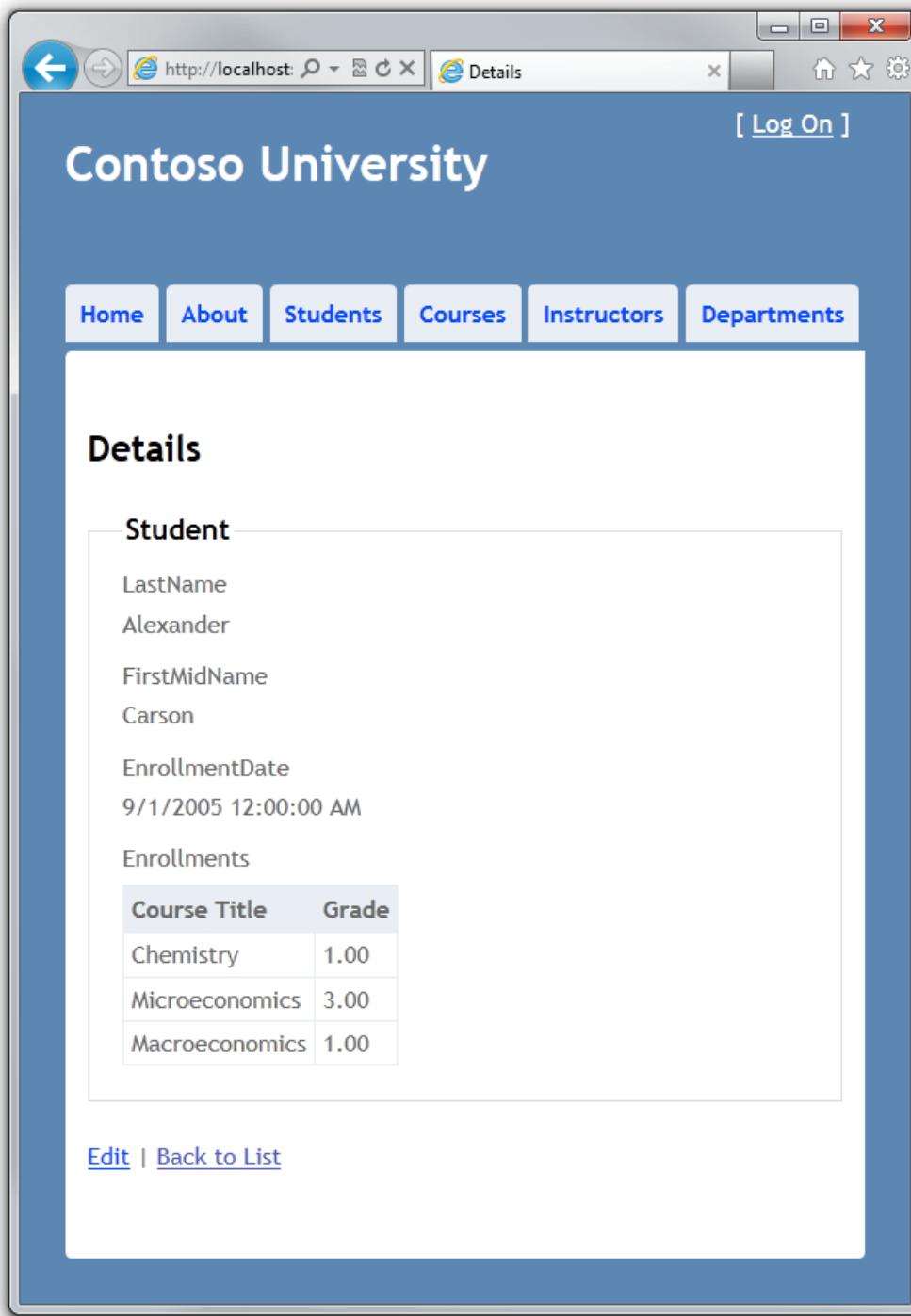
Convention yang telah disebutkan diatas tentu saja masih dapat di-override sesuai kebutuhan, seperti pada contoh di atas dimana nama tabel tidak menggunakan bentuk jamak, nanti akan diajarkan lebih banyak lagi tentang convention dan bagaimana melakukan override pada tutorial Creating a More Complex Data Model (<http://www.asp.net/entity-framework/tutorials/creating-a-more-complex-data-model-for-an-asp-net-mvc-application>).

Pada tutorial ini telah dibuat aplikasi sederhana dengan menggunakan Entity Framework dan SQL Server Compact untuk menyimpan data dan menampilkannya. Pada bagian selanjutnya akan diajarkan bagaimana membuat operasi CRUD (create, read, update, delete).

Link lain yang berhubungan dengan tutorial Entity Framework dapat ditemukan pada link berikut  
<http://www.asp.net/entity-framework/tutorials/advanced-entity-framework-scenarios-for-an-mvc-web-application>.

# Implementasi Fungsi CRUD dengan menggunakan Entity Framework pada Aplikasi ASP.NET MVC

Pada tutorial ini akan dibuat beberapa halaman web seperti berikut.



[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Edit

**Student**

Last Name

First/Middle Name

Enrollment Date

[Back to List](#)

[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Create

**Student**

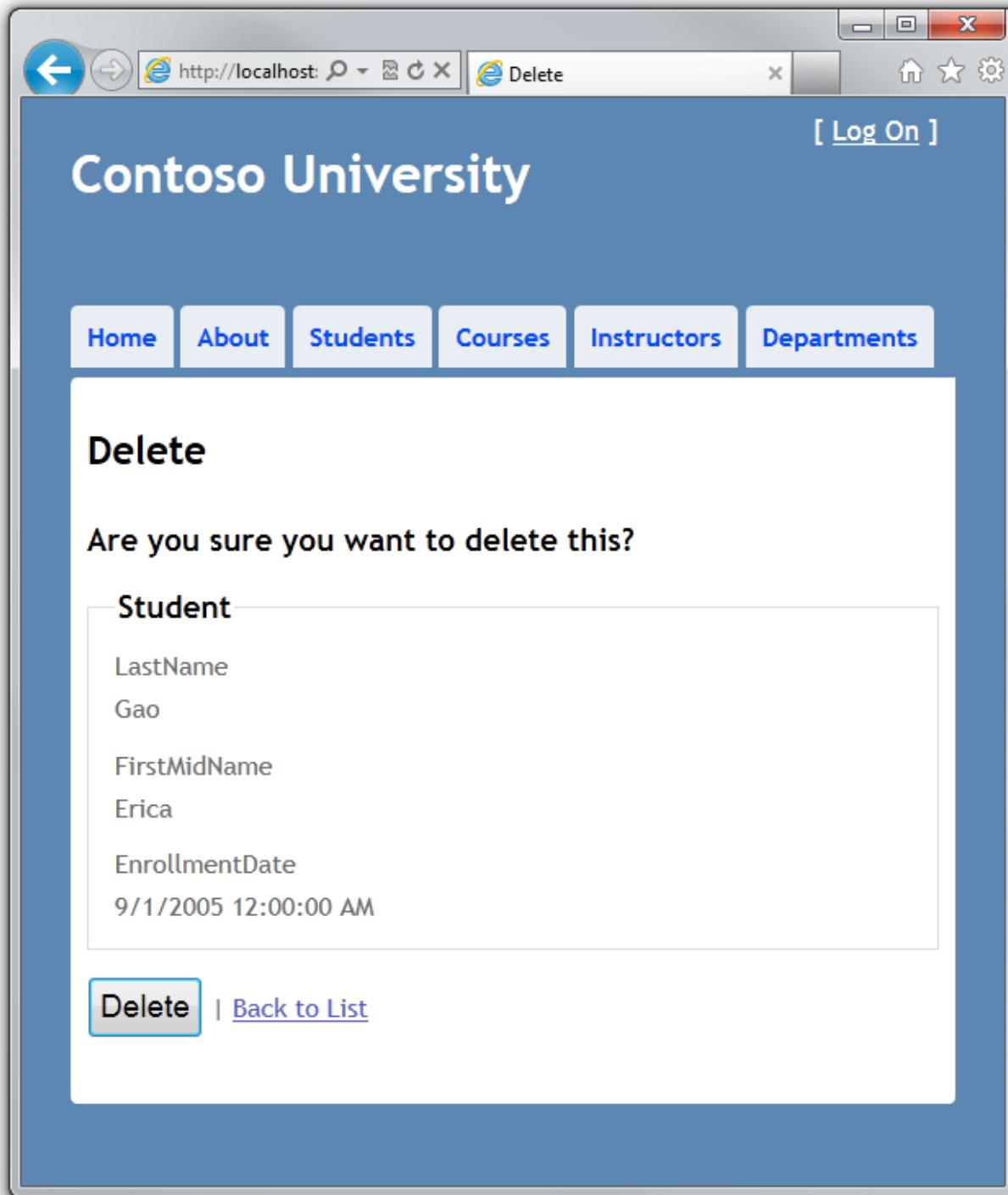
LastName

FirstMidName

EnrollmentDate

**Create**

[Back to List](#)



## Membuat Halaman Detail

Pada kode scaffolded untuk halaman Index tidak memasukkan property Enrollments dikarenakan property tersebut berisi collection. Pada halaman Detail baru akan ditampilkan collection tersebut.

Pada file *Controllers\StudentController.cs*, method action untuk view Detail adalah seperti berikut :

```
public ViewResult Details(int id)
{
    Student student = db.Students.Find(id);
    return View(student);
}
```

Kode di atas menggunakan method `Find` untuk mengambil sebuah entity tunggal `Student` yang sesuai dengan nilai key yang diberikan pada parameter `id`.

Buka file `Views\Student\Details.cshtml`. Semua field yang ditampilkan menggunakan helper `DisplayFor`, seperti pada contoh berikut.

```
<div class="display-label">LastName</div>
<div class="display-field">
    @Html.DisplayFor(model => model.LastName)
</div>
```

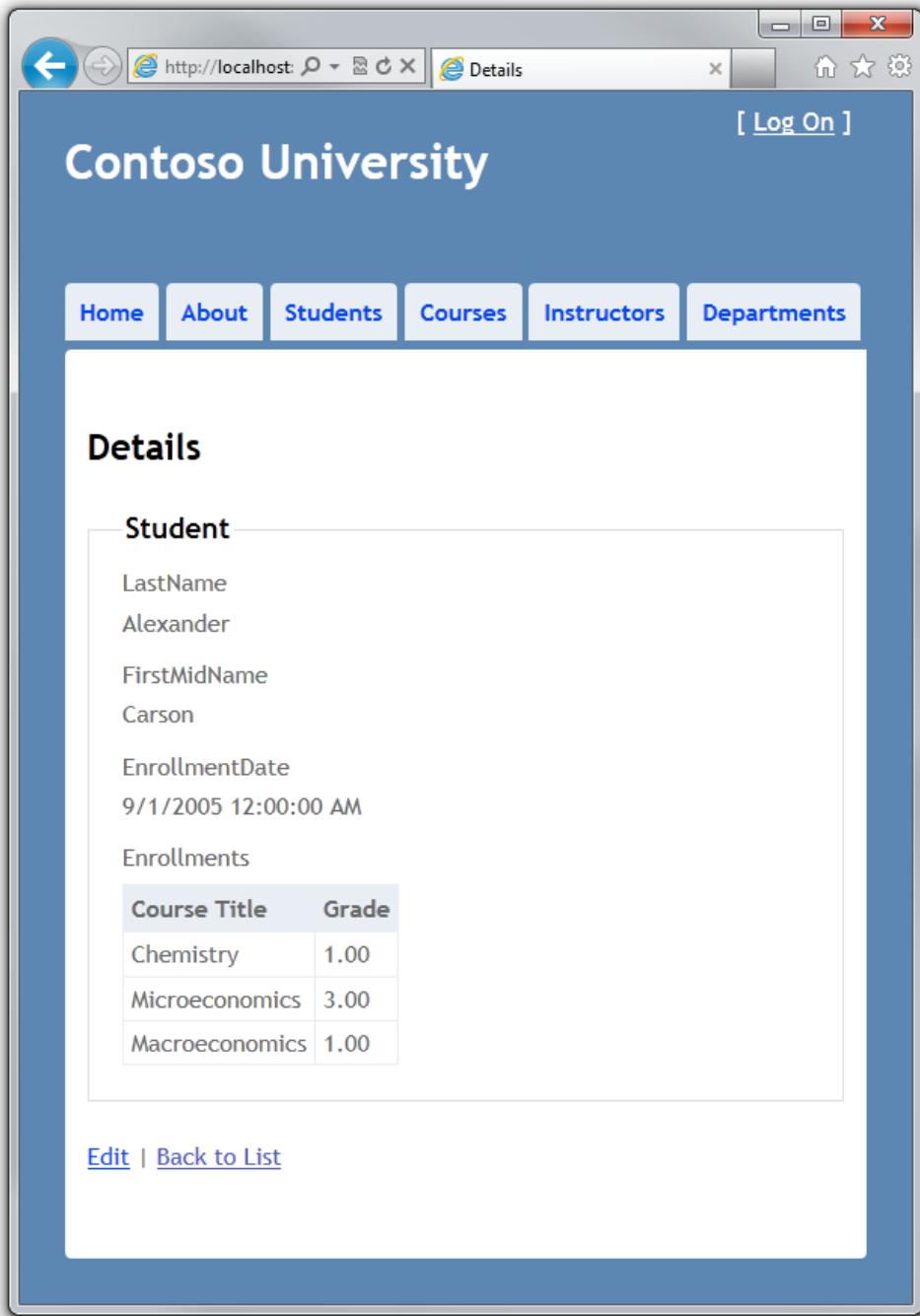
Sedangkan untuk menampilkan daftar enrollment, tambahkan kode berikut ini setelah field `EnrollmentDate` dan sebelum tag penutup `fieldset`.

```
<div class="display-label">
    @Html.LabelFor(model => model.Enrollments)
</div>
<div class="display-field">
    <table>
        <tr>
            <th>Course Title</th>
            <th>Grade</th>
        </tr>
        @foreach (var item in Model.Enrollments)
        {
            <tr>
                <td>@item.CourseTitle</td>
                <td>@item.Grade</td>
            </tr>
        }
    </table>
</div>
```

```
<tr>
    <td>
        @Html.DisplayFor(modelItem => item.Course.Title)
    </td>
    <td>
        @Html.DisplayFor(modelItem => item.Grade)
    </td>
</tr>
}
</table>
</div>
```

Kode tersebut melakukan pengulangan entity dalam navigation property `Enrollments`. Untuk setiap entity `Enrollments` dalam property tersebut akan menampilkan course title dan grade. Course title didapat dari entity `Course` pada navigation property `Course` pada entity `Enrollments`. Seluruh data tersebut didapat dari database ketika diperlukan. (Dalam istilah lainnya adalah lazy loading. Developer tidak perlu menentukan kapan loading dilakukan, jadi saat pertama kali akses terhadap property tersebut dilakukan, maka query akan dikirimkan ke database untuk melakukan pengambilan data. Bahasan lebih lanjut tentang lazy loading dapat ditemukan pada tutorial berikutnya Reading Related Data <http://www.asp.net/entity-framework/tutorials/reading-related-data-with-the-entity-framework-in-an-asp-net-mvc-application>).

Jalankan halaman dengan cara memilih tab Students dan klik hyperlink Detail. Maka akan dapat dilihat halaman berikut ini.



## Membuat Halaman Create

Pada file `Controllers\StudentController.cs`, ganti method aksi `HttpPost Create` dengan kode berikut ini untuk menambahkan blog `try-catch` pada method scaffolded.

```
[HttpPost]  
public ActionResult Create(Student student)
```

```

{
    try
    {
        if (ModelState.IsValid)
        {
            db.Students.Add(student);

            db.SaveChanges();

            return RedirectToAction("Index");
        }
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        ModelState.AddModelError("", "Unable to save changes. Try again, and if the
problem persists see your system administrator.");
    }
    return View(student);
}

```

Kode ini menambahkan entity `Students` yang dibuat oleh model binder ASP.NET MVC ke dalam set entity `Students` kemudian menyimpannya perubahannya pada database. (Model binder merupakan fungsionalitas dari ASP.NET MVC untuk memudahkan dalam bekerja dengan data yang disubmit oleh form; model binder mengubah nilai-nilai yang dikirimkan menjadi tipe pada .NET Framework dan mengirimkannya kepada method action sebagai parameter. Pada kasus ini, model binder menginstansiasi entity `Students` dengan menggunakan nilai property dari collection `Form`).

Perbedaan antara kode ini dengan kode yang dibuat secara scaffolding otomatis adalah pada bagian block `try-catch`. Jika terdapat exception dari `DataException` tertangkap saat proses penyimpanan maka pesan kesalahan akan ditampilkan. Jenis kesalahan tersebut biasanya disebabkan oleh sesuatu yang bersifat extenal, bukan karena kesalahan dalam pemrograman, dan pada pesan tersebut pengguna diberikan pesan untuk mencoba lagi. Kode pada file `Views\Student\Create.cshtml` mempunyai isi yang mirip dengan isi pada file `Details.cshtml` kecuali pada bagian helper `EditorFor`

dan `ValidationMessageFor` yang digunakan pada setiap field. Berikut ini adalah contoh dari kode yang digunakan:

```
<div class="editor-label">  
    @Html.LabelFor(model => model.LastName)  
</div>  
  
<div class="editor-field">  
    @Html.EditorFor(model => model.LastName)  
    @Html.ValidationMessageFor(model => model.LastName)  
</div>
```

Tidak diperlukan perubahan pada file *Create.cshtml*.

Jalankan halaman dengan memilih tabs **Students** dan klik **Create New**.

The screenshot shows a web browser window with the URL `http://localhost`. The title bar says "Create". The main content area displays the "Contoso University" logo and navigation links for Home, About, Students, Courses, Instructors, and Departments. Below this, a "Create" section is shown for a "Student". It contains three input fields: "LastName", "FirstMidName", and "EnrollmentDate", each with a corresponding text input box. A "Create" button is located below the input fields. At the bottom of the form is a link "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Create

**Student**

LastName

FirstMidName

EnrollmentDate

**Create**

[Back to List](#)

Validasi data akan bekerja secara default. Masukan nama dan tanggal yang tidak valid dan klik tombol **Create** maka akan ditampilkan pesan kesalahan.

The screenshot shows a web browser window with the URL <http://localhost>. The page title is "Create" and the header says "[ Log On ]". The main content area displays the "Contoso University" logo. Below it is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The "Students" link is highlighted in blue. The main content area has a heading "Create" followed by a "Student" section. It contains three input fields: "LastName" with value "Gao", "FirstMidName" with value "Erica", and "EnrollmentDate" with value "9/31/2005". A red error message next to the date field states: "The value '9/31/2005' is not valid for EnrollmentDate.". Below the input fields is a "Create" button. At the bottom of the form is a link "Back to List".

Contoso University

[ Log On ]

Home About Students Courses Instructors Departments

Create

Student

LastName

Gao

FirstMidName

Erica

EnrollmentDate

9/31/2005

The value '9/31/2005' is not valid for EnrollmentDate.

Create

[Back to List](#)

Pada kasus ini, validasi dilakukan pada sisi client dengan menggunakan Javascript. Tetapi juga terdapat validasi pada sisi server. Sehingga jika validasi pada sisi client gagal, maka data yang salah masih dapat ditangkap dan exception akan ditampilkan oleh kode server.

Masukkan data tanggal yang valid, misal 9/1/2005 dan kemudian klik **Create** maka akan dapat dilihat data baru ditampilkan pada halaman Index.

The screenshot shows a web browser window for 'Students' at 'http://localhost'. The title bar says 'Students'. The main content area displays the 'Contoso University' logo and a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word 'Students' is prominently displayed. A link 'Create New' is visible. A table lists student data with columns: Last Name, First Name, and Enrollment Date. Each row has 'Edit | Details | Delete' links. The last row, which contains the data 'Gao' and 'Erica' for the first two columns and '9/1/2005 12:00:00 AM' for the third, is highlighted with a red border.

Last Name	First Name	Enrollment Date
Alexander	Carson	9/1/2005 12:00:00 AM
Alonso	Meredith	9/1/2002 12:00:00 AM
Anand	Arturo	9/1/2003 12:00:00 AM
Barzdukas	Gytis	9/1/2002 12:00:00 AM
Li	Yan	9/1/2002 12:00:00 AM
Justice	Peggy	9/1/2001 12:00:00 AM
Norman	Laura	9/1/2003 12:00:00 AM
Olivetto	Nino	9/1/2005 12:00:00 AM
Gao	Erica	9/1/2005 12:00:00 AM

## Membuat Halaman Edit

Pada file `Controllers\StudentController.cs`, method `HttpGet Edit` (satu-satunya yang tidak menggunakan atribut `HttpPost`) menggunakan method `Find` untuk mendapatkan entity `Students` yang dipilih, seperti yang sudah dilihat pada method `Details`. Method ini tidak perlu diubah.

Ganti action method `HttpPost Edit` dengan kode berikut untuk menambahkan blok `try-catch`.

```
[HttpPost]  
  
public ActionResult Edit(Student student)  
{  
  
    try  
    {  
  
        if (ModelState.IsValid)  
        {  
  
            db.Entry(student).State = EntityState.Modified;  
  
            db.SaveChanges();  
  
            return RedirectToAction("Index");  
        }  
  
    }  
  
    catch (DataException)  
    {  
  
        //Log the error (add a variable name after DataException)  
  
        ModelState.AddModelError("", "Unable to save changes. Try again, and if the  
problem persists see your system administrator.");  
  
    }  
  
    return View(student);  
}
```

Kode di atas mirip dengan kode yang telah dibuat pada method `HttpPost Create`. Namun pada kode ini tidak menambahkan entity yang dibuat oleh model binder pada set entity, kode ini memberikan tanda pada entity yang menandakan telah terjadi perubahan. Ketika method `SaveChange` dipanggil, penanda `Modified` menyebabkan Entity Framework untuk membuat perintah SQL untuk melakukan update pada

baris di database. Semua kolumn pada baris tersebut akan diupdate, termasuk kolom yang tidak diubah oleh user dan saat proses update tidak mempertimbangkan terjadinya konflik concurrency. (Untuk mempelajari penanganan concurrency akan dibahas pada tutorial Handling Concurrency <http://www.asp.net/entity-framework/tutorials/handling-concurrency-with-the-entity-framework-in-an-asp-net-mvc-application>).

## Entity State dan Method Attach dan SaveChanges

Database context akan melakukan pemeriksaan apakah entity dalam memori sinkron dengan row dalam database, dan informasi tersebut menentukan apa yang terjadi ketika method `SaveChanges` dipanggil. Sebagai contoh, ketika sebuah entity baru dimasukkan ke method `Add`, maka state dari entity tersebut menjadi `Add`. Sehingga ketika method `SaveChanges` dipanggil database context akan memanggil perintah SQL INSERT.

Berikut adalah state yang mungkin dimiliki oleh entity :

- `Added`. Entity belum ada di database. Method `SaveChanges` akan mengeluarkan perintah `INSERT`.
- `Unchanged`. Tidak ada yang perlu dikerjakan saat method `SaveChanges` dipanggil. Ketika operasi membaca sebuah entity dari database maka entity akan memiliki status ini.
- `Modified`. Beberapa atau seluruh property pada entity mengalami perubahan. Method `SaveChanges` akan mengeluarkan perintah `UPDATE`.
- `Deleted`. Entity ditandai untuk dihapus. Method `SaveChanges` akan mengeluarkan perintah `DELETE`.
- `Detached`. Entity tidak sedang ditangani oleh context database.

Pada aplikasi desktop biasanya perubahan state atau status biasanya diset secara otomatis. pada aplikasi tipe ini, ketika dilakukan pembacaan entity dan melakukan perubahan terhadap beberapa nilai property. Hal tersebut menyebabkan state dari entity akan secara otomatis berubah menjadi `Modified`. Dan ketika method `SaveChanges` dipanggil, Entity Framework akan membuat perintah SQL untuk `UPDATE` yang akan mengupdate hanya property-property yang diubah saja.

Sedangkan pada aplikasi web urutan seperti di atas tidak dapat sepenuhnya terjadi, karena instance dari context database yang membaca sebuah entity akan dihapus setelah halaman ditampilkan. Ketika method action `HttpPost Edit` dipanggil, merupakan request baru dan menghasilkan sebuah instance context baru, hal ini mengharuskan developer untuk mengatur state entity menjadi `Modified` secara manual. Kemudian ketika method `SaveChanges` dipanggil, Entity Framework akan mengupdate seluruh kolom pada row database, karena context tidak mengetahui property mana saja yang telah diubah.

Jika diinginkan agar perintah SQL `UPDATE` hanya melakukan update pada field-field yang diubah saja, maka dapat dilakukan dengan menyimpan nilai sebelumnya dengan cara tertentu (misalnya sebagai hidden field) sehingga nilai-nilai tersebut masih ada saat method `HttpPost Edit` dipanggil.

Kemudian dapat dibuat sebuah entity `Student` dengan menggunakan nilai sebelumnya tersebut, dan

panggil method `Attach` dengan entity tersebut, update nilai-nilai entity dengan nilai baru kemudian panggil method `SaveChanges`. Untuk mengetahui lebih banyak tentang hal ini dapat melihat post Add/Attach and Entity States (<http://blogs.msdn.com/b/adonet/archive/2011/01/29/using-dbcontext-in-ef-feature-ctp5-part-4-add-attach-and-entity-states.aspx>) dan Local Data (<http://blogs.msdn.com/b/adonet/archive/2011/02/01/using-dbcontext-in-ef-feature-ctp5-part-7-local-data.aspx>) pada blog team Entity Framework.

File `Views\Student>Edit.cshtml` mempunyai isi yang sama dengan `Create.cshtml` dan tidak perlu ada perubahan pada kode didalamnya.

Jalankan halaman dengan memilih tab **Students** kemudian klik hyperlink **Edit**.

The screenshot shows a web browser window with the URL `http://localhost`. The title bar says "Edit". The main content area displays the Contoso University logo and navigation menu. The menu includes "Home", "About", "Students", "Courses", "Instructors", and "Departments". Below the menu, the word "Edit" is displayed. A form titled "Student" contains fields for "LastName" (value: Alexander), "FirstMidName" (value: Carson), and "EnrollmentDate" (value: 9/1/2005 12:00:00 AM). A "Save" button is present at the bottom of the form. At the bottom of the page, there is a link "Back to List".

[ Log On ]

# Contoso University

Home About Students Courses Instructors Departments

## Edit

**Student**

LastName

FirstMidName

EnrollmentDate

**Save**

[Back to List](#)

Ubah beberapa data kemudian klik **Save** dan akan dilihat perubahannya pada halaman Index.

The screenshot shows a web browser window with the URL `http://localhost`. The title bar says "Students". The main content area is titled "Contoso University" and has a sub-section titled "Students". Below this, there is a link "Create New". A table lists nine students with columns for "Last Name", "First Name", and "Enrollment Date". The "Edit | Details | Delete" links are visible for each row. The "Enrollment Date" for the first student, "Alexander Carson", is highlighted with a red box.

	Last Name	First Name	Enrollment Date
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alexander	Carson	9/1/2011 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alonso	Meredith	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Anand	Arturo	9/1/2003 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Barzdukas	Gytis	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Li	Yan	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Justice	Peggy	9/1/2001 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Norman	Laura	9/1/2003 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Olivetto	Nino	9/1/2005 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Gao	Erica	9/1/2005 12:00:00 AM

## Membuat Halaman Untuk Delete Data

Pada file `Controllers\StudentController.cs`, kode template untuk method `HttpGet Delete` menggunakan method `Find` untuk mendapatkan entity `Student` yang dipilih seperti yang telah dilihat sebelumnya pada method `Detail` dan `Edit`. Tetapi untuk mengimplementasikan sebuah custom

pesan error ketika memanggil method `SaveChanges`, yang dilakukan adalah menambahkan beberapa fungsionalitas pada method tersebut dan penyesuaian pada view.

Seperti yang terlihat pada operasi update dan create, operasi delete membutuhkan dua action method. Method yang dipanggil sebagai tanggapan terhadap request GET untuk menampilkan view yang memberikan pengguna kesempatan untuk memilih untuk menyetujui atau membatalkan operasi delete. Jika pengguna menyetujui maka request POST akan dilakukan. Ketika hal itu terjadi maka method `HttpPost Delete` akan dipanggil dan method yang melakukan operasi delete akan benar-benar dilakukan.

Akan ditambahkan blok `try-catch` pada method `HttpPost Delete` untuk menangani error yang mungkin terjadi saat melakukan update pada database. Jika error terjadi, maka method `HttpPost Delete` akan memanggil method `HttpGet Delete`, melewatkannya parameter yang menunjukkan bahwa telah terjadi error. Method `HttpGet Delete` kemudian akan menampilkan halaman konfirmasi beserta pesan error, yang memberikan kesempatan kepada user untuk melakukan pembatalan atau mengulang lagi operasi tersebut.

Ganti kode method action `HttpGet Delete` dengan kode berikut ini, yang akan mengelola pelaporan error :

```
public ActionResult Delete(int id, bool? saveChangesError)
{
    if (saveChangesError.GetValueOrDefault())
    {
        ViewBag.ErrorMessage = "Unable to save changes. Try again, and if the problem persists see your system administrator.";
    }
    return View(db.Students.Find(id));
}
```

Kode memiliki sebuah optional parameter boolean yang menunjukkan apakah kode dipanggil setelah terjadi kegagalan saat menyimpan perubahan. Parameter tersebut akan bernilai null (`false`) ketika method `HttpGet Delete` dipanggil sebagai tanggapan atas request halaman. Ketika hal itu dipanggil oleh method `HttpPost Delete` sebagai tanggapan saat terjadi error saat update database, parameter akan bernilai `true` dan pesan error akan ditampilkan pada view.

Ganti action method `HttpPost Delete` (yang bernama `DeleteConfirmed` dengan kode berikut ini, yang akan melakukan operasi delete dan akan menangani kesalahan ketika update database.

```

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)

{
    try
    {
        Student student = db.Students.Find(id);

        db.Students.Remove(student);

        db.SaveChanges();
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        return RedirectToAction("Delete",
            new System.Web.Routing.RouteValueDictionary {
                { "id", id },
                { "saveChangesError", true } });
    }
    return RedirectToAction("Index");
}

```

Kode di atas akan menangani entity yang dipilih, kemudian memanggil method `Remove` untuk memberikan status `Deleted` pada entity tersebut. ketika method `SaveChanges` dipanggil maka perintah SQL untuk `DELETE` akan dibuat.

Untuk melakukan peningkatan performance maka dapat dihindari penggunaan query SQL yang tidak diperlukan dengan mengganti kode yang memanggil method `Find` dan `Remove` dengan kode seperti berikut :

```

Student studentToDelete = new Student() { StudentID = id };

db.Entry(studentToDelete).State = EntityState.Deleted;

```

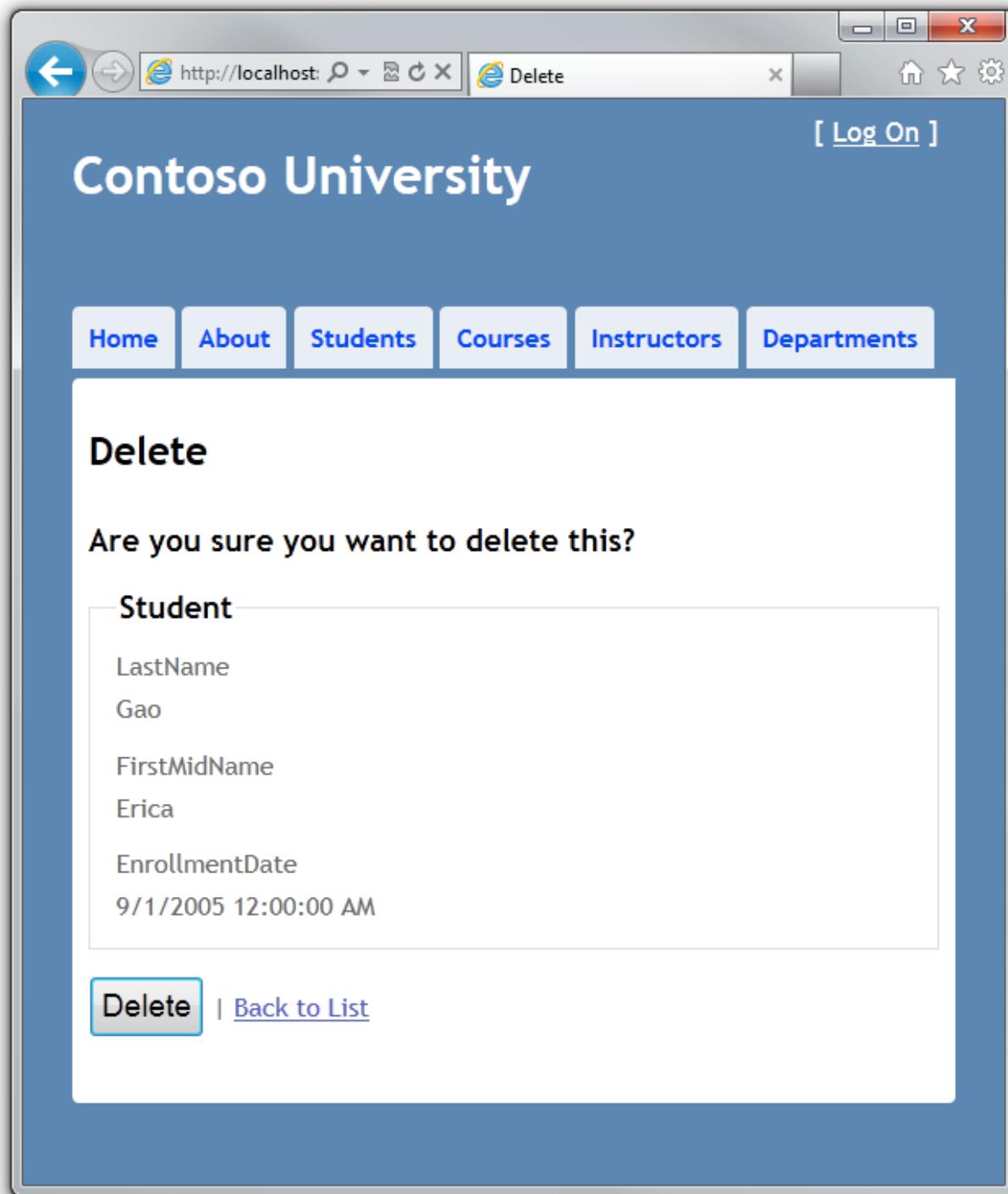
Kode tersebut melakukan instansiasi entity `Student` dengan hanya menggunakan nilai primary key saja dan kemudian memberikan nilai `Deleted` pada state untuk entity tersebut.

Sebagai catatan, method `HttpGet Delete` tidak berfungsi untuk menghapus data. Operasi delete yang menggunakan request GET mempunyai celah keamanan. Untuk mendapatkan informasi lebih lanjut dapat melihat post ASP.NET MVC Tip #46 — Don't use Delete Links because they create Security Holes (<http://stephenwalther.com/blog/archive/2009/01/21/asp.net-mvc-tip-46-donrsquot-use-delete-links-because.aspx>) pada blog milik Stephen Walther.

Pada file `Views\Student\Delete.cshtml` tambahkan kode berikut diantara heading `h2` dan heading `h3`.

```
<p class="error">@ViewBag.ErrorMessage</p>
```

Jalankan halaman dan pilih tab **Students** kemudian klik hyperlink **Delete**.



Klik tombol **Delete**. Halaman Index akan menampilkan data tanpa data student yang telah dihapus.

## Memastikan Koneksi Database Tidak Ditinggalkan Terbuka

Untuk memastikan koneksi ke database telah ditutup maka dapat dipanggil method `Dispose` pada akhir class `StudentController` pada file `StudentController.cs` seperti pada contoh berikut :

```
protected override void Dispose(bool disposing)
{
    db.Dispose();
    base.Dispose(disposing);
}
```

Class base `Controller` telah memiliki implement interface `IDisposable` sehingga akan dengan mudah untuk melakukan override pada method `Dispose (bool)` agar melakukan membuang instance context.

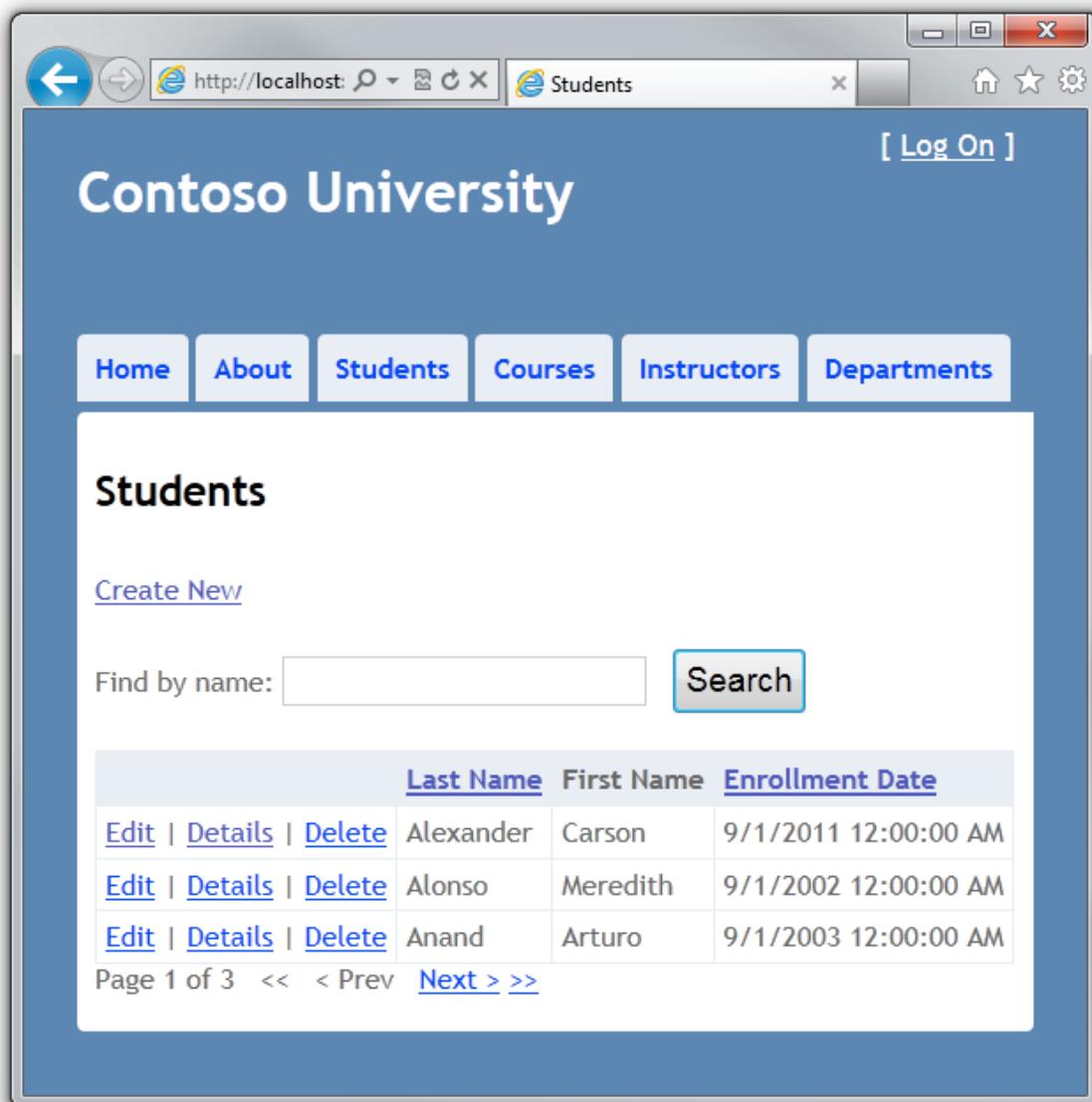
Sekarang telah selesai dibuat halaman-halaman yang menangani operasi CRUD untuk entity `Student`. Pada tutorial berikutnya akan dipaparkan fungsi sorting dan pagging.

Link lain yang berhubungan dengan tutorial Entity Framework dapat ditemukan pada link berikut  
<http://www.asp.net/entity-framework/tutorials/advanced-entity-framework-scenarios-for-an-mvc-web-application>.

# Sorting, Filtering dan Pagging dengan Entity Framework pada Aplikasi ASP.NET MVC

Pada tutorial sebelumnya telah dibuat sekumpulan halaman web untuk operasi CRUD pada entity `Student`. Pada tutorial ini akan dibahas pembuatan fungsi sorting, filtering dan pagging pada halaman Index `Student`. Selain itu juga akan dibuat halaman yang berfungsi untuk melakukan grouping sederhana.

Gambar-gambar di bawah menunjukkan bentuk halaman yang akan dibuat. Pada judul kolom terdapat link yang dapat diklik. Link tersebut bersifat sebagai tombol toggle yang akan memberikan fungsi untuk melakukan sorting secara descending atau ascending.



## Menambahkan Link Sorting Kolom Pada Halaman Index Student

Untuk menambahkan fungsi sorting pada halaman Index Student, perlu dilakukan pengubahan method Index pada controller Student dan penambahan kode pada view.

### Menambahkan Fungsionalitas Sorting pada Index Method

Pada halaman Controllers\StudentController.cs, ganti kode pada method Index dengan kode berikut ini.

```
public ViewResult Index(string sortOrder)
{
    ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "Name desc" : "";
    ViewBag.DateSortParm = sortOrder == "Date" ? "Date desc" : "Date";
    var students = from s in db.Students
                  select s;
    switch (sortOrder)
    {
        case "Name desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "Date desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
}
```

```
    return View(students.ToList());  
}
```

Kode di atas akan menerima parameter `sortOrder` dari query string pada URL yang disediakan oleh ASP.NET MVC sebagai parameter ke method action. Parameter ini akan bertipe string, baik itu untuk Name ataupun Date yang dapat diikuti oleh spasi dan string “desc” untuk memberikan spesifikasi jenis order descending.

Pada saat request halaman Index pertama kali tidak ada query string yang dikirimkan. Data student akan ditampilkan berdasarkan urutan `LastName` secara ascending, hal ini dilakukan sesuai pada bagian default yang tertulis pada bagian kode `switch`. Ketika user mengklik salah satu hyperlink pada judul kolom dengan nilai `sortOrder` yang sesuai dari nilai query string.

Dua variable ViewBag digunakan sehingga view dapat menyesuaikan bagian hyperlink pada judul kolom dengan nilai query string yang sesuai :

```
ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "Name desc" : "";  
ViewBag.DateSortParm = sortOrder == "Date" ? "Date desc" : "Date";
```

Terdapat ternary statement. Yang pertama menyatakan jika parameter `sortOrder` bernilai null atau kosong , maka `ViewBag.NameSortParm` harus di set ke "Nama desc", jika tidak, harus di set ke string kosong.

Terdapat empat kemungkinan tergantung bagaimana data saat ini diurutkan :

- Jika urutan saat ini adalah **Last Name** yang diurut secara ascending, maka link **Last Name** harus ditentukan menjadi **Last Name** descending, dan link **Enrollment Date** harus ditentukan menjadi **Date** ascending.
- Jika urutan saat ini adalah **Last Name** yang diurut secara descending, maka link harus menunjukkan **Last Name** ascending (yaitu , string kosong) dan **Date** ascending.
- Jika urutan saat ini adalah **Date** yang diurut secara ascending, maka link harus **menunjukkan Last Name** ascending dan **Date** descending.
- Jika urutan saat ini adalah **Date** yang diurut descending, maka link harus menunjukkan **Last Name** ascending dan **Date** descending.

Method menggunakan LINQ to Entities untuk melakukan pengurutan kolom. Pada kode terlihat terdapat pembuatan variable `IQueryable` sebelum pernyataan `switch`, kemudian nilai variable

akan diubah di dalam pernyataan `switch` dan terakhir memanggil method `ToList` setelah akhir pernyataan `switch`. Ketika dilakukan pembuatan atau modifikasi variable-variable `IQueryable`, belum ada query yang dikirimkan ke database. Query tidak akan dieksekusi sampai object `IQueryable` dikonversi menjadi collection dengan memanggil method `ToList`. Kode tersebut menghasilkan satu query saja yang tidak akan dieksekusi sampai pernyataan `return View`.

## Menambahkan Hyperlink di Header Kolom pada Student Index View

Pada file `Views\Student\Index.cshtml`, ganti pada bagian judul kolom yang berada diantara element `<tr>` dan `</tr>` dengan kode di bawah ini.

```
<tr>

    <th></th>

    <th>
        @Html.ActionLink("Last Name", "Index", new { sortOrder=ViewBag.NameSortParm })
    </th>

    <th>
        First Name
    </th>

    <th>
        @Html.ActionLink("Enrollment Date", "Index", new { sortOrder=ViewBag.DateSortParm })
    </th>

</tr>
```

Kode ini menggunakan informasi dari property `ViewBag` untuk membuat hyperlink dengan nilai-nilai query string yang sesuai.

Jalankan halaman dan klik kolom judul untuk memastikan pengurutan berfungsi.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title bar says "Students". The page header features the "Contoso University" logo and a "[ Log On ]" link. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Students" and contains a "Create New" link. Below this is a table listing student data with columns: Last Name, First Name, and Enrollment Date. Each row includes edit, details, and delete links.

	Last Name	First Name	Enrollment Date
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alexander	Carson	9/1/2011 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alonso	Meredith	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Anand	Arturo	9/1/2003 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Barzdukas	Gytis	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Gao	Erica	9/1/2005 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Justice	Peggy	9/1/2001 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Li	Yan	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Norman	Laura	9/1/2003 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Olivetto	Nino	9/1/2005 12:00:00 AM

# Menambahkan Kolom Pencarian pada halaman Student Index

Untuk menambahkan fungsi pencarian pada halaman [Index](#), maka perlu ditambahkan text box dan tombol submit pada view dan penambahan pada method [Index](#). Pada text box dapat dimasukkan kata kunci untuk pencarian berdasarkan field nama pertama dan nama akhir.

## Menambahkan Fungsionalitas Filter pada Index Method

Pada file Controllers\StudentController.cs, ganti method Index dengan kode berikut ini.

```
public ViewResult Index(string sortOrder, string searchString)
{
    ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "Name desc" : "";
    ViewBag.DateSortParm = sortOrder == "Date" ? "Date desc" : "Date";
    var students = from s in db.Students
                  select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s =>
s.LastName.ToUpper().Contains(searchString.ToUpper())
                     ||
s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));
    }
    switch (sortOrder)
    {
        case "Name desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "Date desc":
```

```

        students = students.OrderByDescending(s => s.EnrollmentDate);

        break;

    default:

        students = students.OrderBy(s => s.LastName);

        break;

    }

}

return View(students.ToList());
}

```

Ditambahkan parameter searchString pada method Index. Juga ditambahkan klausa where pada perintah LINQ untuk memfilter data student yang memiliki nama awal dan nama akhir sesuai kata kunci yang diberikan. Kata kunci didapat dari text box yang akan ditambahkan kemudian pada view. Pada kode ditambahkan klausa where yang akan dieksekusi jika parameter searchString mempunyai nilai.

```

if (!String.IsNullOrEmpty(searchString))

{
    students = students.Where(s => s.LastName.ToUpper().Contains(searchString.ToUpper())
                           || s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));
}

```

#### Note

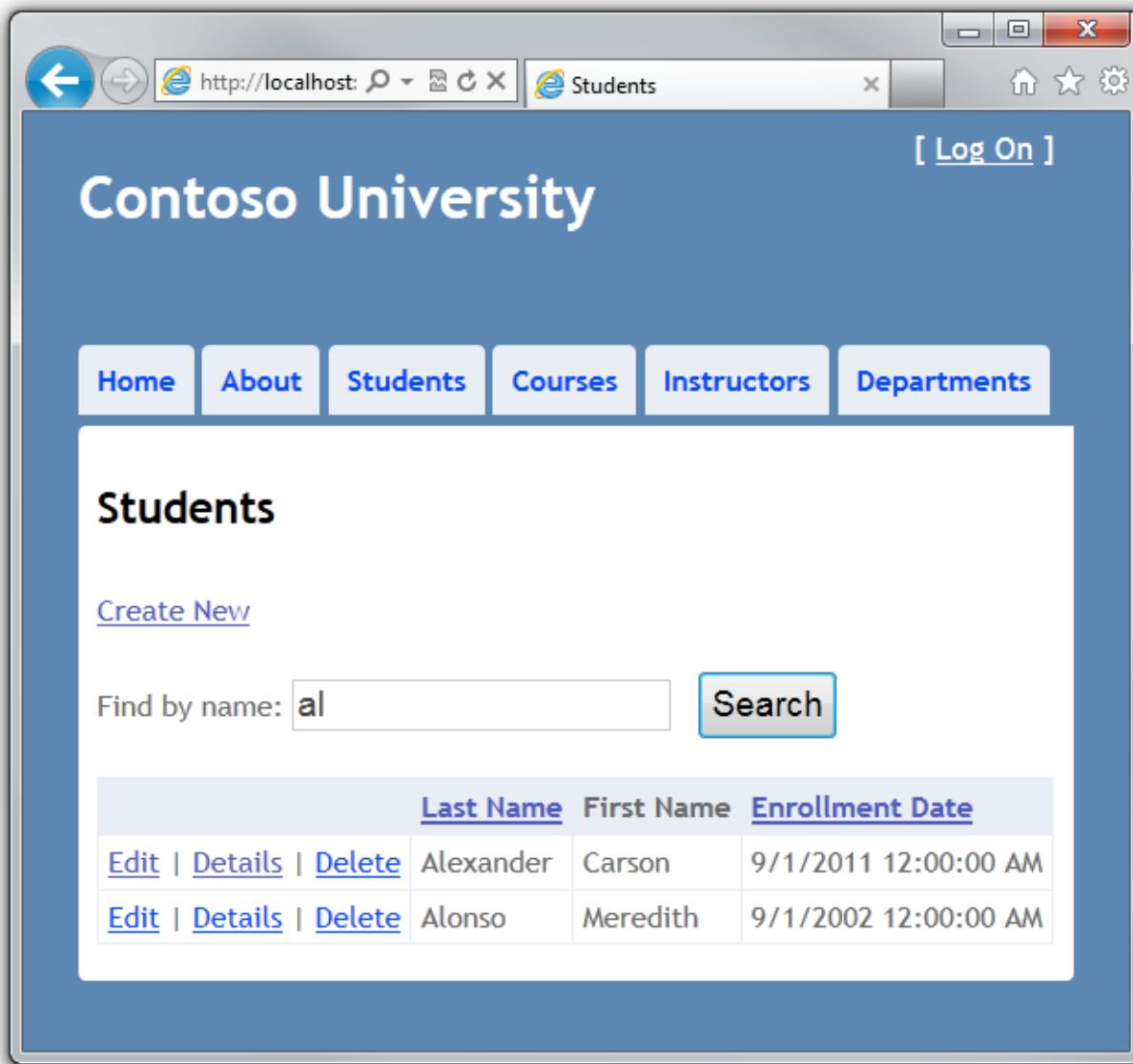
Pada Framework .NET, method `Contains` akan mengembalikan seluruh row pada data ketika dimasukkan nilai string kosong, tetapi pada provider Entity Framework untuk SQL Server Compact 4.0 akan mengembalikan 0 row. Maka pada kode di contoh perlu dipastikan mendapatkan hasil yang sama pada berbagai versi SQL Server. Pada Framework .NET method `Contains` melakukan pembandingan secara case-sensitive, sedangkan pada Entity Framework SQL Server melakukan pembandingan data secara case-insensitive. Maka digunakan pemanggilan method `ToUpper` untuk menghindari perbedaan tersebut, dan akan mengembalikan dalam bentuk koleksi `IEnumerable` bukan objek `IQueryable`. (Ketika memanggil method Contains dengan output `IEnumerable` maka akan didapat implementasi Framework .NET, sedangkan jika output yang didapat adalah `IQueryable` maka didapat implementasi provide database)

## Menambahkan Kolom Pencarian pada Student Index View

Pada file `Views\Student\Index.cshtml` tambahkan text box dan tombol **Search** sebelum tag pembuka `table`.

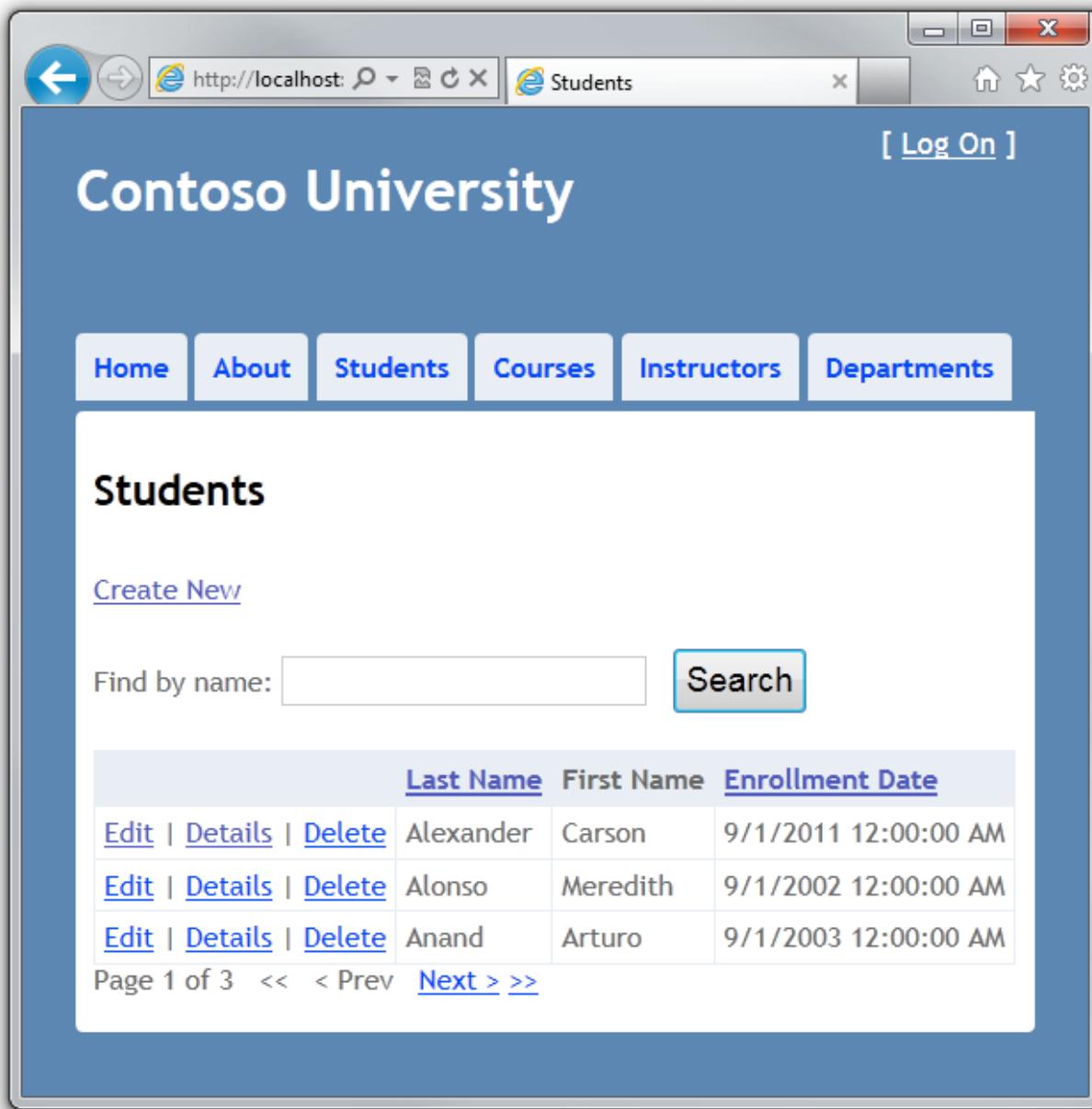
```
@using (Html.BeginForm())
{
    <p>
        Find by name: @Html.TextBox("SearchString")
        <input type="submit" value="Search" /></p>
}
```

Jalankan halaman dan isikan nilai pada text box kemudian tekan tombol Search untuk memastikan filtering berfungsi.



## Menambahkan Pagging pada halaman Student Index

Untuk menambahkan pagging pada halaman Student Index, hal yang pertama dilakukan adalah melakukan install paket PagedList NuGet. Kemudian akan dilakukan penambahan kode pada method Index dan link paging pada view Index. Gambar berikut ini diperlihatkan hasilnya.

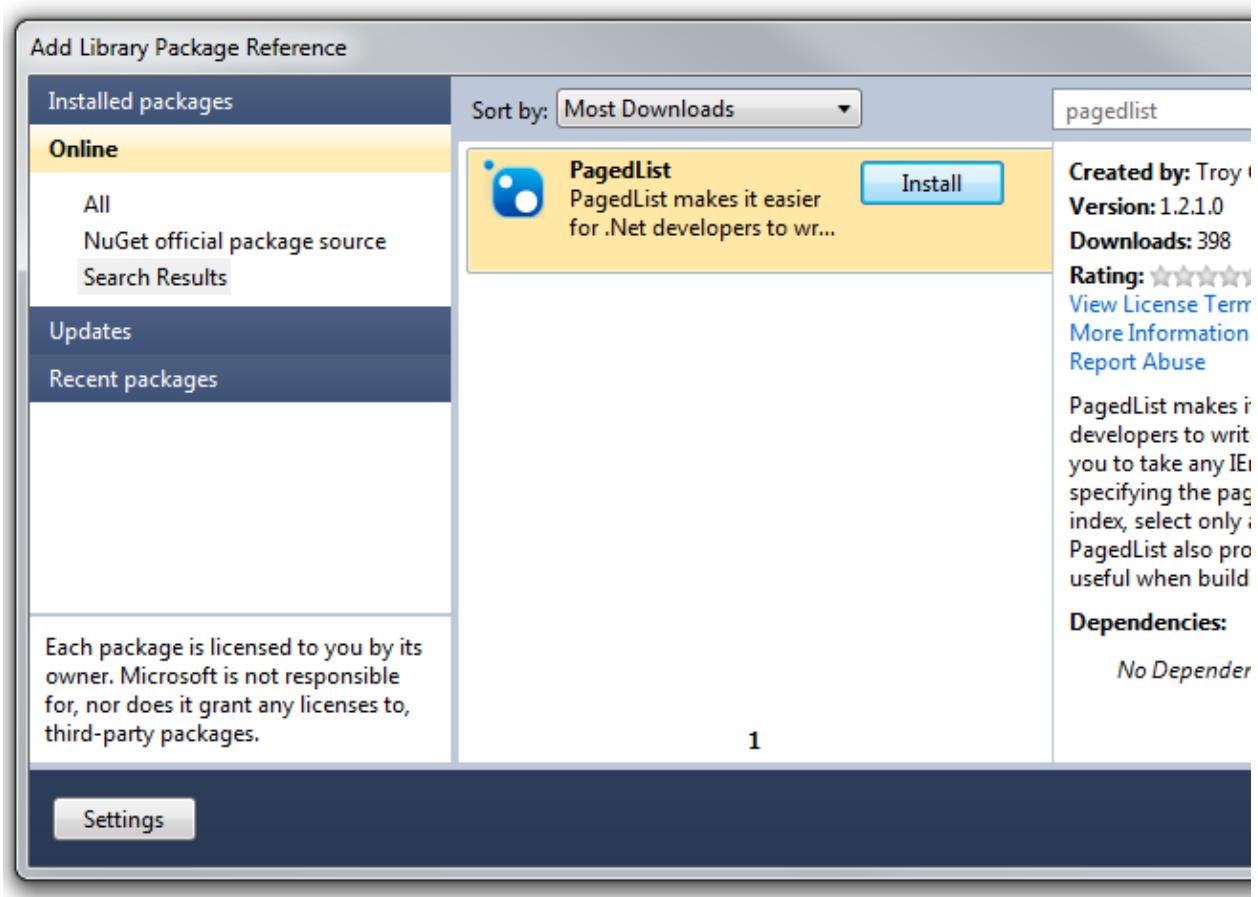


## Installasi Paket PagedList NuGet

Pada paket PageList NuGet akan didapatkan koleksi bertipe `PagedList`. Ketika hasil query disimpan pada koleksi `PagedList` maka akan didapat property dan method untuk keperluan pagging.

Pada **Visual Studio**, pastikan telah dipilih project (bukan solution). pada menu **Tools**, pilih **Library Package Manager** dan pilih **Add Library Package Reference**.

Pada dialog box **Add Library Package Reference**, klik tab **Online** dan masukkan “pagedlist” pada kolom pencarian. Ketika ditemui paket `PagedList`, klik **Install**.



## Menambahkan Fungsionalitas Pagging pada Index Method

Pada file Controllers\StudentController.cs, tambahkan baris `using PagedList;`

```
using PagedList;
```

Ganti method `Index` dengan kode berikut

```
public ViewResult Index(string sortOrder, string currentFilter, string searchString, int? page)
{
    ViewBag.CurrentSort = sortOrder;
    ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "Name desc" : "";
    ViewBag.DateSortParm = sortOrder == "Date" ? "Date desc" : "Date";
}
```

```
if (Request.HttpMethod == "GET")

{
    searchString = currentFilter;
}

else

{
    page = 1;
}

ViewBag.CurrentFilter = searchString;

var students = from s in db.Students
               select s;

if (!String.IsNullOrEmpty(searchString))
{
    students = students.Where(s =>
s.LastName.ToUpper().Contains(searchString.ToUpper()))
                     ||
s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));
}

switch (sortOrder)
{
    case "Name desc":
        students = students.OrderByDescending(s => s.LastName);
        break;
    case "Date":
        students = students.OrderBy(s => s.EnrollmentDate);
        break;
    case "Date desc":
```

```
        students = students.OrderByDescending(s => s.EnrollmentDate);

        break;

    default:

        students = students.OrderBy(s => s.LastName);

        break;

    }

    int pageSize = 3;

    int pageNumber = (page ?? 1);

    return View(students.ToPagedList(pageNumber, pageSize));

}
```

Pada kode ini ditambahkan parameter `page`, parameter untuk menyimpan nilai pengurutan, parameter untuk menyimpan nilai pencarian seperti pada kode berikut :

```
public ViewResult Index(string sortOrder, string currentFilter, string searchString, int?
page)
```

Saat halaman ditampilkan pertama kali atau ketika user belum mengklik link paging maka nilai variabel `page` adalah null. Jika link paging diklik maka variabel `page` akan berisi nomer halaman yang akan ditampilkan.

Property `ViewBag` menangani nilai pengurutan yang sedang dilakukan, hal ini perlu tetap dipertahankan saat link paging diklik, agar saat berpindah halaman pengurutan yang telah dilakukan sebelumnya tetap masih ada.

```
ViewBag.CurrentSort = sortOrder;
```

Property dari `ViewBag` yang lain menangani nilai filter, karena nilai tersebut akan dimasukkan kembali pada text box pada saat halaman berikutnya ditampilkan. Selain itu nilai filter tersebut juga harus diikutkan pada link paging agar proses filtering masih dilakukan saat berpindah halaman. Terakhir, jika nilai string pencarian berubah saat paging dilakukan maka nilai page akan direset menjadi 1, hal ini dilakukan karena proses filter yang baru akan menampilkan data yang berbeda.

```
if (Request.HttpMethod == "GET")
{
    searchString = currentFilter;
}
else
{
    page = 1;
}
ViewBag.CurrentFilter = searchString;
```

Pada akhir method, hasil query data student akan dikonversi menjadi koleksi `PagedList` seperti pada contoh kode berikut :

```
int pageSize = 3;
int pageNumber = (page ?? 1);
return View(students.ToPagedList(pageNumber, pageSize));
```

Method `ToPagedList` berisi parameter inputan dari nilai variabel `pageNumber`. Dua tanda tanya menyatakan operator untuk mendefinisikan nilai default untuk tipe nullable. Ekspresi `(page ?? 1)` berarti nilai `page` adalah 1 jika nilai variable `page` adalah null.

## Menambahkan Link Pagging pada Halaman Student Index

Pada file `Views\Student\Index.cshtml`, ganti kode yang ada dengan kode berikut ini.

```
@model PagedList.IPagedList<ContosoUniversity.Models.Student>

@{
    ViewBag.Title = "Students";
```

```
}
```

```
<h2>Students</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>

@using (Html.BeginForm())
{
    <p>
        Find by name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
        <input type="submit" value="Search" /></p>
    }

<table>
<tr>
    <th></th>
    <th>
        @Html.ActionLink("Last Name", "Index", new { sortOrder=ViewBag.NameSortParm,
currentFilter=ViewBag.CurrentFilter })
    </th>
    <th>
        First Name
    </th>
    <th>
        @Html.ActionLink("Enrollment Date", "Index", new { sortOrder =
ViewBag.DateSortParm, currentFilter = ViewBag.CurrentFilter })
    </th>
</tr>
```

```

@foreach (var item in Model) {

    <tr>

        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.StudentID }) |
            @Html.ActionLink("Details", "Details", new { id=item.StudentID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.StudentID })
        </td>

        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>

        <td>
            @Html.DisplayFor(modelItem => item.FirstMidName)
        </td>

        <td>
            @Html.DisplayFor(modelItem => item.EnrollmentDate)
        </td>

    </tr>
}

</table>

<div>

    Page @(Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber)
    of @Model.PageCount

    @if (Model.HasPreviousPage)
    {
        @Html.ActionLink("<<", "Index", new { page = 1, sortOrder = ViewBag.CurrentSort,
    }

```

```

currentFilter=ViewBag.CurrentFilter  })

    @Html.Raw(" ");

    @Html.ActionLink("< Prev", "Index", new { page = Model.PageNumber - 1, sortOrder
= ViewBag.CurrentSort, currentFilter=ViewBag.CurrentFilter  })

}

else

{

    @:<<

    @Html.Raw(" ");

    @:< Prev

}

@if (Model.HasNextPage)

{

    @Html.ActionLink("Next >", "Index", new { page = Model.PageNumber + 1, sortOrder
= ViewBag.CurrentSort, currentFilter=ViewBag.CurrentFilter  })

    @Html.Raw(" ");

    @Html.ActionLink(">>", "Index", new { page = Model.PageCount, sortOrder =
ViewBag.CurrentSort, currentFilter=ViewBag.CurrentFilter  })

}

else

{

    @:Next >

    @Html.Raw(" ")

    @:>>

}

</div>

```

@model pada baris paling atas menyatakan view akan menggunakan objek PagedList, dan tidak menggunakan objek List lagi.

Sedangkan pada text box akan berisi string pencarian yang tengah dilakukan.

```
Find by name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
```

Link pada kolom judul menggunakan query string untuk mengirimkan nilai string pencarian ke controller sehingga user dapat melakukan hasil pengurutan dan pencarian sekaligus.

```
@Html.ActionLink("Last Name", "Index", new { sortOrder = ViewBag.NameSortParm, currentFilter = ViewBag.CurrentFilter })
```

Pada baris di bagian bawah halaman akan terlihat antarmuka seperti berikut, ini disebabkan oleh kode di atas.

**Page [current page number] of [total number of pages] << < Prev Next > >>**

Simbol << adalah link untuk ke halaman pertama, < Prev adalah link untuk ke halaman sebelumnya dan seterusnya. Jika user saat ini berada di halaman 1 maka link untuk ke halaman sebelumnya akan tidak aktif. Saat user sedang berada di akhir maka link ke halaman berikutnya akan tidak aktif. Setiap link paging akan mengirimkan nilai nomer halaman baru, nilai pengurutan dan nilai pencarian kepada controller dalam bentuk query string.

Jika tidak terdapat halaman yang ditampilkan, maka “Page 0 of 0” akan ditampilkan. (Pada kasus ini nomer halaman akan lebih besar daripada jumlah halaman karena `Model.PageNumber` bernilai 1 sedangkan `Model.PageCount` bernilai 0).

Jalankan halaman maka akan dilihat antarmuka seperti berikut ini.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title bar says "Students". The main content area has a blue header with the text "[ Log On ]" and the "Contoso University" logo. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The "Students" link is highlighted. The main content area has a white background with a title "Students" and a "Create New" link. There is a search bar with a placeholder "Find by name:" and a "Search" button. Below the search bar is a table with three columns: Last Name, First Name, and Enrollment Date. The table contains three rows of student data. At the bottom of the table are links for "Edit", "Details", and "Delete" for each student, followed by the student's name and enrollment date. Below the table are links for "Page 1 of 3", "<< < Prev", and "Next > >".

	Last Name	First Name	Enrollment Date
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alexander	Carson	9/1/2011 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Alonso	Meredith	9/1/2002 12:00:00 AM
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Anand	Arturo	9/1/2003 12:00:00 AM

Klik link paging pada berbagai tipe pengurutan data untuk memastikan paging berfungsi. Kemudian lakukan string pencarian dan lakukan lagi klik paging untuk memastikan kasus seperti ini juga berfungsi.

## **Membuat halaman About yang berfungsi untuk menampilkan data Statistic Student**

Pada halaman About di website Universitas Contoso ini akan ditampilkan berapa banyak siswa yang telah mendaftarkan diri untuk setiap tanggal pendaftaran. Untuk membuat hal ini diperlukan proses pengelompokan dan perhitungan sederhana dan berikut adalah hal-hal yang dilakukan untuk membuat hal tersebut :

- Membuat class view model untuk data yang akan ditampilkan pada view.
- Modifikasi method `About` pada controller `Home`.
- Modifikasi view `About`.

## Membuat View Model

Buat folder `ViewModels`, kemudian buat file `EnrollmentDateGroup.cs` di dalam folder tersebut dan gunakan kode berikut pada file tersebut.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.ViewModels

{
    public class EnrollmentDateGroup
    {
        [DisplayFormat(DataFormatString = "{0:d}")]
        public DateTime? EnrollmentDate { get; set; }

        public int StudentCount { get; set; }
    }
}
```

## Modifikasi Home Controller

Pada file `HomeController.cs` tambahkan baris using seperti berikut :

```
using ContosoUniversity.DAL;
using ContosoUniversity.Models;
using ContosoUniversity.ViewModels;
```

Tambahkan variable berikut untuk database context.

```
private SchoolContext db = new SchoolContext();
```

Ganti method About dengan kode seperti berikut :

```
public ActionResult About()
{
    var data = from student in db.Students
               group student by student.EnrollmentDate into dateGroup
               select new EnrollmentDateGroup()
{
    EnrollmentDate = dateGroup.Key,
    StudentCount = dateGroup.Count()
};

    return View(data);
}
```

Dan tambahkan method Dispose :

```
protected override void Dispose(bool disposing)
{
    db.Dispose();
    base.Dispose(disposing);
}
```

## Modifikasi About View

Ganti kode pada *Views\Home\About.cshtml* dengan kode berikut ini :

```
@model IEnumerable<ContosoUniversity.ViewModels.EnrollmentDateGroup>
```

```
@{

    ViewBag.Title = "Student Body Statistics";

}

<h2>Student Body Statistics</h2>



| Enrollment Date                                           | Students                        |
|-----------------------------------------------------------|---------------------------------|
| <code>@String.Format("{0:d}", item.EnrollmentDate)</code> | <code>@item.StudentCount</code> |



@{
    foreach (var item in Model) {
        <tr>
            <td>
                @String.Format("{0:d}", item.EnrollmentDate)
            </td>
            <td>
                @item.StudentCount
            </td>
        </tr>
    }
}
```

Jalankan halaman dan akan dapat dilihat hasil seperti berikut ini.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The page title is "About Us". At the top, there is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the main content area has a heading "Student Body Statistics". Underneath the heading is a table with two columns: "Enrollment Date" and "Students". The table data is as follows:

Enrollment Date	Students
9/1/2001	1
9/1/2002	3
9/1/2003	2
9/1/2005	2
9/1/2011	1

Dari paparan di atas maka telah dapat dilihat bagaimana cara untuk membuat data model dan membuat operasi CRUD, membuat fungsi untuk pengurutun, filter, paging dan grouping. Pada tutorial berikut akan dimulai bahasan tentang topik lanjutan tentang bahasan lebih lanjut tentang data model.

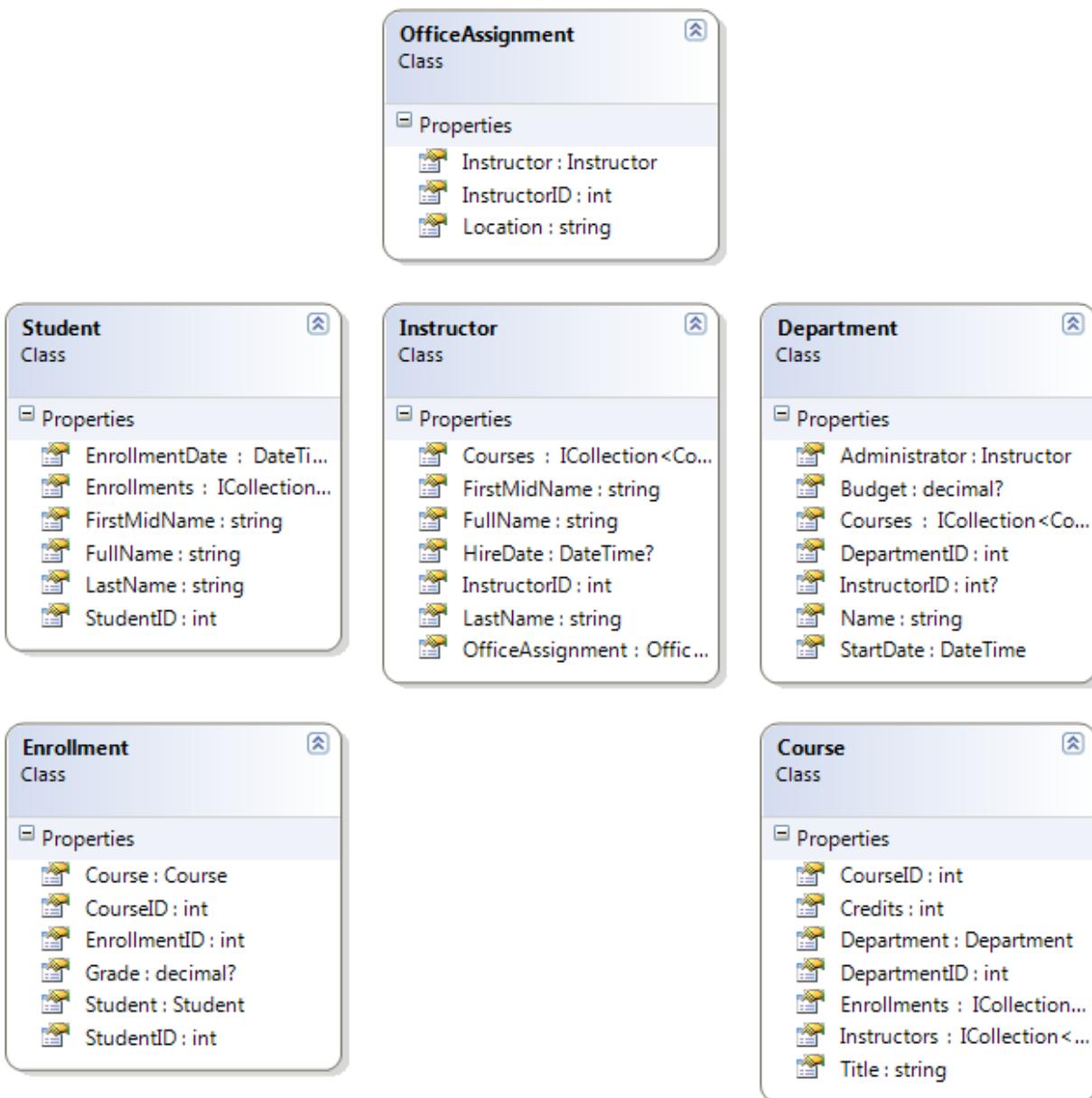
Link lain yang berhubungan dengan tutorial Entity Framework dapat ditemukan pada link berikut  
<http://www.asp.net/entity-framework/tutorials/advanced-entity-framework-scenarios-for-an-mvc-web-application>.

# Membuat Data Model Yang Lebih Rumit untuk Aplikasi ASP.NET MVC

Pada tutorial sebelumnya kita bekerja dengan data model yang sederhana yang terdiri atas tiga entity.

Pada tutorial ini akan ditambahkan beberapa entity dan relasi serta atribut data annotation untuk mengontrol prilaku dari class-class model.

Ilustrasi berikut ini menggambarkan class-class model dari data model yang lengkap :



# Menggunakan Attribute untuk Formatting Control, Validasi dan Mapping Database

Pada bagian ini kita akan melihat contoh atribut-atribut yang dapat kita tambahkan pada class-class model untuk formating, validasi, dan mapping database. Pada bagian berikut, kita akan membuat `School` data model yang lengkap dengan menambahkan atribut yang pada class yang telah dibuat dan membuat class baru untuk tipe entity yang tersisa dalam model.

## Attribute DisplayFormat

Untuk tanggal pendaftaran siswa yang diwakili oleh property `EnrollmentDate` biasanya pada halaman web ditampilkan waktu dan tanggal. Dengan menggunakan atribut data annotation maka kita dapat menambahkan kode untuk mengubah format keluaran sesuai keinginan yang akan ditampilkan pada seluruh halaman web. Berikut adalah contoh kode penambahan atribut tersebut pada property `EnrollmentDate` yang terdapat pada class `Student`.

Pada file `Model\Student.cs` tambahkan statement using untuk penggunaan namespace `System.ComponentModel.DataAnnotations` and tambahkan atribut `DisplayFormat` pada property `EnrollmentDate`. Berikut adalah kode lengkapnya :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int StudentID { get; set; }

        public string LastName { get; set; }

        public string FirstMidName { get; set; }

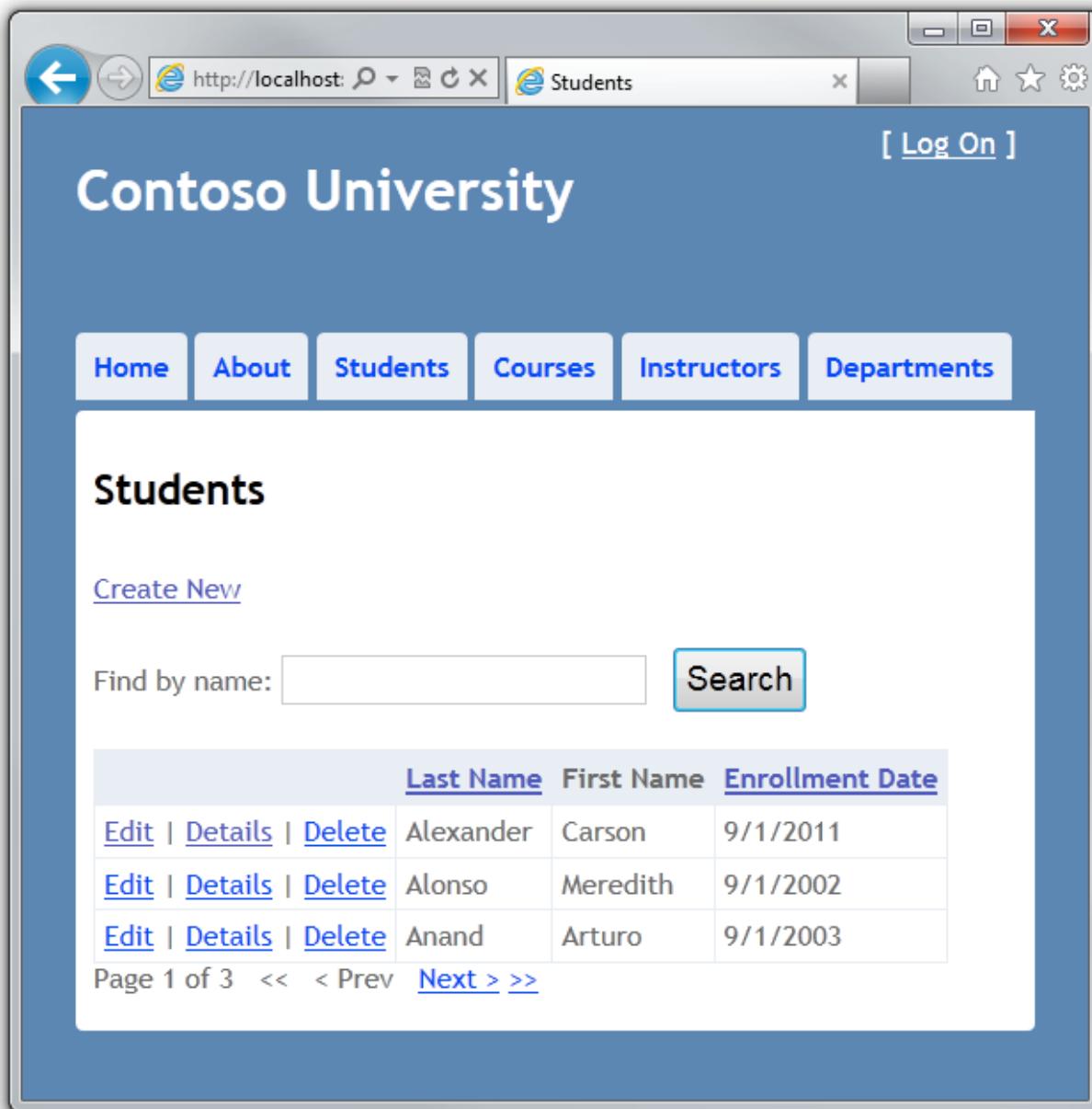
        [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

```
    }  
}
```

Dapat dilihat pada contoh di atas, format string yang digunakan di atas hanya akan menampilkan tanggal pendek saja. Setting `ApplyFormatInEditMode` dengan nilai true, membuat format tersebut akan digunakan pada textbox saat mode edit.

Jalankan halaman Index Student dan kita akan dapat melihat waktu tidak akan ditampilkan lagi pada tanggal penerimaan, yang ditampilkan hanya tanggal saja. Format tanggal yang sama akan dapat dilihat pada halaman Student yang lain.



## Attribute MaxLength

Dengan menggunakan atribut dapat ditentukan aturan untuk validasi data dan pesan yang ingin ditampilkan. Sebagai contoh sebuah input nama tidak boleh diisi dengan lebih dari 50 karakter. Untuk menambahkan limitasi seperti itu maka dapat ditambahkan atribut `Range` pada property `LastName` dan `FirstMidName` seperti contoh kode berikut ini :

```
using System;  
  
using System.Collections.Generic;
```

```

using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int StudentID { get; set; }

        [MaxLength(50)]
        public string LastName { get; set; }

        [MaxLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
        public string FirstMidName { get; set; }

        [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
        public DateTime EnrollmentDate { get; set; }

        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }
}

```

Jika pengguna memasukkan nama akhir lebih besar dari 50 karakter maka akan ditampilkan default pesan kesalahan. Sedangkan jika nama awal yang dimasukkan terlalu panjang maka pesan kesalahan yang akan ditampilkan sesuai dengan nilai ErrorMessage yang telah ditentukan pada kode di atas.

Sekarang, jalankan halaman Create kemudian masukkan nama awal dan akhir dengan jumlah karakter lebih besar dari 50 karakter dan tekan tombol **Create**, maka kita akan melihat pesan kesalahan seperti pada gambar berikut ini.

[ Log On ]

# Contoso University

Home About Students Courses Instructors Departments

## Create

**Student**

Last Name

A very long name longer! The field LastName must be a string or array type with a maximum length of '50'.

First Mid Name

Another very long name First name cannot be longer than 50 characters.

Enrollment Date

1/1/2011

**Create**

[Back to List](#)

Dianjurkan untuk selalu menentukan panjang maksimum untuk setiap property yang bertipe string. Jika hal ini tidak dilakukan maka saat database dibuat (jika kita menggunakan konsep Code First), kolom-

kolom akan memiliki panjang maksimal yang dibolehkan untuk string. Hal ini membuat struktur tabel database menjadi tidak efisien.

## Attribute Column

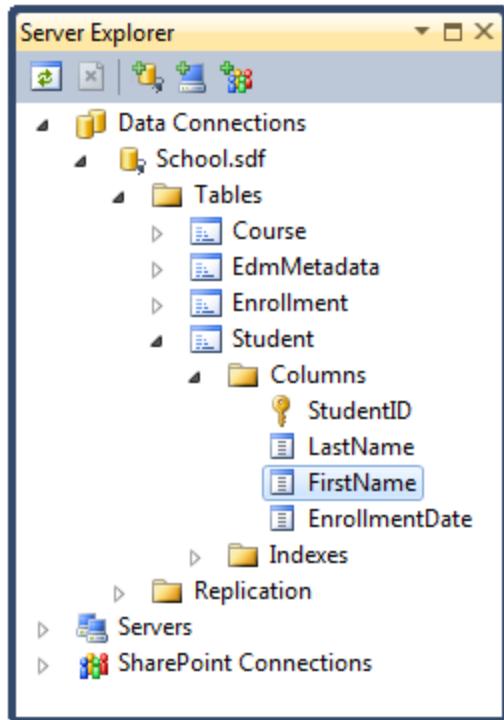
Kita dapat menentukan atribut untuk melakukan kontrol pada class dan property yang dipetakan ke database. Misal kita menggunakan nama `FirstMidName` sebagai field untuk menyimpan nama awal dan nama tengah, sedangkan nama kolom tabel yang digunakan adalah `FirstName`, maka untuk memenuhi kebutuhan ini dapat digunakan atribut `Column`.

Atribut `Column` ditentukan saat database dibuat, kolom `FirstName` yang terdapat pada tabel `Student` akan dipetakan ke property `FirstMidName`. Dengan kata lain, jika digunakan kode `Student.FirstMidName` maka artinya nilainya didapat dari atau akan mengupdate kolom `FirstName` dari tabel `Student`. (Jika kita tidak melakukan nama kolom yang pada suatu property, maka diasumsikan nama kolom sama dengan nama property tersebut.)

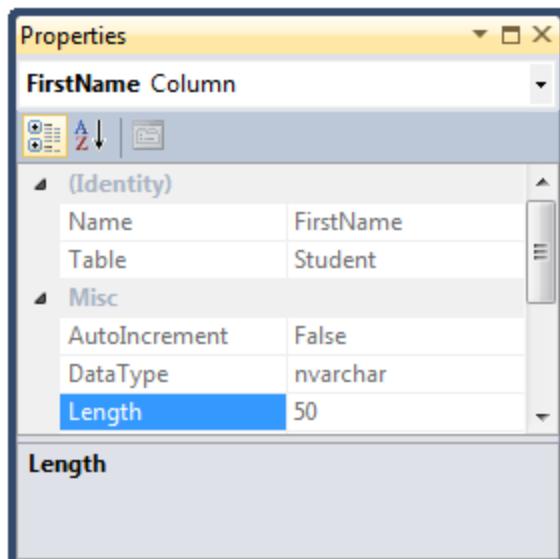
Berikut adalah contoh penggunaan atribut `Column` pada property `FirstMidName` :

```
[Column("FirstName")]
public string FirstMidName { get; set; }
```

Jalankan halaman Index Student dan kita belum melihat perubahan. (Kita dapat hanya menjalankan site dan melihat halaman home, kita harus halaman Index Student agar terjadi akses ke database yang secara otomatis akan terjadi proses penghapus dan pembuatan ulang tabel.) Jika kita membuka database dengan menggunakan **Server Explorer**, kita dapat melihat terdapat kolom `FirstName` pada tabel `Student`.



Pada jendela Properties dapat dilihat field tersebut mempunyai definisi panjang 50 karakter sesuai dengan atribut **MaxLength** yang telah ditambahkan sebelumnya.



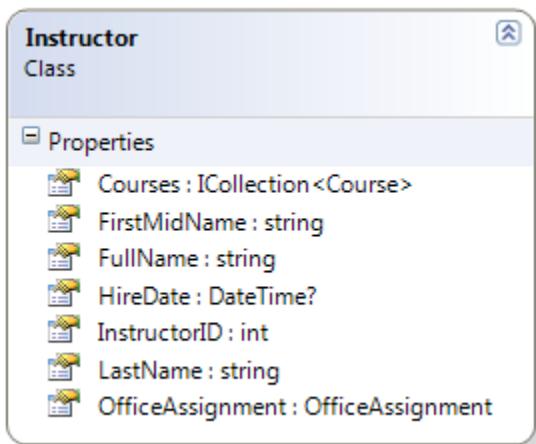
Pada kasus kebanyakan, kita dapat membuat perubahan pada pemetaan dengan menggunakan method calls yang akan dapat dilihat nanti pada tutorial ini.

Pada bagian ini kita akan membuat beberapa atribut data annotation pada **School** data model. Pada bagian ini akan dibuat class untuk entity atau modifikasi sebuah class yang telah dibuat pada tutorial pertama.

Note :

Jika kita mencoba untuk melakukan kompilasi sebelum menyelesaikan pembuatan seluruh class entity maka kita mungkin akan mendapatkan pesan error.

## Membuat Instructor Entity



Buat file *Models/Instructors.cs*, dengan menggunakan kode di bawah ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Instructor
    {
        public Int32 InstructorID { get; set; }

        [Required(ErrorMessage = "Last name is required.")]
    }
}
```

```
[Display(Name = "Last Name")]
[MaxLength(50)]
public string LastName { get; set; }

[Required(ErrorMessage = "First name is required.")]
[Column("FirstName")]
[Display(Name = "First Name")]
[MaxLength(50)]
public string FirstMidName { get; set; }

[DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
[Required(ErrorMessage = "Hire date is required.")]
[Display(Name = "Hire Date")]
public DateTime? HireDate { get; set; }

public string FullName
{
    get
    {
        return LastName + ", " + FirstMidName;
    }
}

public virtual ICollection<Course> Courses { get; set; }
public virtual OfficeAssignment OfficeAssignment { get; set; }
}
```

Dapat dilihat terdapat beberapa property yang sama diantara entity `Student` dan `Instructor`. Pada tutorial pada seri ini yaitu Implementing Inheritance (<http://www.asp.net/entity-framework/tutorials/implementing-inheritance-with-the-entity-framework-in-an-asp-net-mvc-application>), kita akan melakukan refactor dengan menggunakan *inheritance* untuk menghilangkan *redundancy*.

## Attribute Required dan Attribute Display

Pada kode di bawah ini dapat dilihat bahwa atribut pada property `LastName` adalah field yang harus diisi, dan akan menampilkan caption “Last Name” (walaupun nama dari property-nya adalah `LastName` tanpa ada spasi), dan nilai dari property ini tidak boleh dari 50 karakter.

```
[Required(ErrorMessage = "Last name is required.")]
[Display(Name = "Last Name")]
[MaxLength(50)]

public string LastName { get; set; }
```

## Calculated Property, FullName

`FullName` adalah property gabungan antara dua buat property. Property ini hanya memiliki accessor `get` dan tidak ada kolom `FullName` yang akan dibuat pada database.

```
public string FullName
{
    get
    {
        return LastName + ", " + FirstMidName;
    }
}
```

## Navigation Property, Courses dan OfficeAssignment

Property `Courses` dan `OfficeAssignment` adalah property navigation. Seperti yang telah dijelaskan sebelumnya, keduanya biasanya didefinisikan sebagai virtual sehingga dapat menggunakan

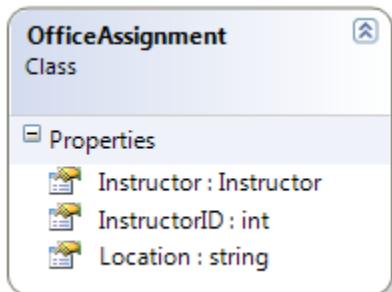
fitur dari Entity Framework yaitu *lazy loading*. Sebagai tambahan, jika property navigation dapat menangani beberapa entity maka tipe yang harus digunakan adalah `ICollection`.

Pada kasus ini, seorang instruktur dapat mengajar beberapa course, maka `Courses` didefinisikan sebagai koleksi dari entity `Course`. Seorang instruktur hanya memungkinkan memiliki sebuah kantor, maka `OfficeAssignment` didefinisikan sebagai sebuah entity `OfficeAssignment` (nilainya mungkin adalah null jika tidak ada kantor yang digunakan oleh instruktur).

```
public virtual ICollection<Course> Courses { get; set; }

public virtual OfficeAssignment OfficeAssignment { get; set; }
```

## Membuat Entity `OfficeAssignment`



Buat file `Models/OfficeAssignment.cs` dan gunakan kode berikut ini sebagai isi dari file tersebut :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class OfficeAssignment
    {
        [Key]
        public int InstructorID { get; set; }
```

```
[MaxLength(50)]  
[Display(Name = "Office Location")]  
public string Location { get; set; }  
  
public virtual Instructor Instructor { get; set; }  
}  
}
```

## Attribute Key

Terdapat relasi one-to-zero-or-one antara entity `Instructor` dan `OfficeAssignment`. Relasi sebuah kantor hanya terjadi kepada instruktur, dimana primary key akan menjadi foreign key pada entity `Instructor`. Tetapi Entity Framework tidak dapat secara otomatis mengenali `InstructorID` sebagai primary key dari entity karena nama property tersebut tidak mengikuti aturan yaitu namanya harus `ID` atau `classnameID`. Maka digunakanlah atribut `Key` untuk menentukan hal tersebut.

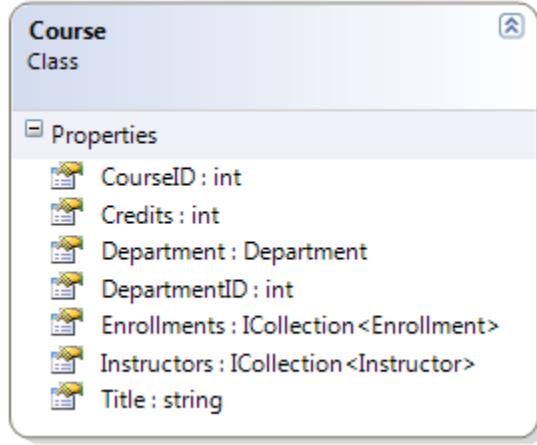
```
[Key]  
public int InstructorID { get; set; }
```

Kita juga dapat menggunakan atribut `Key` jika property untuk primary key mempunyai nama yang tidak mengikuti aturan `ID` dan `classnameID`.

## Instructor Navigation Property

Entity `Instructor` mempunyai property navigation `OfficeAssignment` yang boleh memiliki nilai `null` (karena instruktur memungkinkan untuk tidak memiliki kantor), dan entity `OfficeAssignment` tidak memiliki property navigation `Instructor` yang tidak boleh memiliki nilai `null` (karena sebuah kantor tidak akan ada jika tidak memiliki instruktur). Ketika entity `Instructor` memiliki relasi dengan entity `OfficeAssignment`, setiap entity akan mempunyai referensi ke masing-masing property `navigation`-nya.

## Modifikasi Course Entity



Pada file Modes/Course.cs, ganti kode yang ada dengan kode berikut ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Display(Name = "Number")]
        public int CourseID { get; set; }

        [Required(ErrorMessage = "Title is required.")]
        [MaxLength(50)]
        public string Title { get; set; }

        [Required(ErrorMessage = "Number of credits is required.")]
    }
}
```

```

[Range(0, 5, ErrorMessage = "Number of credits must be between 0 and 5.")]

public int Credits { get; set; }

[Display(Name = "Department")]

public int DepartmentID { get; set; }

public virtual Department Department { get; set; }

public virtual ICollection<Enrollment> Enrollments { get; set; }

public virtual ICollection<Instructor> Instructors { get; set; }

}

}

```

## Attribute DatabaseGenerated

Atribut `DatabaseGenerated` dengan parameter `None` pada property `CourseID` menetapkan bahwa nilai dari primary key akan diberikan oleh pengguna, bukan hasil database.

```

[DatabaseGenerated(DatabaseGeneratedOption.None)]

[Display(Name = "Number")]

public int CourseID { get; set; }

```

Biasanya Entity Framework mengasumsikan nilai-nilai primary key akan digenerate oleh database. Skenario tersebut biasanya yang sering kita gunakan. Tetapi untuk entity `Course`, user akan menentukan sendiri nilai tersebut, misal seri 1000 akan digunakan untuk nilai suatu departemen, dan seri 2000 akan digunakan untuk departemen yang lain.

## Foreign Key dan Property Navigation

Property foreign key dan property navigation pada entity `Course` mencerminkan relasi seperti berikut :

- Sebuah course yang ditentukan pada suatu departemen, maka akan terdapat foreign key `DepartemenID` dan property navigation `Departement` :

```
public int DepartmentID { get; set; }

public virtual Department Department { get; set; }
```

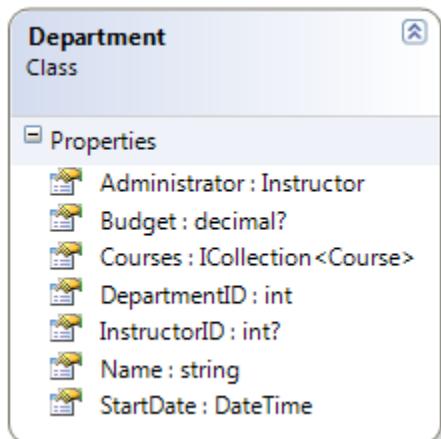
- Course dapat memiliki banyak student sehingga terdapat property navigation Enrollment :

```
public virtual ICollection<Enrollment> Enrollments { get; set; }
```

- Course dapat diajarkan oleh beberapa instruktur, maka akan terdapat property navigation Instructors :

```
public virtual ICollection<Instructor> Instructors { get; set; }
```

## Membuat Entity Department



Buat file `Models\Departement.cs`, gunakan kode berikut untuk isi file tersebut :

```
using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models

{
```

```

public class Department
{
    public int DepartmentID { get; set; }

    [Required(ErrorMessage = "Department name is required.")]
    [MaxLength(50)]
    public string Name { get; set; }

    [DisplayFormat(DataFormatString = "{0:c}")]
    [Required(ErrorMessage = "Budget is required.")]
    [Column(TypeName = "money")]
    public decimal? Budget { get; set; }

    [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
    [Required(ErrorMessage = "Start date is required.")]
    public DateTime StartDate { get; set; }

    [Display(Name = "Administrator")]
    public int? InstructorID { get; set; }

    public virtual Instructor Administrator { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}

```

## Attribute Column

Sebelumnya kita telah menggunakan atribut `Column` untuk mengubah pemetaan property dengan kolom. Pada kode di atas dapat dilihat penggunaan atribut `Column` untuk menentukan tipe kolom pada database :

```
[Column(TypeName = "money")]

public decimal? Budget { get; set; }
```

Hal tersebut biasanya tidak diperlukan, karena Entity Framework telah menentukan tipe data SQL Server yang sesuai tipe CLR yang telah kita tentukan untuk property tersebut. tipe CLR `decimal` biasanya akan dipetakan dengan tipe SQL Server `decimal`. Tetapi pada kasus ini karena kolumn akan menyimpan data dengan nilai mata uang maka akan lebih sesuai jika menggunakan tipe data `money`.

## Foreign Key dan Property Navigation

Property foreign key dan navigation mencerminkan relasi seperti berikut :

- Sebuah departemen mungkin memiliki atau tidak memiliki seorang administrator, dan administrator adalah pasti seorang instruktur. Oleh karena itu property `InstructorID` disertakan sebagai foreign key bagi entity `Instructor`, dan penggunaan tanda tanya (?) setelah tipe int dimaksudkan untuk menyatakan property tersebut memungkinkan untuk memiliki nilai null. Property navigation diberi nama `Administrator` tetapi mempunyai tipe dari entity `Instructor` :

```
public int? InstructorID { get; set; }

public virtual Instructor Administrator { get; set; }
```

- Sebuah departemen memungkinkan memiliki lebih dari satu course, maka akan terdapat property navigation `Courses` :

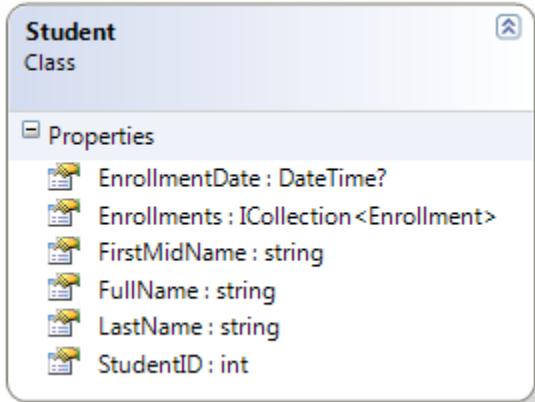
```
public virtual ICollection<Course> Courses { get; set; }
```

### Note :

Berdasarkan aturan/perjanjian, Entity Framework memungkinkan cascade delete untuk foreign key yang mempunyai nilai bukan null dan untuk relasi many-to-many. Hal ini dapat menghasilkan aturan circular cascade delete yang akan menyebabkan keluarnya exception jika initializer code berjalan. Sebagai contoh, jika kita tidak menentukan bahwa property `Departement.InstructorID` dapat mempunyai nilai

null (nullable) maka kita akan dapat melihat exception dengan pesan “The referential relationship will result in a cyclical reference that's not allowed.” jika initializer code berjalan.

## Modifikasi Entity Student



Pada file *Models\Student.cs*, ganti kode yang ada dengan kode di bawah ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int StudentID { get; set; }

        [Required(ErrorMessage = "Last name is required.")]
        [Display(Name = "Last Name")]
        [MaxLength(50)]
        public string LastName { get; set; }
    }
}
```

```

[Required(ErrorMessage = "First name is required.")]
[Column("FirstName")]
[Display(Name = "First Name")]
[MaxLength(50)]
public string FirstMidName { get; set; }

[Required(ErrorMessage = "Enrollment date is required.")]
[DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
[Display(Name = "Enrollment Date")]
public DateTime? EnrollmentDate { get; set; }

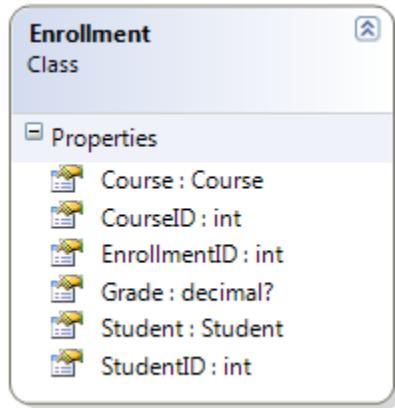
public string FullName
{
    get
    {
        return LastName + ", " + FirstMidName;
    }
}

public virtual ICollection<Enrollment> Enrollments { get; set; }
}

```

Pada kode di atas juga ditambahkan atribut-atribut seperti yang telah kita lihat pada class-class sebelumnya.

## Modifikasi Entity Enrollment



Pada *Models\Enrollment.cs* ganti kode yang ada dengan kode berikut ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Enrollment
    {
        public int EnrollmentID { get; set; }

        public int CourseID { get; set; }

        public int StudentID { get; set; }

        [DisplayFormat(DataFormatString = "{0:#.#}", ApplyFormatInEditMode = true,
        NullDisplayText = "No grade")]
        public decimal? Grade { get; set; }
    }
}
```

```
    public virtual Course Course { get; set; }

    public virtual Student Student { get; set; }

}

}
```

## Foreign Key dan Property Navigation

Property foreign key dan navigation mencerminkan relasi seperti berikut :

- Sebuah record enrollment adalah untuk sebuah course, maka berikut adalah kode untuk foreign key `CourseID` dan property navigation `Course` :

```
public int CourseID { get; set; }

public virtual Course Course { get; set; }
```

- Sebuah record enrollment adalah untuk seorang student, maka berikut adalah kode untuk foreign key `StudentID` dan property navigation `Student` :

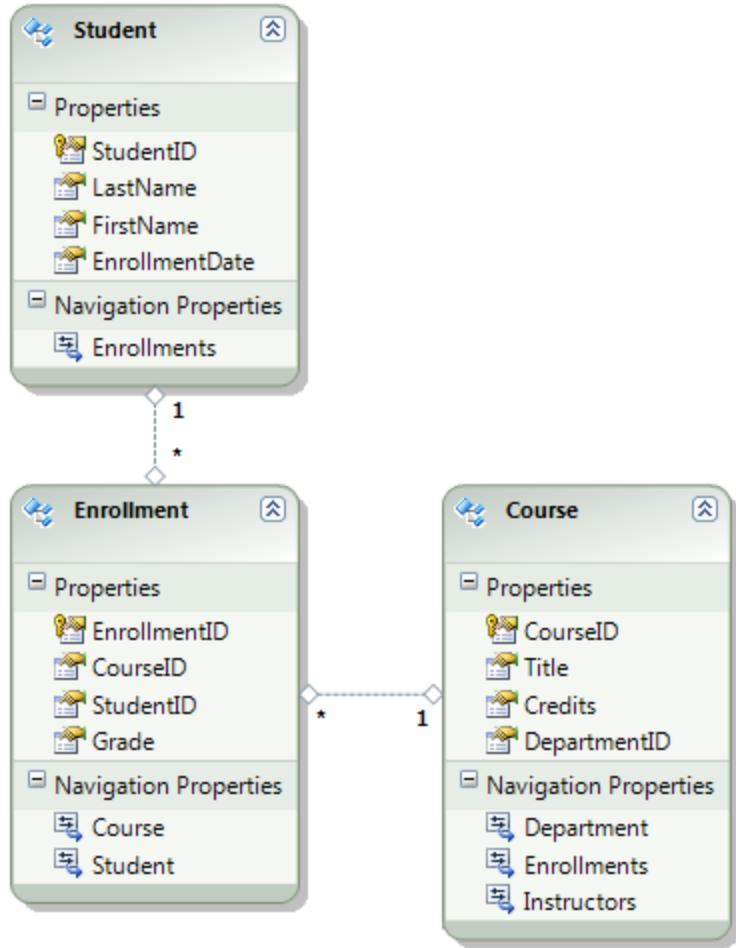
```
public int StudentID { get; set; }

public virtual Student Student { get; set; }
```

## Relasi Many-to-Many

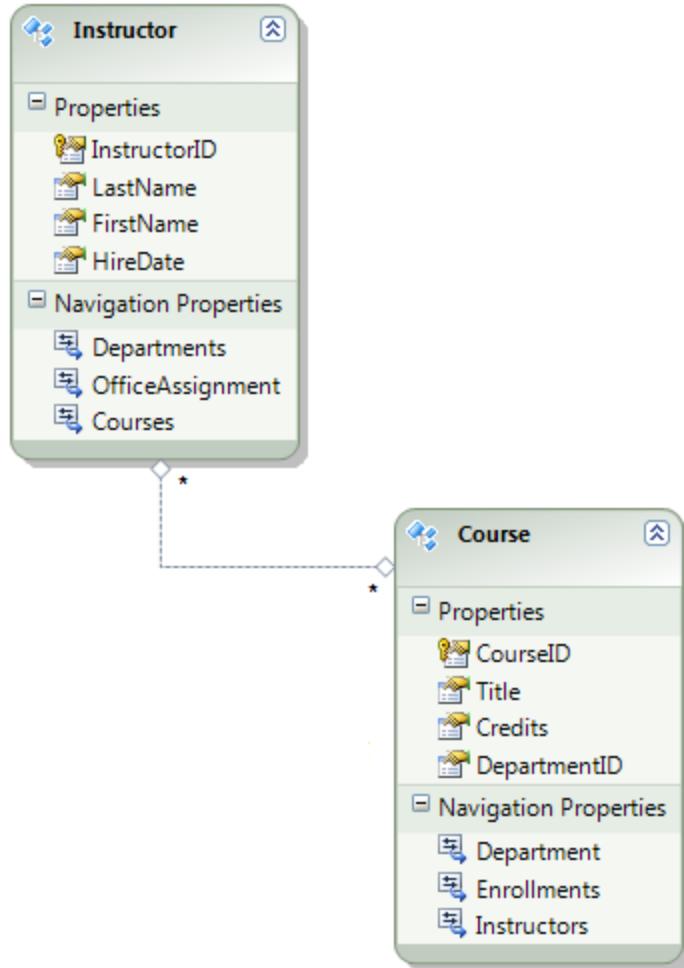
Terdapat relasi many-to-many antara entity `Student` dengan `Course` dan entity `Enrollment` sesuai dengan tabel dengan relasi many-to-many yang terdapat pada database. Ini artinya tabel `Enrollment` terdapat data tambahan selain foreign key (pada kasus ini terdapat data tambahan primary key dan property `Grade`).

Ilustrasi berikut ini memperlihatkan relasi yang terdapat pada entity diagram. (diagram ini adalah hasil dari tool designer Entity Framework, cara pembuatan diagram ini bukan merupakan bagian dari tutorial ini.)

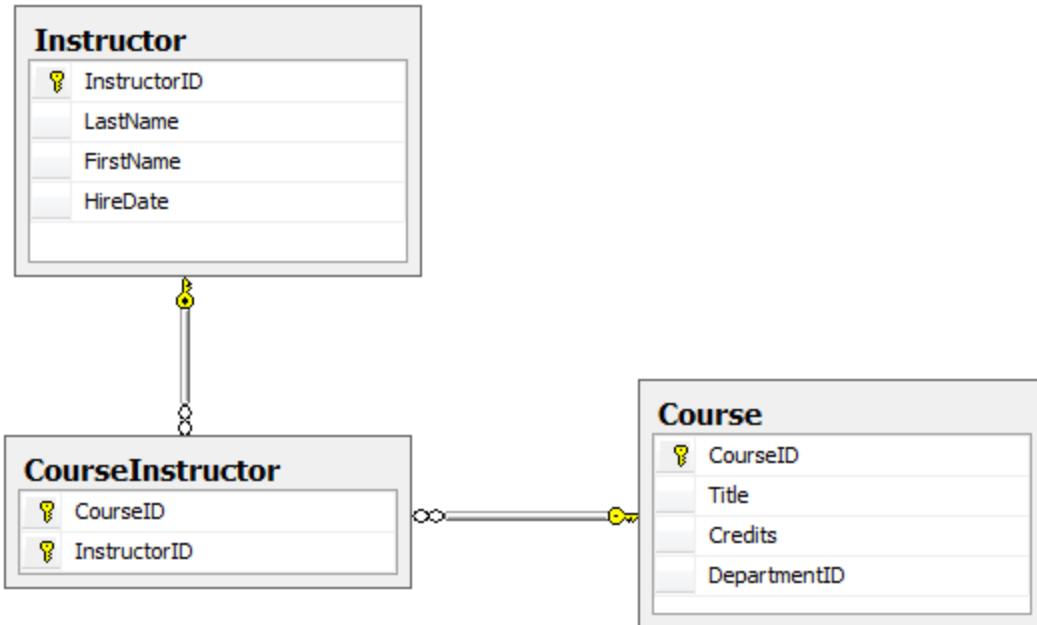


Setiap garis relasi yang mempunyai tanda 1 pada akhir garis dan tanda asterik (\*) pada ujung lainnya menyatakan relasi one-to-many.

Jika tabel Enrollment tidak memiliki informasi grade, maka cukup berisi dua foreign key yaitu `CourseID` dan `StudentID`. Pada kasus tersebut maka hal itu sesuai dengan tabel many-to-many join tanpa payload pada database, dan kita tidak perlu membuat class model untuk hal tersebut sama sekali. Entity Instructor dan Course memiliki relasi many-to-many seperti tersebut di atas maka seperti yang kita lihat pada diagram di bawah, tidak terdapat class entity antara kedua entity tersebut :



Maka sebuah tabel join diperlukan dalam database, yang ditunjukkan dalam diagram database berikut ini :



Entity Framework secara otomatis akan membuat tabel `CourseInstructor`, dan kita dapat membaca dan update data secara langsung dengan menggunakan property navigation `Instructor.Courses` dan `Course.Instructors`.

## Attribute `DisplayFormat`

Atribut `DisplayFormat` pada property `Grade` mendefinisikan bagaimana format keluaran yang akan ditampilkan :

```

[DisplayFormat(DataFormatString = "{0:#.#}", ApplyFormatInEditMode = true,
NullDisplayText = "No grade")]

public decimal? Grade { get; set; }

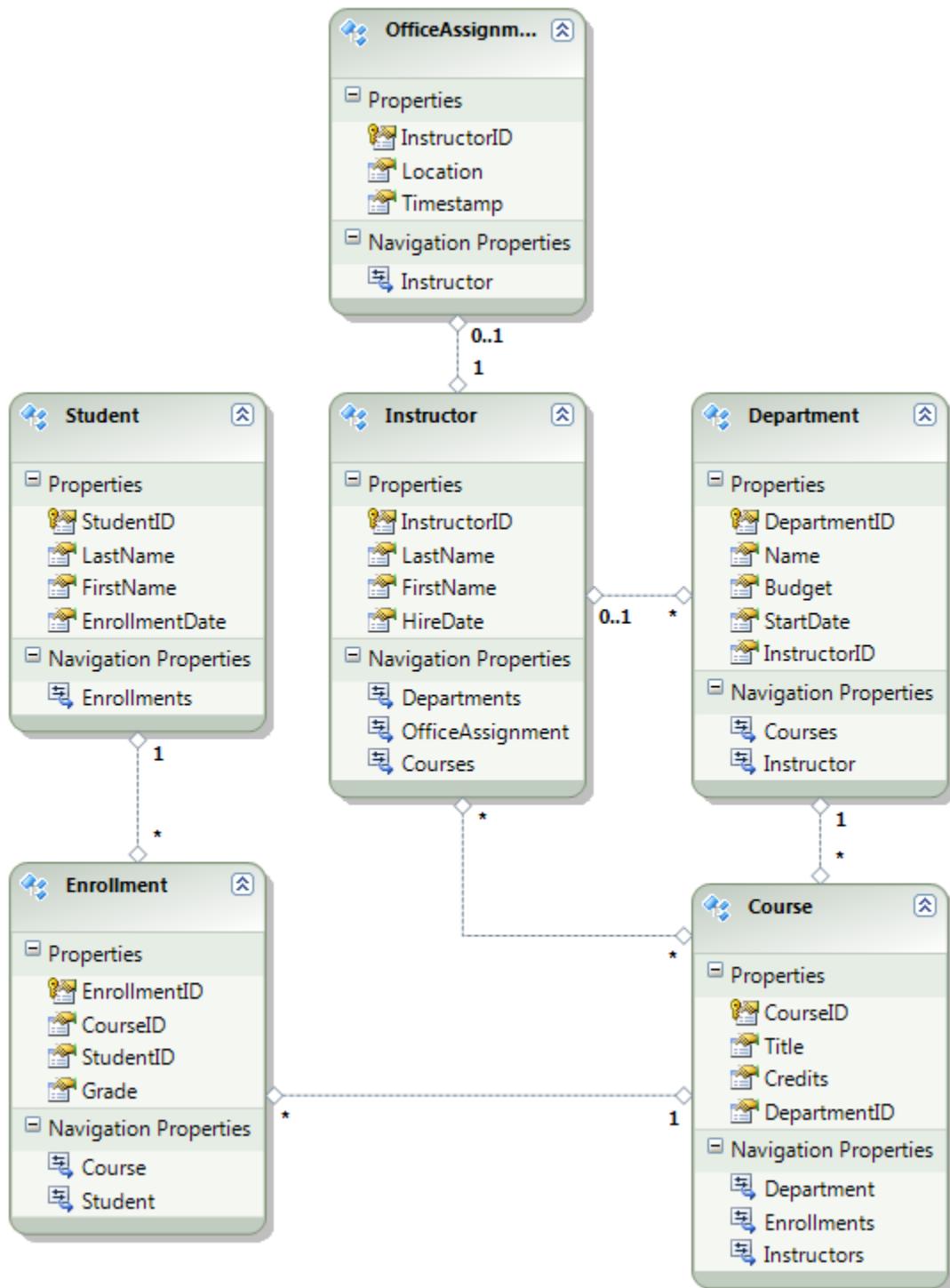
```

Keterangan :

- Grade akan ditampilkan dengan menggunakan 2 digit yang akan dipisahkan oleh periode, sebagai contoh “3.5” atau “4.0”.
- Cara tersebut juga akan digunakan saat berada pada mode edit (format pada text box).
- Jika belum memiliki grade maka akan ditampilkan text “No grade”.

# Menampilkan Relasi pada Entity Diagram

Ilustrasi berikut menampilkan model School dalam diagram dari designer Entity Framework :



Selain garis relasi many-to-many (\* to \*) and one-to-many (1 to \*), kita dapat melihat terdapat garis relasi one-to-zero-or-one (1 to 0..1) antara entity `Instructor` dan `OfficeAssignment` dan garis relasi zero-or-one-to-many (0..1 to \*) antara entity `Instructor` dan `Departement`.

## Modifikasi Database Context

Selanjutnya akan ditambahkan entity baru pada class `SchoolDataContext` dan melakukan modifikasi beberapa mapping dengan menggunakan pemanggilan fluent API ( digunakan kata “fluent” karena sering menggunakan sejumlah method dalam satu statement). Dalam kasus lain kita akan lebih memilih menggunakan method dibanding atribut, karena tidak terdapatnya fungsi tertentu pada atribut. Pada kasus yang lain, kita akan mengguna method ketika method dan atribut tersedia (beberapa orang lebih memilih untuk tidak menggunakan atribut).

Ganti kode pada `DAL\SchoolContext.cs` dengan kode berikut ini :

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using ContosoUniversity.Models;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace ContosoUniversity.Models
{
    public class SchoolContext : DbContext
    {
        public DbSet<Course> Courses { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<OfficeAssignment> OfficeAssignments { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
    }
```

```

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

modelBuilder.Entity<Instructor>()

    .HasOptional(p => p.OfficeAssignment).WithRequired(p => p.Instructor);

modelBuilder.Entity<Course>()

    .HasMany(c => c.Instructors).WithMany(i => i.Courses)

    .Map(t => t.MapLeftKey("CourseID")

        .MapRightKey("InstructorID")

        .ToTable("CourseInstructor")));

modelBuilder.Entity<Department>()

    .HasOptional(x => x.Administrator);

}

}

}

```

Terdapat statement baru yaitu method OnModelCreating yang menentukan hubungan seperti berikut :

- Relasi one-to-zero-or-one antara entity `Instructor` dengan `OfficeAssignment` :

```

modelBuilder.Entity<Instructor>()

    .HasOptional(p => p.OfficeAssignment).WithRequired(p => p.Instructor);

```

- Relasi many-to-many antara entity `Instructor` dan `Course`. Kode berikut menentukan tabel dan nama kolumn untuk join. Konsep Code First dapat melakukan konfigurasi relasi many-to-many ini secara otomatis tanpa harus membuat kode, tapi kita tidak bisa menentukan sendiri karena secara otomatis kita akan mendapatkan nama default seperti `InstructorInstructorID` untuk kolom `Instructor`.

```

modelBuilder.Entity<Course>()

    .HasMany(c => c.Instructors).WithMany(i => i.Courses)

    .Map(t => t.MapLeftKey("CourseID")

        .MapRightKey("InstructorID"))

```

```
.ToTable("CourseInstructor"));
```

- Relasi zero-or-one-to-many antara tabel Instructor dan Departement. Dengan kata lain, departemen mungkin punya atau tidak punya instruktur yang ditugaskan sebagai administrator; penugasan administrator direpresentasikan dalam property navigation `Departement Administrator`.

```
modelBuilder.Entity<Department>()

    .HasOptional(x => x.Administrator);
```

Untuk mengetahui lebih lanjut tentang “fluent API” dapat mengunjungi blog ini <http://blogs.msdn.com/b/aspnetue/archive/2011/05/04/entity-framework-code-first-tutorial-supplement-what-is-going-on-in-a-fluent-api-call.aspx>, yang merupakan posting dari blog milik ASP.NET User Education Team.

## Inisialisasi Database dan Data Testing

Sebelumnya telah dibuat `DAL\School\Initializer.cs` untuk melakukan inisialisasi database untuk memasukkan data test kedalamnya. Sekarang ganti kode tersebut dengan kode di bawah ini :

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using ContosoUniversity.Models;

namespace ContosoUniversity.DAL

{

    public class SchoolInitializer : DropCreateDatabaseIfModelChanges<SchoolContext>

    {
```

```

protected override void Seed(SchoolContext context)
{
    var students = new List<Student>
    {
        new Student { FirstMidName = "Carson", LastName = "Alexander",
EnrollmentDate = DateTime.Parse("2005-09-01") },
        new Student { FirstMidName = "Meredith", LastName = "Alonso",
EnrollmentDate = DateTime.Parse("2002-09-01") },
        new Student { FirstMidName = "Arturo", LastName = "Anand",
EnrollmentDate = DateTime.Parse("2003-09-01") },
        new Student { FirstMidName = "Gytis", LastName = "Barzdukas",
EnrollmentDate = DateTime.Parse("2002-09-01") },
        new Student { FirstMidName = "Yan", LastName = "Li",
EnrollmentDate = DateTime.Parse("2002-09-01") },
        new Student { FirstMidName = "Peggy", LastName = "Justice",
EnrollmentDate = DateTime.Parse("2001-09-01") },
        new Student { FirstMidName = "Laura", LastName = "Norman",
EnrollmentDate = DateTime.Parse("2003-09-01") },
        new Student { FirstMidName = "Nino", LastName = "Olivetto",
EnrollmentDate = DateTime.Parse("2005-09-01") }
    };
    students.ForEach(s => context.Students.Add(s));
    context.SaveChanges();

    var instructors = new List<Instructor>
    {
        new Instructor { FirstMidName = "Kim", LastName = "Abercrombie",
HireDate = DateTime.Parse("1995-03-11") },
        new Instructor { FirstMidName = "Fadi", LastName = "Fakhouri",
HireDate = DateTime.Parse("2002-07-06") },
        new Instructor { FirstMidName = "Roger", LastName = "Harui",
HireDate = DateTime.Parse("1998-07-01") },
        new Instructor { FirstMidName = "Candace", LastName = "Kapoor",

```

```

HireDate = DateTime.Parse("2001-01-15") },

        new Instructor { FirstMidName = "Roger", LastName = "Zheng",
HireDate = DateTime.Parse("2004-02-12") }

};

instructors.ForEach(s => context.Instructors.Add(s));

context.SaveChanges();

var departments = new List<Department>

{

    new Department { Name = "English", Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), InstructorID = 1 },

    new Department { Name = "Mathematics", Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), InstructorID = 2 },

    new Department { Name = "Engineering", Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), InstructorID = 3 },

    new Department { Name = "Economics", Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), InstructorID = 4 }

};

departments.ForEach(s => context.Departments.Add(s));

context.SaveChanges();

var courses = new List<Course>

{

    new Course { CourseID = 1050, Title = "Chemistry", Credits = 3,
DepartmentID = 3, Instructors = new List<Instructor>() },

    new Course { CourseID = 4022, Title = "Microeconomics", Credits = 3,
DepartmentID = 4, Instructors = new List<Instructor>() },

    new Course { CourseID = 4041, Title = "Macroeconomics", Credits = 3,
DepartmentID = 4, Instructors = new List<Instructor>() },

    new Course { CourseID = 1045, Title = "Calculus", Credits = 4,
DepartmentID = 2, Instructors = new List<Instructor>() },

    new Course { CourseID = 3141, Title = "Trigonometry", Credits = 4,

```

```

DepartmentID = 2, Instructors = new List<Instructor>() },
    new Course { CourseID = 2021, Title = "Composition", Credits = 3,
DepartmentID = 1, Instructors = new List<Instructor>() },
    new Course { CourseID = 2042, Title = "Literature", Credits = 4,
DepartmentID = 1, Instructors = new List<Instructor>() }

};

courses.ForEach(s => context.Courses.Add(s));

context.SaveChanges();

courses[0].Instructors.Add(instructors[0]);
courses[0].Instructors.Add(instructors[1]);
courses[1].Instructors.Add(instructors[2]);
courses[2].Instructors.Add(instructors[2]);
courses[3].Instructors.Add(instructors[3]);
courses[4].Instructors.Add(instructors[3]);
courses[5].Instructors.Add(instructors[3]);
courses[6].Instructors.Add(instructors[3]);
context.SaveChanges();

var enrollments = new List<Enrollment>

{
    new Enrollment { StudentID = 1, CourseID = 1050, Grade = 1 },
    new Enrollment { StudentID = 1, CourseID = 4022, Grade = 3 },
    new Enrollment { StudentID = 1, CourseID = 4041, Grade = 1 },
    new Enrollment { StudentID = 2, CourseID = 1045, Grade = 2 },
    new Enrollment { StudentID = 2, CourseID = 3141, Grade = 4 },
    new Enrollment { StudentID = 2, CourseID = 2021, Grade = 4 },
    new Enrollment { StudentID = 3, CourseID = 1050 },
    new Enrollment { StudentID = 4, CourseID = 1050 },
}

```

```

        new Enrollment { StudentID = 4, CourseID = 4022, Grade = 4 },
        new Enrollment { StudentID = 5, CourseID = 4041, Grade = 3 },
        new Enrollment { StudentID = 6, CourseID = 1045 },
        new Enrollment { StudentID = 7, CourseID = 3141, Grade = 2 },
    };

    enrollments.ForEach(s => context.Enrollments.Add(s));

    context.SaveChanges();

    var officeAssignments = new List<OfficeAssignment>
    {
        new OfficeAssignment { InstructorID = 1, Location = "Smith 17" },
        new OfficeAssignment { InstructorID = 2, Location = "Gowan 27" },
        new OfficeAssignment { InstructorID = 3, Location = "Thompson 304" },
    };

    officeAssignments.ForEach(s => context.OfficeAssignments.Add(s));

    context.SaveChanges();
}
}
}

```

Seperti yang telah kita lihat pada tutorial pertama sudah diketahui bahwa kode tersebut akan membuat objek entity baru dan mengisi contoh data ke dalam property-property untuk kebutuhan testing. Namun dengan memperhatikan entity Course yang memiliki relasi many-to-many dengan entity Instructor maka ditangani dengan kode berikut :

```

var officeAssignments = new List<OfficeAssignment>
{
    new OfficeAssignment { InstructorID = 1, Location = "Smith 17" },
    new OfficeAssignment { InstructorID = 2, Location = "Gowan 27" },

```

```

new OfficeAssignment { InstructorID = 3, Location = "Thompson 304" },
};

officeAssignments.ForEach(s => context.OfficeAssignments.Add(s));

context.SaveChanges();

```

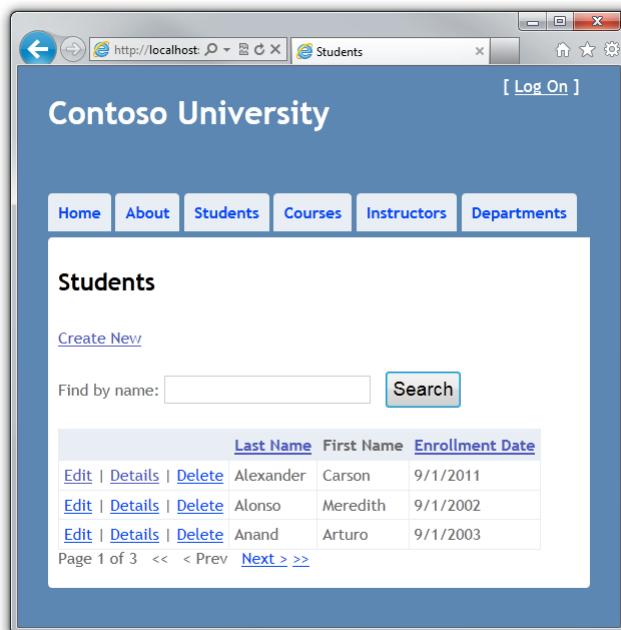
Ketika objek `Course` dibuat, maka kita akan menginisialisasi property navigation `Instructors` dengan collection kosong dengan cara seperti berikut `Instructors = new List()`. Dengan begitu kita dapat menambahkan entity `Instructor` ke `Course` dengan menggunakan method `Instructors.Add`. Jika kita tidak membuat list kosong seperti cara di atas, maka tidak akan dimungkinkan untuk menambahkan relasi seperti di atas dengan cara tadi, karena property `Instructors` bernilai null sehingga tidak memiliki method `Add`.

Note :

Harap diperhatikan saat mendeploy aplikasi ke web server production, kita harus menghilangkan kode-kode yang kita tambahkan tersebut.

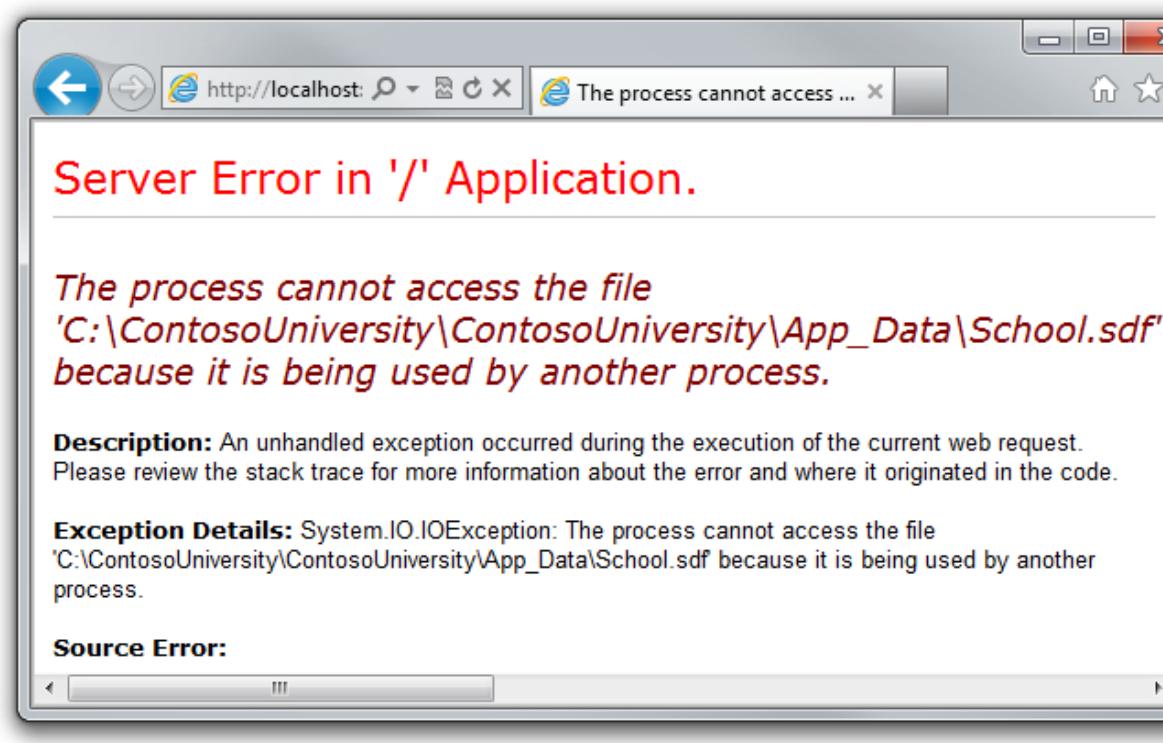
## Menghapus dan Membuat Ulang Database

Sekarang jalankan site dan pilih halaman Index Student.

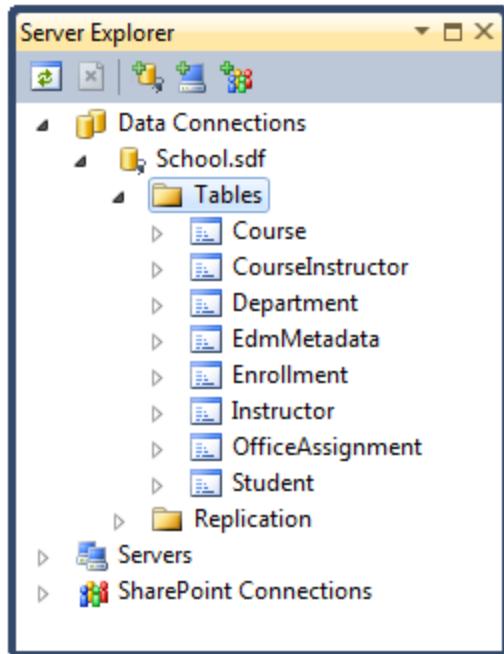


Halaman yang kita lihat ini sama seperti sebelumnya, tetapi terdapat proses pembuatan ulang database yang dilakukan secara background.

Jika kita tidak dapat melihat tampilan halaman Index Student atau halaman menampilkan error dengan pesan yang menyatakan bahwa file *School.sdf* sedang digunakan seperti pada gambar berikut di bawah, maka kita perlu membuka **Server Explorer** dan menutup koneksi terhadap database. Kemudian coba untuk menampilkan halaman tadi lagi.



Setelah langkah diatas dilakukan, kembali buka **Server Explorer** dan buka daftar tabel seperti pada gambar di bawah, maka kita akan dapat melihat seluruh tabel telah dibuat.



Selain tabel `EdmMetadata`, kita juga melihat tabel yang tidak dibuat class modelnya yaitu `CourseInstructor`. Seperti yang telah dijelaskan sebelumnya bahwa tabel tersebut merupakan tabel untuk relasi many-to-many antara entity `Instructor` dan `Course`.

Klik kanan pada tabel `CourseInstructor` dan pilih **Show Tabel Data** untuk membuktikan bahwa terdapat data tabel tersebut sebagai hasil dari entity `Instructor` yang telah kita tambahkan dengan menggunakan property navigation `Course.Instructor`.

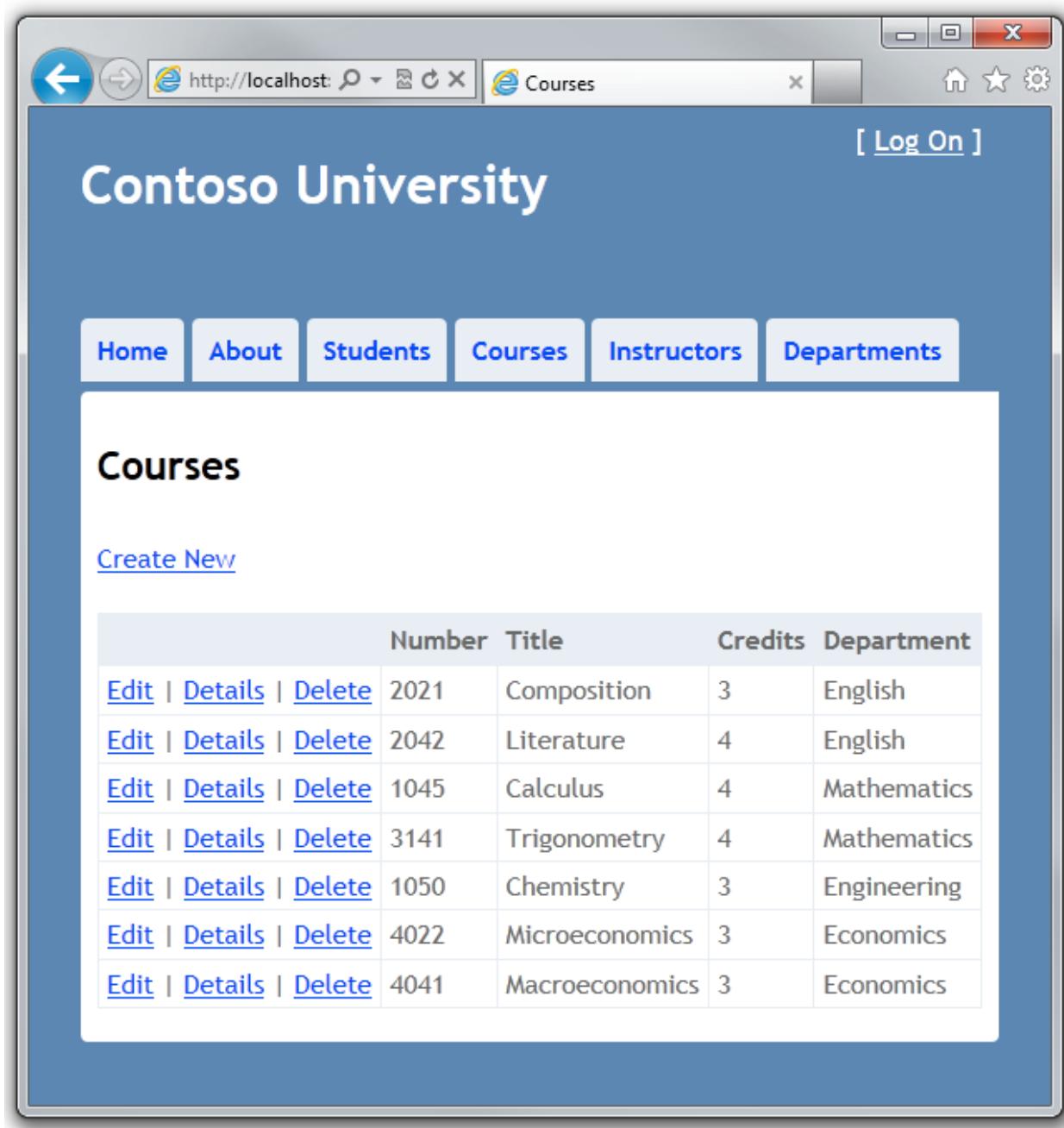
	CourseID	InstructorID
▶	1045	4
	1050	1
	1050	2
	2021	4
	2042	4
	3141	4
	4022	3
	4041	3
*	NULL	NULL

Sekarang kita telah memiliki data model yang lebih rumit. Pada tutorial selanjutnya kita akan belajar cara-cara berbeda dalam mengakses.

# Membaca Data Terkait dengan Entity Framework pada Aplikasi ASP.NET MVC

Pada tutorial sebelumnya kita telah melengkapi data model School. Pada tutorial ini berisi bagaimana membaca data terkait yang ada, data yang akan dimuat oleh Entity Framework ke property-property navigation.

Gambar berikut ini akan memperlihatkan halaman yang nanti akan kita buat.



The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title bar says "Instructors". The page header features the Contoso University logo and a "Log On" link. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Instructors" and contains a "Create New" link. Below this is a table listing five instructors with columns for Last Name, First Name, Hire Date, and Office. Each row has "Select", "Edit", "Details", and "Delete" links. The table rows are: 1. Abercrombie, Kim, 3/11/1995, Smith 17; 2. Fakhouri, Fadi, 7/6/2002, Gowan 27; 3. Harui, Roger, 7/1/1998, Thompson 304; 4. Kapoor, Candace, 1/15/2001 (highlighted); 5. Zheng, Roger, 2/12/2004. Below the table is a section titled "Courses Taught by Selected Instructor" with a table showing courses taught by Kapoor: ID 1045, Title Calculus, Department Mathematics; ID 2021, Title Composition, Department English; ID 2042, Title Literature, Department English; ID 3141, Title Trigonometry, Department Mathematics. At the bottom is a section titled "Students Enrolled in Selected Course" with a table showing students enrolled in Trigonometry: Alonso, Meredith, Grade 4.00; Norman, Laura, Grade 2.00.

	Last Name	First Name	Hire Date	Office
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	Smith 17
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Gowan 27
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001	
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004	

**Courses Taught by Selected Instructor**

ID	Title	Department
1045	Calculus	Mathematics
2021	Composition	English
2042	Literature	English
3141	Trigonometry	Mathematics

**Students Enrolled in Selected Course**

Name	Grade
Alonso, Meredith	4.00
Norman, Laura	2.00

# Lazy, Eager, dan Explicit Loading Data

Ada beberapa cara yang digunakan oleh Entity Framework untuk memuat data terkait ke property-navigation dari entity :

- *Lazy loading*, ketika entity pertama kali dibaca, data terkait belum diambil. Namun ketika kita mulai mengakses property navigation maka data yang dibutuhkan oleh property navigation akan secara otomatis diambil. Hasil dari beberapa query dikirim ke database – satu untuk entity itu sendiri dan query yang lain untuk setiap data yang terkait harus diambil.

```
departments = context.Departments
foreach (Department d in departments) ← Query: all Department rows
{
    foreach (Course c in d.Courses) ← Query: Course rows related to
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

- *Eager loading*, ketika entity dibaca maka data yang terkait juga akan diambil. Biasanya ini adalah hasil dari query single join yang mengambil seluruh data yang dibutuhkan. Untuk menggunakan eager loading maka digunakan method `Include`.

```
departments = context.Departments.Include(x => x.Courses)
foreach (Department d in departments) ← Query: all Department
{
    foreach (Course c in d.Courses) ← rows and related
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

- *Explicit loading*, yang ini sama seperti lazy loading bedanya kita melakukan pengambilan data terkait dari kode. Pengambilan data tidak dilakukan secara otomatis saat kita mengakses property navigation. Kita mengambil data secara manual dengan cara memanggil method `Collection.Load` untuk koleksi data atau `Reference.Load` untuk property-property pada entity tunggal. (Pada contoh berikut, jika kita ingin mengambil property navigation `Administrator`, maka kita bisa mengubah `Collection(x => x.Course)` dengan `Reference(x => x.Administrator)`).

```

departments = context.Departments.ToList();
foreach (Department d in departments) ← Query: all Department rows
{
    context.Entry(d).Collection(x => x.Courses).Load(); ← Query: Course rows
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}

```

Lazy loading dan explicit loading juga sering disebut sebagai *deffered loading*, karena keduanya tidak langsung mengambil nilai property-property.

Secara umum, jika diketahui kita membutuhkan data setiap kali entity dipanggil maka eager loading menawarkan performansi terbaik untuk itu, karena sebuah query dikirimkan ke database lebih effisien dibandingkan mengiriman sebagian-sebagian query setiap ingin mengambil data. Sebagai contoh, jika setiap departement mempunyai 10 course. Maka hasil dari eager loading hanya berupa satu join query. Sedangkan lazy loading dan explicit loading akan menghasilkan 11 query.

Tetapi kita tidak sering mengakses property-property navigation milik entity atau hanya akan mengakses sebagian kecil dari set entity saja, maka lazy loading lebih effisien untuk digunakan, karena jika menggunakan eager loading maka seluruh akan banyak data yang kita butuhkan yang akan kita dapatkan juga. Biasanya kita menggunakan explisit loading ketika fitur lazy loading dimatikan. Skenario yang mungkin membuat fitur lazy loading dimatikan adalah saat proses serialization, ketika kita tidak membutuhkan seluruh property-navigation dimuat. Jika fitur lazy loading aktif, maka seluruh property-navigation akan dimuat secara otomatis, karena proses serialization akan mengakses seluruh property tersebut.

Fitur lazy loading pada class database context aktif secara default. Ada dua cara yang dapat kita lakukan untuk mematikan fitur ini, yaitu :

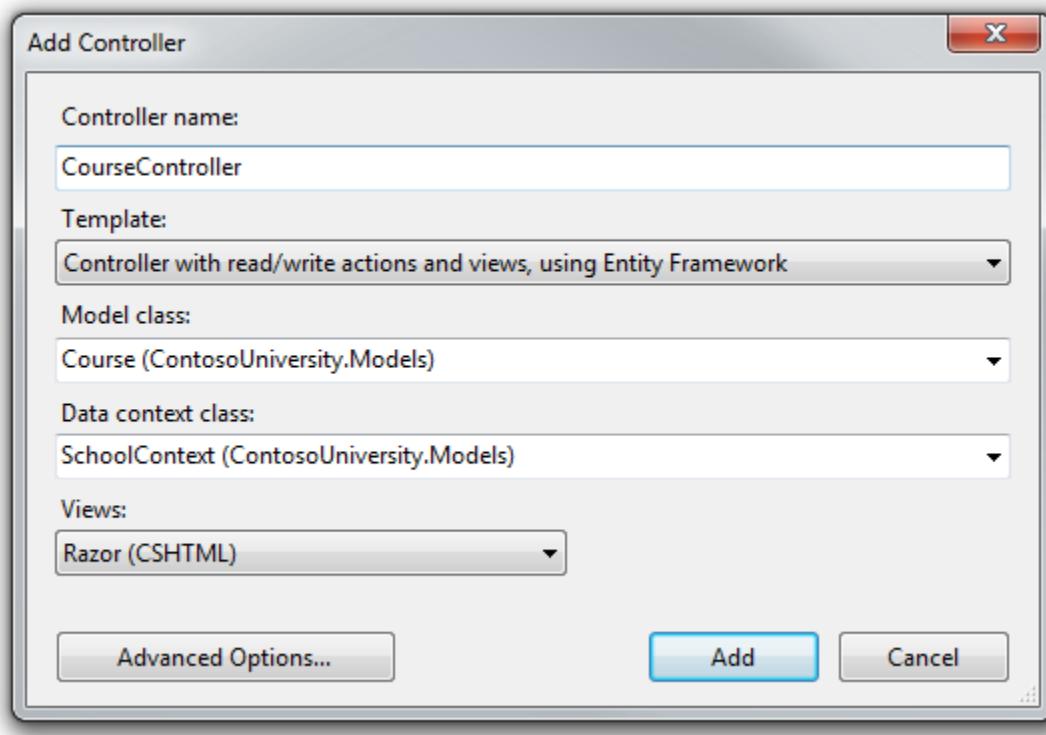
- Untuk menonaktifkan fitur ini hanya pada property-navigation tertentu saja, kita dapat menambahkan keyword `virtual` ketika kita mendeklarasikan property.
- Untuk menonaktifkan fitur ini pada semua property navigation dapat kita lakukan dengan memberikan nilai `false` pada `LazyLoadingEnabled`.

Lazy loading dapat menutupi kode yang menyebabkan masalah performansi. Sebagai contoh, kode yang tidak dispesifikasikan eager loading atau explicit loading tetapi sangat sering melakukan akses entity dan property navigation pada setiap iterasi adalah sangat tidak effisien (karena banyak akses bolak-balik ke database), tetapi kode tersebut akan berjalan tanpa error jika menggunakan lazy loading. Mematikan fitur lazy loading adalah cara untuk mengetahui kode yang bergantung dengan lazy loading, karena hal itu akan menyebabkan property-navigation bernilai null dan kode akan mengalami kegagalan.

## Membuat Halaman Course Index untuk Menampilkan Nama Departement

Pada entity `Course` terdapat property navigation yang berisi entity `Departement`. Untuk menampilkan nama departemen yang terdapat pada daftar course, kita harus mengambil property `Name` dari entity `Departement` yang berada dalam property navigation `Course.Department`.

Buat controller untuk entity `Course` dengan cara seperti yang sebelumnya saat kita membuat controller `Student`.



Buka file `Controllers\CourseController.cs` dan perhatikan method `Index` :

```
public ViewResult Index()
{
    var courses = db.Courses.Include(c => c.Department);
    return View(courses.ToList());
}
```

Pada kode tersebut dapat kita lihat digunakan eager loading untuk property navigation `Departement`, karena terdapat penggunaan method `Include`.

Buka file `Views\Course\Index.cshtml` dan ganti kode yang sudah ada dengan kode berikut ini :

```
@model IEnumerable<ContosoUniversity.Models.Course>

@{
    ViewBag.Title = "Courses";
}

<h2>Courses</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>

<table>
    <tr>
        <th></th>
        <th>Number</th>
        <th>Title</th>
        <th>Credits</th>
        <th>Department</th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id=item.CourseID }) |
                @Html.ActionLink("Details", "Details", new { id=item.CourseID }) |
            </td>
        </tr>
    }
</table>
```

```

        @Html.ActionLink("Delete", "Delete", new { id=item.CourseID })

    </td>

    <td>

        @Html.DisplayFor(modelItem => item.CourseID)

    </td>

    <td>

        @Html.DisplayFor(modelItem => item.Title)

    </td>

    <td>

        @Html.DisplayFor(modelItem => item.Credits)

    </td>

    <td>

        @Html.DisplayFor(modelItem => item.Department.Name)

    </td>

</tr>

}

</table>

```

Berikut adalah kode-kode yang telah kita ubah :

- Mengubah heading **Index** menjadi **Courses**.
- Memindahkan link pada baris ke sebelah kiri.
- Menambahkan kolom **Number** yang menampilkan nilai property **CourseID**.
- Mengubah nama heading **DepartmentID** menjadi **Departement**.

Pada kolom terakhir ditampilkan nilai property **Name** dari entity **Departement** yang dimuat ke property navigation **Departement** :

```

<td>

    @Html.DisplayFor(modelItem => item.Department.Name)

</td>

```

Jalankan halaman dan pilih tab Courses kemudian dapat kita lihat daftar dengan nama departement seperti pada gambar berikut :

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title of the page is "Courses". The browser interface includes standard controls like back, forward, search, and refresh buttons, along with a "Log On" link in the top right corner.

The main content area displays the "Contoso University" logo and a navigation menu with links for Home, About, Students, Courses, Instructors, and Departments. The "Courses" link is highlighted, indicating the current page.

## Courses

[Create New](#)

	Number	Title	Credits	Department
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2021	Composition	3	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2042	Literature	4	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1045	Calculus	4	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	3141	Trigonometry	4	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1050	Chemistry	3	Engineering
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4022	Microeconomics	3	Economics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4041	Macroeconomics	3	Economics

## **Membuat Halaman Instructors Index untuk Menampilkan Course dan Enrollment**

Pada bagian ini kita akan membuat controller dan view untuk entity `Instructor` dan menampilkannya pada halaman Instructor Index.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title of the page is "Instructors". The header features the "Contoso University" logo and a "Log On" link. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Instructors" and contains a "Create New" link. Below this is a table listing five instructors with columns: Last Name, First Name, Hire Date, and Office. Each row includes "Select", "Edit", "Details", and "Delete" links. The table data is as follows:

	Last Name	First Name	Hire Date	Office
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	Smith 17
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Gowan 27
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001	
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004	

Below the table is a section titled "Courses Taught by Selected Instructor" containing another table with columns: ID, Title, and Department. The table data is as follows:

	ID	Title	Department
<a href="#">Select</a>	1045	Calculus	Mathematics
<a href="#">Select</a>	2021	Composition	English
<a href="#">Select</a>	2042	Literature	English
<a href="#">Select</a>	3141	Trigonometry	Mathematics

Finally, there is a section titled "Students Enrolled in Selected Course" containing a table with columns: Name and Grade. The table data is as follows:

Name	Grade
Alonso, Meredith	4.00
Norman, Laura	2.00

Halaman ini membaca dan menampilkan data dengan cara berikut :

- Daftar instruktur menampilkan data dari entity `OfficeAssignment`. Entity `Instructor` dan `OfficeAssignment` mempunyai relasi one-to-zero-or-one. Kita akan menggunakan eager loading pada entity `OfficeAssignment`. Seperti yang telah dijelaskan sebelumnya, eager loading lebih effisien jika kita membutuhkan data dari seluruh baris dari tabel utama. Pada kasus ini kita akan menampilkan office assignment dari seluruh instruktur yang ditampilkan.
- Ketika user memilih seorang instruktur, maka entity `Course` yang berhubungan dengan data instruktur tersebut akan ditampilkan. Entity `Instructor` dan `Course` mempunyai relasi many-to-many. Kita akan menggunakan eager loading untuk entity `Course` dan entity `Departement` yang terkait. Pada kasus ini, lazy loading mungkin akan lebih effisien karena kita hanya membutuhkan course untuk instruktur yang dipilih saja. Namun, pada contoh ini akan diberikan cara penggunaan eager loading untuk property navigation dalam entity yang berada di dalam property navigation.
- Ketika user memilih course, data terkait dari entity `Enrollment` ditampilkan. Relasi antara entity `Course` dan `Enrollment` adalah one-to-many. Kita akan menambahkan explicit loading pada entity `Enrollment` dan entity `Student` yang terkait. (explicit loading sebenarnya tidak diperlukan karena pada kasus ini fitur lazy loading aktif, tetapi disini hanya untuk menunjukkan bagaimana menggunakan explicit loading.)

## Membuat View Model untuk View dari Instructor Index

Halaman Instructor Index menampilkan tiga tabel yang berbeda. Karena itu kita akan membuat sebuah view model untuk menangani ketiga tabel tersebut.

Pada folder ViewModels, buat file `InstructorIndexData.cs` dan ganti kode didalamnya dengan kode berikut ini :

```
using System;
using System.Collections.Generic;
using ContosoUniversity.Models;

namespace ContosoUniversity.ViewModels
{
    public class InstructorIndexData
    {
        public IEnumerable<Instructor> Instructors { get; set; }

        public IEnumerable<Course> Courses { get; set; }

        public IEnumerable<Enrollment> Enrollments { get; set; }
    }
}
```

```
}
```

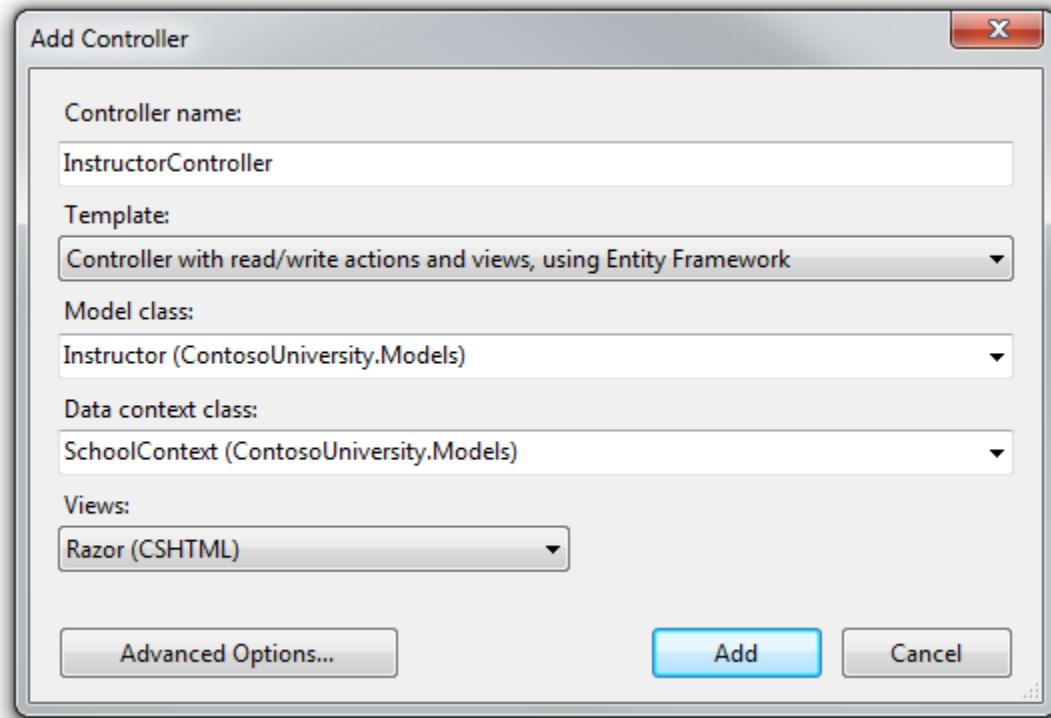
## Penambahan Style pada Row yang dipilih

Untuk menandai baris yang dipilih, kita bisa membuat tanda dengan menambahkan warna latar yang berbeda. Untuk membuat antarmuka seperti itu kita bisa menambahkan bagian baru pada file *Content\Site.css* dengan kode seperti berikut ini.

```
/* MISC
-----
.selectedrow
{
    background-color: #EEEEEE;
}
```

## Membuat Controller dan View untuk Instructor

Membuat controller untuk entity Instrutor, dengan cara seperti yang telah kita lakukan saat membuat controller untuk entity Student.



Buka file *Controllers\InstructorController.cs* dan tambahkan statement using untuk namespace *ViewModels*.

```
using ContosoUniversity.ViewModels;
```

Pada method *Index* dapat dilihat penggunaan eager loading pada property navigation *OfficeAssignment* :

```
public ViewResult Index()
{
    var instructors = db.Instructors.Include(i => i.OfficeAssignment);
    return View(instructors.ToList());
}
```

Ganti kode pada method *Index* dengan kode berikut ini, kode ini berfungsi untuk memuat data tambahan dan meletakkannya pada view model :

```
public ActionResult Index(Int32? id, Int32? courseID)
{
    var viewModel = new InstructorIndexData();

    viewModel.Instructors = db.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.Courses.Select(c => c.Department))
        .OrderBy(i => i.LastName);

    if (id != null)
    {
```

```

        ViewBag.InstructorID = id.Value;

        viewModel.Courses = viewModel.Instructors.Where(i => i.InstructorID ==
id.Value).Single().Courses;

    }

    if (courseID != null)

    {

        ViewBag.CourseID = courseID.Value;

        viewModel.Enrollments = viewModel.Courses.Where(x => x.CourseID ==
courseID).Single().Enrollments;

    }

    return View(viewModel);

}

```

Method di atas memiliki parameter query string yang bersifat opsional, parameter yang dapat dimasukkan adalah nilai ID dari instruktur yang dipilih dan course yang dipilih, dan akan menampilkan data pada view. Nilai untuk query string akan diberikan oleh hyperlink **Select** yang terdapat pada halaman.

Kode akan mengawali dengan membuat sebuah instance view model dan meletakkannya pada data instruktur dalam bentuk list :

```

var viewModel = new InstructorIndexData();

viewModel.Instructors = db.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.Courses.Select(c => c.Department))
    .OrderBy(i => i.LastName);

```

Baris di atas menunjukkan penggunaan eager loading untuk property navigation `Instructor.OfficeAssignment` dan `Instructor.Courses`. Pada kode di atas dapat dilihat contoh penggunaan eager loading untuk property navigation `Course.Department` dengan menggunakan method `Select` didalam method `Include`. Hasilnya kemudian diurut berdasarkan nama akhir dari instruktur.

Jika data salah satu instruktur dipilih, data instruktur tersebut didapat dari list instruktur di dalam view model. Property `Courses` milik view model dimuat dengan entity `Course` yang didapat dari property navigation `Courses` milik instruktur.

```
if (id != null)
{
    ViewBag.InstructorID = id.Value;

    viewModel.Courses = viewModel.Instructors.Where(i => i.InstructorID ==
id.Value).Single().Courses;
}
```

Method `Where` akan mengembalikan collection, tetapi pada kasus ini kriteria yang tepat hanya akan mengembalikan sebuah entity `Instructor` saja. Untuk mendapatkan hal tersebut digunakan method `Single`. Dengan mendapatkan satu entity `Instructor`, maka kita dapat mengakses property `Course` dari entity tersebut.

Penggunaan method `Single` akan mengeluarkan exception jika ternyata collection bernilai null, untuk menghindari keluarnya exception maka dapat digunakan method `SingleOrDefault` yang akan mengembalikan null jika collection bernilai kosong. Ada beberapa cara yang dapat digunakan untuk penggunaan method `Single` untuk tujuan di atas, yang diperlihatkan pada contoh di bawah ini :

```
.Single(i => i.InstructorID == id.Value)
```

Atau

```
.Where(i => i.InstructorID == id.Value).Single()
```

Selanjutnya jika sebuah course dipilih, nilainya didapat dari list course pada view model. Property `Enrollments` milik view model dimuat dengan entity `Enrollment` yang didapat dari property navigation `Enrollments` milik course.

```
if (courseID != null)
{
    ViewBag.CourseID = courseID.Value;

    viewModel.Enrollments = viewModel.Courses.Where(x => x.CourseID ==
courseID).Single().Enrollments;
}
```

Terakhir, view model akan dikembalikan ke view :

```
return View(viewModel);
```

## Modifikasi View pada Instructor Index

Pada file `Views\Instructor\Index.cshtml`, ganti kode yang ada dengan kode berikut ini :

```
@model ContosoUniversity.ViewModels.InstructorIndexData

@{
    ViewBag.Title = "Instructors";
}

<h2>Instructors</h2>

<p>
```

```

@Html.ActionLink("Create New", "Create")

</p>

<table>

    <tr>

        <th></th>

        <th>Last Name</th>

        <th>First Name</th>

        <th>Hire Date</th>

        <th>Office</th>

    </tr>

    @foreach (var item in Model.Instructors)

    {

        string selectedRow = "";

        if (item.InstructorID == ViewBag.InstructorID)

        {

            selectedRow = "selectedrow";

        }

        <tr class="@selectedRow" valign="top">

            <td>

                @Html.ActionLink("Select", "Index", new { id = item.InstructorID }) |

                @Html.ActionLink("Edit", "Edit", new { id = item.InstructorID }) |

                @Html.ActionLink("Details", "Details", new { id = item.InstructorID })

            |

            @Html.ActionLink("Delete", "Delete", new { id = item.InstructorID })

        </td>

        <td>

            @item.LastName

        </td>
    }

```

```

<td>
    @item.FirstMidName
</td>
<td>
    @String.Format("{0:d}", item.HireDate)
</td>
<td>
@if (item.OfficeAssignment != null)
{
    @item.OfficeAssignment.Location
}
</td>
</tr>
}
</table>

```

Berikut adalah hal-hal yang diubah dari kode sebelumnya :

- Mengganti judul halaman dari **Index** menjadi **Instructors**.
- Memindahkan link pada baris ke sebelah kiri.
- Menghapus kolom **FullName**.
- Menambahkan kolom Office yang menampilkan nilai dari `item.OfficeAssignment.Location` jika nilai `item.OfficeAssignment` tidak null. (Karena relasinya adalah one-to-zero-or-one maka ada kemungkinan tidak ada keterkaitan dengan entity `OfficeAssignment`.)

```

<td>
@if (item.OfficeAssignment != null)
{
    @item.OfficeAssignment.Location
}

```

```
</td>
```

- Menambahkan kode yang dapat menambahkan `baris class="selectedrow"` ke elemen tr secara dinamik ketika baris instruktur dipilih. Style CSS untuk warna latar belakang untuk baris yang dipilih telah kita buat sebelumnya. (atribut `valign` akan berguna jika kolom berisi lebih dari beberapa baris.)

```
string selectedRow = "";  
  
if (item.InstructorID == ViewBag.InstructorID)  
{  
    selectedRow = "selectedrow";  
}  
  
<tr class="@selectedRow" valign="top">
```

- Menambahkan `ActionLink` dengan label Select yang berfungsi untuk mengirimkan nilai ID instruktur ke method `Index`.

Jalankan halaman untuk melihat daftar instruktur. Pada halaman akan ditampilkan property `Location` jika ada hubungan dengan entity `OfficeAssignment`, dan akan menampilkan sel kosong jika tidak ada hubungan dengan entity `OfficeAssignment`.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title of the page is "Instructors". The main content area displays the heading "Instructors" and a link "Create New". Below this is a table listing five instructors with columns for Last Name, First Name, Hire Date, and Office. Each row contains links for Select, Edit, Details, and Delete.

	Last Name	First Name	Hire Date	Office
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	Smith 17
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Gowan 27
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001	
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004	

Pada file Views\Instructor\Index.cshtml, tambahkan kode berikut ini setelah elemen table. Fungsi kode ini untuk menampilkan course yang berhubungan dengan instruktur ketika sebuah data instruktur dipilih.

```
@if (Model.Courses != null)
{
    <h3>Courses Taught by Selected Instructor</h3>
    <table>
        <tr>
            <th></th>
            <th>ID</th>
            <th>Title</th>
```

```

<th>Department</th>
</tr>

@foreach (var item in Model.Courses)
{
    string selectedRow = "";
    if (item.CourseID == ViewBag.CourseID)
    {
        selectedRow = "selectedrow";
    }
    <tr class="@selectedRow">
        <td>
            @Html.ActionLink("Select", "Index", new { courseID = item.CourseID })
        </td>
        <td>
            @item.CourseID
        </td>
        <td>
            @item.Title
        </td>
        <td>
            @item.Department.Name
        </td>
    </tr>
}
</table>
}

```

Kode tersebut akan membaca property `Courses` pada view model untuk menampilkan daftar course. Kode tersebut juga menyediakan hyperlink `Select` yang akan mengirim ID dari course yang dipilih ke method `Index`.

Jalankan page dan pilih sebuah data instruktur. Sekarang kita akan dapat lihat grid yang berisi data course yang ditangani oleh instruktur tersebut. dan setiap dapat kita lihat juga nama departemen dari setiap course tersebut.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title of the page is "Instructors". The main content area displays the "Instructors" section of the Contoso University website. It includes a "Create New" link and a table listing five instructors with columns for Last Name, First Name, Hire Date, and Office. Below the table, there is a section titled "Courses Taught by Selected Instructor" with a table showing four courses with columns for ID, Title, and Department.

	Last Name	First Name	Hire Date	Office
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	Smith 17
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Gowan 27
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001	
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004	

ID	Title	Department
1045	Calculus	Mathematics
2021	Composition	English
2042	Literature	English
3141	Trigonometry	Mathematics

**Note :**

Jika tidak ada highlight pada baris yang dipilih, klik tombol **Refresh** pada browser atau tekan tombol F5. Karena ada kemungkinan kita harus memuat ulang file .css. Jika hal tersebut tidak berhasil, kita dapat mencoba untuk melakukan cara lain, dengan menekan tombol CTRL dan klik **Refresh** secara bersamaan atau tekan tombol CTRL+F5.

Tambahkan kode di bawah ini setelah kode yang sebelumnya telah kita tambahkan. Akan ditampilkan daftar student yang terdaftar pada course yang dipilih.

```
@if (Model.Enrollments != null)  
{  
    <h3>  
        Students Enrolled in Selected Course</h3>  
    <table>  
        <tr>  
            <th>Name</th>  
            <th>Grade</th>  
        </tr>  
        @foreach (var item in Model.Enrollments)  
        {  
            <tr>  
                <td>  
                    @item.Student.FullName  
                </td>  
                <td>  
                    @Html.DisplayFor(modelItem => item.Grade)  
                </td>  
            </tr>  
        }  
    </table>  
}
```

Kode tersebut akan membaca property `Enrollment` dari view model dan menampilkan daftar student yang terdaftar dalam course. Helper `DisplayFor` digunakan untuk menampilkan pesan “No grades” jika grade bernilai null, seperti yang telah didefinisikan pada atribut data annotation `DisplayFormat` untuk field tersebut.

Jalankan halaman dan pilih salah satu data instruktur pada daftar, sekarang akan kita lihat hasilnya seperti pada gambar berikut ini.

The screenshot shows a web browser window for 'Instructors' at 'http://localhost'. The title bar says 'Instructors'. The main content area displays the 'Contoso University' logo and a navigation menu with links to Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word 'Instructors' is highlighted in bold. A link 'Create New' is visible. A table lists five instructors with columns for Last Name, First Name, Hire Date, and Office. The table includes links for Select, Edit, Details, and Delete. The last row is highlighted. Below the table, a section titled 'Courses Taught by Selected Instructor' shows a table with columns ID, Title, and Department for courses taught by the selected instructor. The table includes links for Select. The last row is highlighted. At the bottom, a section titled 'Students Enrolled in Selected Course' shows a table with columns Name and Grade for students enrolled in the selected course. The table includes links for Select.

	Last Name	First Name	Hire Date	Office
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	Smith 17
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Gowan 27
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001	
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004	

ID	Title	Department
<a href="#">Select</a>	1045	Calculus
<a href="#">Select</a>	2021	Composition
<a href="#">Select</a>	2042	Literature
<a href="#">Select</a>	3141	Trigonometry
		Mathematics

Name	Grade
Alonso, Meredith	4.00
Norman, Laura	2.00

## Menambahkan Explicit Loading

Buka file InstructorController.cs dan perhatikan bagaimana method Index mendapatkan data enrollment untuk course yang dipilih.

```
if (courseID != null)
{
    ViewBag.CourseID = courseID.Value;

    viewModel.Enrollments = viewModel.Courses.Where(x => x.CourseID ==
courseID).Single().Enrollments;
}
```

ketika data instruktur diambil, kita menggunakan eager loading untuk property navigation Courses dan property Department dari setiap course. Kemudian collection Courses akan diletakkan ke dalam view model dan kita akan dapat mengakses property navigation dari salah satu entity yang ada pada collection tersebut. Karena kita menentukan eager loading untuk property navigation Course.Enrollments maka data dari property tersebut akan ditampilkan pada halaman sebagai hasil dari lazy loading.

Jika kita mematikan fitur lazy loading tanpa mengubah kode yang ada, maka property Enrollments akan bernilai null. Untuk memuat property Enrollments maka perlu digunakan eager loading atau explicit loading. Pada contoh ini diperlihatkan penggunaan explicit loading, ganti kode method Index dengan kode berikut ini :

```
public ActionResult Index(Int32? id, Int32? courseID)
{
    var viewModel = new InstructorIndexData();

    viewModel.Instructors = db.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.Courses.Select(c => c.Department))
        .OrderBy(i => i.LastName);

    if (id != null)
    {
```

```

        ViewBag.InstructorID = id.Value;

        viewModel.Courses = viewModel.Instructors.Where(i => i.InstructorID ==
id.Value).Single().Courses;

    }

    if (courseID != null)
    {

        ViewBag.CourseID = courseID.Value;

        var selectedCourse = viewModel.Courses.Where(x => x.CourseID ==
courseID).Single();

        db.Entry(selectedCourse).Collection(x => x.Enrollments).Load();

        foreach (Enrollment enrollment in selectedCourse.Enrollments)
        {

            db.Entry(enrollment).Reference(x => x.Student).Load();
        }

        viewModel.Enrollments = selectedCourse.Enrollments;
    }

    return View(viewModel);
}

```

Setelah memilih entity Course, kode di bawah ini akan memperlihatkan untuk mendapatkan property navigation Enrollments dengan cara explicit loading :

```
db.Entry(selectedCourse).Collection(x => x.Enrollments).Load();
```

Begitu juga kode di bawah ini :

```
db.Entry(enrollment).Reference(x => x.Student).Load();
```

Terdapat penggunaan method Collection untuk memuat collection property, tetapi untuk memuat sebuah property dari sebuah entity digunakan method Reference.

Sekarang kita jalankan page Instructor Index dan kita akan melihat tidak ada perbedaan tampilkan dengan tampilkan sebelum kita mengubah kode, padahal pada kode ini digunakan cara yang berbeda dalam mengambil data.

Kita telah menggunakan tiga cara (lazy, eager dan explicit) untuk mengambil data ke property-property navigation. Pada tutorial selanjutnya kita akan mempelajari cara untuk mengupdate data.

Berikut adalah link sumber bacaan Entity Framework yang lain <http://www.asp.net/entity-framework/tutorials/advanced-entity-framework-scenarios-for-an-mvc-web-application>.

# **Update Data Terkait dengan Entity Framework pada Aplikasi ASP.NET MVC**

---

Pada tutorial kita akan belajar mengupdate data dengan menggunakan Entity Framework. pada kasus relasi yang sederhana, maka proses update dapat dilakukan dengan mudah. Sedangkan jika relasi adalah many-to-many maka langkah-langkah untuk update akan lebih panjang.

Pada gambar di bawah ini diperlihatkan halaman-halaman yang akan kita buat.

The screenshot shows a web browser window with the URL <http://localhost>. The title bar says "Create". The main content area displays the "Contoso University" logo and a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Create" is displayed in large bold letters. A form titled "Course" is shown with fields for Number (1000), Title (Algebra), Credits (5), and Department (Mathematics). A "Create" button is at the bottom of the form. At the bottom left of the main content area is a link "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Create

**Course**

Number  
1000

Title  
Algebra

Credits  
5

Department  
Mathematics ▾

**Create**

[Back to List](#)

The screenshot shows a web browser window with the URL <http://localhost>. The title bar says "Edit". The main content area displays the Contoso University logo and navigation menu. The menu items are Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Edit" is displayed. A form titled "Course" contains fields for Number (1000), Title (Algebra), Credits (5), and Department (Mathematics). A "Save" button is at the bottom of the form. At the bottom left, there is a link "Back to List".

[ Log On ]

# Contoso University

Home About Students Courses Instructors Departments

## Edit

**Course**

Number  
1000

Title

Credits

Department

**Save**

[Back to List](#)

The screenshot shows a web application window titled "Contoso University". At the top, there is a navigation bar with links for Home, About, Students, Courses, Instructors, and Departments. A "Log On" link is also present in the top right corner. The main content area is titled "Edit" and contains a form for an "Instructor". The form includes fields for Last Name (Abercrombie), First Name (Kim), Hire Date (3/11/1995), and Office Location (17 Mordor Tower). Below these fields is a section titled "Courses" containing a grid of checkboxes. The grid lists courses such as Calculus, Chemistry, Composition, Literature, Trigonometry, Microeconomics, Macroeconomics, and Algebra. The checkbox for Chemistry is checked. At the bottom of the form is a "Save" button and a "Back to List" link.

<input type="checkbox"/> 1045 Calculus	<input checked="" type="checkbox"/> 1050 Chemistry	<input type="checkbox"/> 2021 Composition
<input type="checkbox"/> 2042 Literature	<input type="checkbox"/> 3141 Trigonometry	<input type="checkbox"/> 4022 Microeconomics
<input type="checkbox"/> 4041 Macroeconomics	<input type="checkbox"/> 1000 Algebra	

## Mengubah Halaman Create dan Edit untuk Course

Saat sebuah entity course dibuat, maka harus memiliki relasi dengan department yang sudah ada. Untuk memfasilitasi ini maka pada view Create dan Edit dibuat pilihan berupa dropdown yang berisi daftar department. Daftar departement yang didapat dari property foreign key

`Course.DepartmentID`. Entity Framework akan memuat property navigation Department dengan

entity Department yang sesuai, yaitu department yang berhubungan dengan Course. Kita akan menggunakan kode yang sudah disediakan, tetapi dengan penambahan untuk penanganan error dan pengurutan pada daftar drop down.

Pada file CourseControllers.cs hapus empat method `Edit` dan `Create` dan ganti dengan kode di bawah ini :

```
public ActionResult Create()

{
    PopulateDepartmentsDropDownList();
    return View();
}

[HttpPost]

public ActionResult Create(Course course)
{
    try
    {
        if (ModelState.IsValid)
        {
            db.Courses.Add(course);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        ModelState.AddModelError("", "Unable to save changes. Try again, and if the
problem persists, see your system administrator.");
    }
}
```

```
        PopulateDepartmentsDropDownList(course.DepartmentID);

        return View(course);
    }

public ActionResult Edit(int id)
{
    Course course = db.Courses.Find(id);

    PopulateDepartmentsDropDownList(course.DepartmentID);

    return View(course);
}

[HttpPost]
public ActionResult Edit(Course course)
{
    try
    {
        if (ModelState.IsValid)
        {
            db.Entry(course).State = EntityState.Modified;

            db.SaveChanges();

            return RedirectToAction("Index");
        }
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        ModelState.AddModelError("", "Unable to save changes. Try again, and if the
problem persists, see your system administrator.");
    }
}
```

```

        }

        PopulateDepartmentsDropDownList(course.DepartmentID);

        return View(course);
    }

private void PopulateDepartmentsDropDownList(object selectedDepartment = null)
{
    var departmentsQuery = from d in db.Departments
                           orderby d.Name
                           select d;

    ViewBag.DepartmentID = new SelectList(departmentsQuery, "DepartmentID", "Name",
selectedDepartment);
}

```

Method `PopulateDepartmentsDropDownList` berfungsi untuk mengambil data seluruh deparment dan diurut berdasarkan nama. Method ini memiliki parameter yang memungkinkan untuk memilih item yg akan dipilih ketika dropdown list ditampilkan.

Method `HttpGet Create` memanggil method `PopulateDepartmentsDropDownList` tanpa nilai parameter, karena belum ada department pada course yang baru.

```

public ActionResult Create()
{
    PopulateDepartmentsDropDownList();

    return View();
}

```

Method `HttpGet Edit` menggunakan method `PopulateDepartmentsDropDownList` yang berisi parameter item yang akan dipilih, yaitu nilai ID dari department yang sudah terkait dengan course yang akan diedit.

```

public ActionResult Edit(int id)

```

```
{  
    Course course = db.Courses.Find(id);  
    PopulateDepartmentsDropDownList(course.DepartmentID);  
    return View(course);  
}
```

Method `HttpPost Create` dan `HttpPost Edit` memanggil method

`PopulateDepartmentsDropDownList` yang berisi parameter item yang dipilih, method ini dipanggil menampilkan ulang halaman ketika ada kesalahan terjadi.

```
catch (DataException)  
{  
    //Log the error (add a variable name after DataException)  
    ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem  
persists, see your system administrator.");  
}  
  
PopulateDepartmentsDropDownList(course.DepartmentID);  
return View(course);
```

Kode di atas berfungsi untuk memastikan halaman akan ditampilkan ulang untuk menampilkan pesan kesalahan, dan item department yang telah dipilih pada dropdown list tetap akan terpilih.

Pada file `Views\Course\Create.cshtml`, tambahkan area baru sebelum area **Title** yang akan digunakan oleh user untuk memasukkan nomer course.

```
<div class="editor-label">  
    @Html.LabelFor(model => model.CourseID)  
</div>  
  
<div class="editor-field">  
    @Html.EditorFor(model => model.CourseID)  
    @Html.ValidationMessageFor(model => model.CourseID)  
</div>
```

Pada file `Views\Course>Edit.cshtml`, `Views\Course>Delete.cshtml`, dan `Views\Course\Details.cshtml`, tambahkan area baru sebelum area **Title** untuk menampilkan nomer course. Nilai nomer tersebut tidak dapat diubah karena primary key.

```
<div class="editor-label">  
    @Html.LabelFor(model => model.CourseID)  
</div>  
  
<div class="editor-field">  
    @Html.DisplayFor(model => model.CourseID)  
</div>
```

Jalankan halaman Course Index kemudian klik **Create New**, dan tambahkan data course baru :

The screenshot shows a web browser window with the URL `http://localhost`. The title bar says "Create". The main content area displays the "Contoso University" logo and a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Create" is displayed in large bold letters. A form titled "Course" is shown with fields for Number (containing "1000"), Title (containing "Algebra"), Credits (containing "5"), and Department (a dropdown menu set to "Mathematics"). A blue "Create" button is at the bottom of the form. At the bottom left of the main content area, there is a link "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Create

**Course**

Number  
1000

Title  
Algebra

Credits  
5

Department  
Mathematics ▾

**Create**

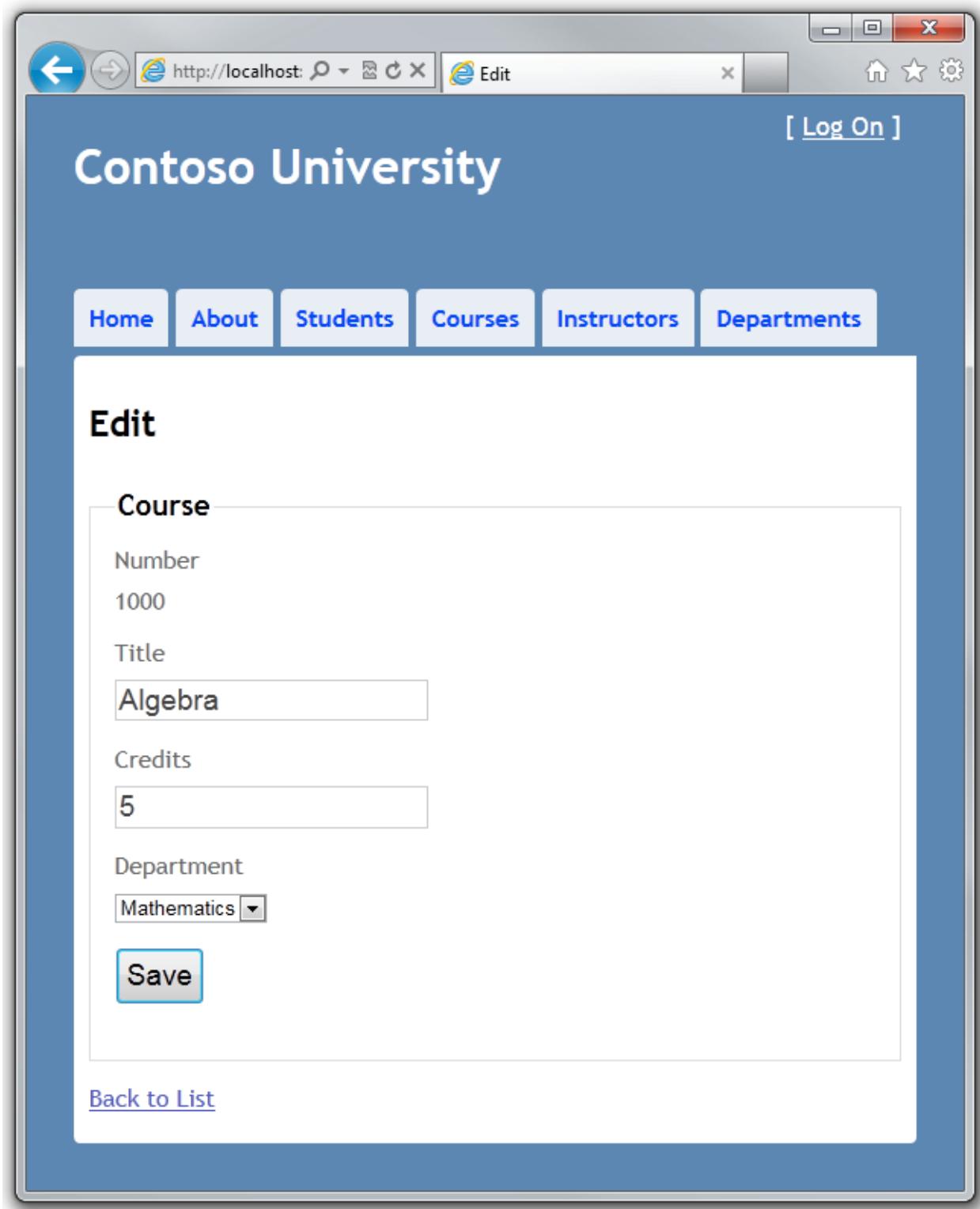
[Back to List](#)

Klik tombol **Create**, maka dapat dilihat course baru ditambahkan pada daftar course. Dapat dilihat nama department pada data yang baru saja ditambahkan sesuai dengan pilihan.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title of the page is "Courses". The main content area displays the heading "Contoso University" and a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Courses" is displayed in large bold letters. A link "Create New" is visible. A table lists eight courses with columns: Number, Title, Credits, and Department. Each row includes "Edit | Details | Delete" links. The data in the table is as follows:

	Number	Title	Credits	Department
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2021	Composition	3	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2042	Literature	4	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1045	Calculus	4	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	3141	Trigonometry	4	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1000	Algebra	5	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1050	Chemistry	3	Engineering
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4022	Microeconomics	3	Economics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4041	Macroeconomics	3	Economics

Klik link **Edit** pada data yang baru saja kita tambahkan.



Ganti nilai yang terdapat pada form, kemudian klik tombol Save. Maka dapat kita lihat perubahan data disimpan, yang hasilnya dapat dilihat pada halaman Course Index.

## Menambah Halaman Edit untuk Instructor

Ketika sedang mengedit data instructor, kita mungkin juga ingin mengupdate data office yang berhubungan dengan instructor tersebut. data tersebut berkaitan dengan entity Instructor dengan entity OfficeAssignment, dimana keduanya mempunyai relasi one-to-zero-or-one, yang artinya kita harus menangani hal-hal berikut ini :

- Jika user menghilangkan nilai dari office assignment yang sebelumnya mempunyai nilai, maka kita harus menghapus entity `OfficeAssignment`.
- Jika user memberikan nilai office assignment yang sebelumnya kosong, maka kita harus membuat entity `OfficeAssignment` baru.
- Jika user mengganti nilai office assignment, maka kita harus mengganti nilai tersebut pada entity `OfficeAssignment` yang telah ada.

Buka file `InstructorController.cs` dan perhatikan method `HttpGet Edit` :

```
public ActionResult Edit(int id)
{
    Instructor instructor = db.Instructors.Find(id);

    ViewBag.InstructorID = new SelectList(db.OfficeAssignments, "InstructorID",
    "Location", instructor.InstructorID);

    return View(instructor);
}
```

Kode yang telah ada menggunakan dropdownlist untuk menampilkan data, jika yang kita inginkan adalah control berupa textbox maka kode yang telah mesti kita ganti dengan kode berikut :

```
public ActionResult Edit(int id)
{
    Instructor instructor = db.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.Courses)
        .Where(i => i.InstructorID == id)
        .Single();

    return View(instructor);
}
```

```
}
```

Pada kode di atas statement `ViewBag` dihilangkan dan menambahkan eager loading pada entity `OfficeAssignment` dan `Course`. (Saat ini kita belum memerlukan `Course`, tapi nanti kita akan membutuhkannya) Kita tidak dapat melakukan eager loading dengan method `Find`, tetapi menggunakan method `Where` dan `Single`.

Ganti method `HttpPost Edit` dengan kode berikut untuk menangani proses update office assignment :

```
[HttpPost]  
  
public ActionResult Edit(int id, FormCollection formCollection)  
{  
  
    var instructorToUpdate = db.Instructors  
        .Include(i => i.OfficeAssignment)  
        .Include(i => i.Courses)  
        .Where(i => i.InstructorID == id)  
        .Single();  
  
    if (TryUpdateModel(instructorToUpdate, "", null, new string[] { "Courses" }))  
    {  
  
        try  
        {  
  
            if  
(String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment.Location))  
            {  
  
                instructorToUpdate.OfficeAssignment = null;  
            }  
  
            db.Entry(instructorToUpdate).State = EntityState.Modified;  
            db.SaveChanges();  
  
            return RedirectToAction("Index");  
        }  
    }  
}
```

```

        }

        catch (DataException)

        {

            //Log the error (add a variable name after DataException)

            ModelState.AddModelError("", "Unable to save changes. Try again, and if the
problem persists, see your system administrator.");

            return View();
        }
    }

    return View(instructorToUpdate);
}

```

Berikut penjelasan dari kode di atas :

- Mengambil entity `Instructor` dari database dengan menggunakan eager loading untuk property navigation `OfficeAssignment` dan `Course`.
- Update nilai entity `Instructor` dengan nilai dari model binder termasuk property navigation `Course`.

```

if (TryUpdateModel(instructorToUpdate, "", null, new string[] { "Courses" }))

```

Jika validasi gagal maka `TryUpdateModel` akan bernilai `false`, dan langsung akan menjalannya kode pada baris terakhir yaitu statement `return View`.

- Jika nilai office location kosong, berikan nilai `Instructor.OfficeAssignment` menjadi null maka hubungan dengan tabel `OfficeAssignment` akan dihapus.

```

if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment.Location))

{
    instructorToUpdate.OfficeAssignment = null;
}

```

- Simpan data ke database.

Pada file `Views\Instructor>Edit.cshtml`, setelah elemen `div` dari area **Hire Date**, tambahkan area baru untuk kebutuhan edit office location :

```
<div class="editor-label">  
    @Html.LabelFor(model => model.OfficeAssignment.Location)  
</div>  
  
<div class="editor-field">  
    @Html.EditorFor(model => model.OfficeAssignment.Location)  
    @Html.ValidationMessageFor(model => model.OfficeAssignment.Location)  
</div>
```

Jalankan halaman kemudian pilih tab **Instructors** dan klik **Edit** pada baris instructor yang diinginkan :

The screenshot shows a web browser window with the URL <http://localhost>. The title bar says "Edit". The page header features the Contoso University logo and a "[ Log On ]" link. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The main content area has a title "Edit" and a form titled "Instructor". The form contains fields for Last Name (Abercrombie), First Name (Kim), Hire Date (3/11/1995), and Office Location (Smith 17). A blue "Save" button is at the bottom of the form. At the bottom of the page is a link "Back to List".

[ Log On ]

# Contoso University

Home    About    Students    Courses    Instructors    Departments

## Edit

Instructor

Last Name

First Name

Hire Date

Office Location

**Save**

[Back to List](#)

Ubah nilai Office Location, kemudian klik tombol **Save**.

The screenshot shows a web browser window with the URL <http://localhost>. The title bar says "Edit". The main content area displays the Contoso University logo and navigation menu with links to Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Edit" is displayed. A form titled "Instructor" contains fields for Last Name (Abercrombie), First Name (Kim), Hire Date (3/11/1995), and Office Location (Harry Potter Closet). The "Office Location" field is highlighted with a red border. A "Save" button is located below the form. At the bottom left, there is a link to "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Edit

Instructor

Last Name

Abercrombie

First Name

Kim

Hire Date

3/11/1995

Office Location

Harry Potter Closet

Save

[Back to List](#)

Pada halaman Index dapat kita lihat data baru yang telah kita isikan. Kita dapat membuka tabel **OfficeAssignment** dengan **Server Explorer** untuk melihat data sebagai berikut.

OfficeAssignment: Query(C:\ContosoU...)

	InstructorID	Location
▶	1	Harry Potter Closet
	2	Gowan 27
	3	Thompson 304
*	NULL	NULL

◀ ▶ | 1 of 3 | ▶ ▶ | | |

Kembali ke halaman **Edit**, dan hilangkan nilai **Office Location** kemudian klik tombol **Save**. Pada halaman Index akan menampilkan nilai kosong sedangkan pada tabel dapat dilihat data tersebut akan dihapus.

OfficeAssignment: Query(C:\ContosoUniv...)

	InstructorID	Location
▶	2	Gowan 27
	3	Thompson 304
*	NULL	NULL

◀ ▶ | 1 of 2 | ▶ ▶ | | |

Kembali ke halaman Edit, dan masukkan nilai baru pada Office Location dan klik Save. Maka pada halaman Index dapat kita lihat nilai tersebut dan pada tabel dapat kita lihat data baru sesuai dengan nilai yang kita masukkan.

OfficeAssignment: Query(C:\ContosoUniv...)

	InstructorID	Location
▶	1	17 Mordor Tower
	2	Gowan 27
	3	Thompson 304
*	NULL	NULL

◀ ▶ | 1 of 3 | ▶ ▶ | | |

# Menambah Course Assignment pada Halaman Edit Instructor

Instructor dapat mengajar lebih dari satu Course. Untuk kebutuhan tersebut, kita dapat melakukan perubahan pada kode yang telah ada untuk menambahkan kemampuan agar user dapat memilih course assignment dengan memilih checkbox seperti pada gambar di bawah ini.

The screenshot shows a web application window titled "Edit" for an "Instructor". The URL in the address bar is "http://localhost:5000/Edit". The page header includes a "Log On" link. The main content area has a title "Contoso University". Below it is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The "Edit" section contains fields for Last Name ("Abercrombie"), First Name ("Kim"), Hire Date ("3/11/1995"), and Office Location ("17 Mordor Tower"). Under the "Courses" heading, there is a table-like structure with two columns. The first column lists course codes and names: 1045 Calculus, 2042 Literature, 4041 Macroeconomics. The second column lists course codes and names: 1050 Chemistry, 3141 Trigonometry, 1000 Algebra. Next to each course name is a checkbox; the "Chemistry" checkbox is checked, while the others are unchecked. At the bottom left of the form is a "Save" button.

Relasi antara entity Course dan Instructor adalah many-to-many, artinya kita tidak memiliki akses langsung ke join tabel dan field foreign key. Sebaliknya, kita akan perlu melakukan menambah dan menghapus entity ke dan dari property navigation `Instructor.Courses`.

Antarmuka di atas memungkinkan kita untuk mengganti course yang ditugaskan kepada instructor dengan cara memilih checkbox yang telah disediakan. Setiap data course dari database akan ditampilkan dalam sebuah checkbox, dan jika ada ada instructor yang ditugaskan pada salah satu course maka akan terlihat centang pada checkbox pada course tersebut. user dapat mencentang atau menghilangkan centang pada checkbox jika ingin mengubah penugasan course.

Untuk menyediakan data pada view berupa list checkbox maka kita membutuhkan class view model. Buat file *AssignedCourseData.cs* di dalam folder *ViewModels* dan ganti dengan kode berikut ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.ViewModels

{
    public class AssignedCourseData
    {
        public int CourseID { get; set; }
        public string Title { get; set; }
        public bool Assigned { get; set; }
    }
}
```

Pada file *InstructorController.cs* pada method *HttpGet Edit*, panggil method baru yang akan memberikan informasi berupa array checkbox menggunakan class view model yang baru seperti yang dilihat pada kode di bawah ini :

```
public ActionResult Edit(int id)
{
    Instructor instructor = db.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.Courses)
        .Where(i => i.InstructorID == id)
```

```

    .Single();

    PopulateAssignedCourseData(instructor);

    return View(instructor);
}

private void PopulateAssignedCourseData(Instructor instructor)
{
    var allCourses = db.Courses;

    var instructorCourses = new HashSet<int>(instructor.Courses.Select(c =>
c.CourseID));

    var viewModel = new List<AssignedCourseData>();

    foreach (var course in allCourses)
    {
        viewModel.Add(new AssignedCourseData
        {
            CourseID = course.CourseID,
            Title = course.Title,
            Assigned = instructorCourses.Contains(course.CourseID)
        });
    }

    ViewBag.Courses = viewModel;
}

```

Kode di atas membaca semua membaca seluruh entity `Course`, kemudian setiap course yang ada akan diperiksa apakah mempunyai hubungan dengan instructor, dengan mencocokan keberadaan course dengan property navigation `Course` milik instructor tersebut. Untuk effesiensi proses look up saat pengecekan course mana saja yang terkait dengan instructor, maka digunakan collection `HashSet`. Property `Assigned` dari course yang terkait dengan instructor akan mempunyai nilai `true`. Property ini akan digunakan oleh view untuk menentukan saat menampilkan checkbox mana saja yang akan ditampilkan dengan status centang. Selanjutnya list akan dikirimkan kedalam property `ViewBag`.

Selanjutnya, menambahkan kode yang akan dieksekusi oleh user saat tombol **Save** diklik. Ganti kode pada **HttpPost Edit** dengan kode berikut ini :

```
[HttpPost]  
  
public ActionResult Edit(int id, FormCollection formCollection, string[]  
selectedCourses)  
{  
  
    var instructorToUpdate = db.Instructors  
        .Include(i => i.OfficeAssignment)  
        .Include(i => i.Courses)  
        .Where(i => i.InstructorID == id)  
        .Single();  
  
    if (TryUpdateModel(instructorToUpdate, "", null, new string[] { "Courses" }))  
    {  
  
        try  
        {  
  
            if  
(String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment.Location))  
            {  
  
                instructorToUpdate.OfficeAssignment = null;  
            }  
  
            UpdateInstructorCourses(selectedCourses, instructorToUpdate);  
  
            db.Entry(instructorToUpdate).State = EntityState.Modified;  
            db.SaveChanges();  
  
            return RedirectToAction("Index");  
        }  
        catch (DataException)
```

```
{  
    //Log the error (add a variable name after DataException)  
  
    ModelState.AddModelError("", "Unable to save changes. Try again, and if the  
problem persists, see your system administrator.");  
  
}  
  
}  
  
PopulateAssignedCourseData(instructorToUpdate);  
  
return View(instructorToUpdate);  
}  
  
  
private void UpdateInstructorCourses(string[] selectedCourses, Instructor  
instructorToUpdate)  
  
{  
  
    if (selectedCourses == null)  
  
    {  
  
        instructorToUpdate.Courses = new List<Course>();  
  
        return;  
  
    }  
  
  
var selectedCoursesHS = new HashSet<string>(selectedCourses);  
  
var instructorCourses = new HashSet<int>  
  
(instructorToUpdate.Courses.Select(c => c.CourseID));  
  
foreach (var course in db.Courses)  
  
{  
  
    if (selectedCoursesHS.Contains(course.CourseID.ToString()))  
  
    {  
  
        if (!instructorCourses.Contains(course.CourseID))  
  
        {  
  
            instructorToUpdate.Courses.Add(course);  
        }  
    }  
}
```

```
        }

    }

    else

    {

        if (instructorCourses.Contains(course.CourseID))

        {

            instructorToUpdate.Courses.Remove(course);

        }

    }

}

}
```

Jika tidak ada checkbox yang dicentang, maka kode di dalam `UpdateInstructorCourses` akan meninisialisasi property navigation `Course` dengan collection kosong.

```
if (selectedCourses == null)

{

    instructorToUpdate.Courses = new List();

    return;

}
```

Jika checkbox untuk sebuah course dipilih tetapi course tersebut tidak berada di dalam property navigation `Instructor.Course`, maka course tersebut akan ditambahkan ke collection di dalam property navigation.

```
if (selectedCoursesHS.Contains(course.CourseID.ToString()))

{

    if (!instructorCourses.Contains(course.CourseID))

    {

        instructorToUpdate.Courses.Add(course);

    }

}
```

```
    }

}
```

Jika course belum dipilih tetapi course berada dalam property navigation Instructor.Course, maka course akan dihapus dari property navigation.

```
else

{
    if (instructorCourses.Contains(course.CourseID))
    {
        instructorToUpdate.Courses.Remove(course);
    }
}
```

Pada file `Views\Instructor>Edit.cshtml`, tambahkan area Course dengan array checkbox dengan menambahkan kode berikut ini setelah elemen div :

```
<div class="editor-field">

    <table>
        <tr>
            @{
                int cnt = 0;
                List<ContosoUniversity.ViewModels.AssignedCourseData> courses =
                ViewBag.Courses;
            }

            foreach (var course in courses) {
                if (cnt++ % 3 == 0) {
                    @: </tr> <tr>
                }
                @: <td>
```

```

<input type="checkbox"
       name="selectedCourses"
       value="@course.CourseID"
       @(Html.Raw(course.Assigned ? "checked=\"checked\"" :
"")) />

        @course.CourseID @: @course.Title
        @:</td>
    }
    @:</tr>
}
</table>
</div>

```

Kode tersebut di atas akan membuat tabel HTML yang terdiri atas tiga kolom yang berfungsi untuk menampilkan elemen checkbox, nomer course dan title course. Seluruh elemen checkbox mempunyai nilai atribut `name` yang sama yaitu “`selectedCourses`”, hal ini menandakan bahwa model binder akan menangani sebagai group. Sedangkan atribut `value` akan berisi nilai dari `CourseID`. Ketika halaman di-post, model binder akan mengirimkan array ke controller yang terdiri atas nilai-nilai `CourseID` untuk checkbox yang dicentang saja.

Ketika checkbox ditampilkan, course yang telah terkait dengan instructor akan mempunyai atribut `checked`.

Setelah mengubah penugasan course, kita mungkin ingin untuk memverifikasi perubahan ketika kembali ke halaman Index. Untuk keperluan tersebut, kita perlu menambahkan kolom ke tabel yang ada di halaman tersebut. dalam kasus ini kita tidak akan menggunakan objek ViewBag, karena informasi yang ingin ditampilkan sudah ada pada property navigation Course dari entity Instructor yang dikirimkan ke view sebagai model.

Pada file `Views\Instructor\Index.cshtml`, tambahkan heading `<th>Courses</th>` dan heading `<th>Office</th>`,seperti kode di bawah ini :

```

<tr>
    <th></th>
    <th>Last Name</th>

```

```
<th>First Name</th>
<th>Hire Date</th>
<th>Office</th>
<th>Courses</th>
</tr>
```

Setelah itu tambahkan kode berikut :

```
<td>
@{
    foreach (var course in item.Courses)
    {
        @course.CourseID @: @course.Title <br />
    }
}</td>
```

Jalankan halaman **Instructor Index** untuk menampilkan course yang terkait dengan instructor :

The screenshot shows a web browser window with the URL <http://localhost:43551> in the address bar. The title bar says "Instructors". The page header features the Contoso University logo and a "Log On" link. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Instructors" and contains a "Create New" link. Below this is a table listing five instructors:

	Last Name	First Name	Hire Date	Office	Courses
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Abercrombie	Kim	3/11/1995	17 Mordor Tower	1050 Chemistry
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Fakhouri	Fadi	7/6/2002	Govan 27	1050 Chemistry
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Harui	Roger	7/1/1998	Thompson 304	4022 Microeconomics 4041 Macroeconomics
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Kapoor	Candace	1/15/2001		1045 Calculus 2021 Composition 2042 Literature 3141 Trigonometry
<a href="#">Select</a>   <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Zheng	Roger	2/12/2004		

Klik **Edit** pada baris instructor yang diinginkan :

The screenshot shows a web browser window with the URL <http://localhost>. The title bar says "Edit". The page header features the Contoso University logo and a "[ Log On ]" link. A navigation menu at the top includes links for Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Edit" and contains a form for editing an "Instructor". The form fields include "Last Name" (Abercrombie), "First Name" (Kim), "Hire Date" (3/11/1995), and "Office Location" (17 Mordor Tower). Below these fields is a "Courses" section containing a grid of checkboxes. The grid has three columns and four rows of data:

	1045 Calculus	1050 Chemistry	2021 Composition
<input type="checkbox"/>	Literature	<input checked="" type="checkbox"/> 3141 Trigonometry	<input type="checkbox"/> 4022 Microeconomics
<input type="checkbox"/>	Macroeconomics	<input type="checkbox"/> 1000 Algebra	

A blue "Save" button is located below the grid. At the bottom of the form is a link "Back to List".

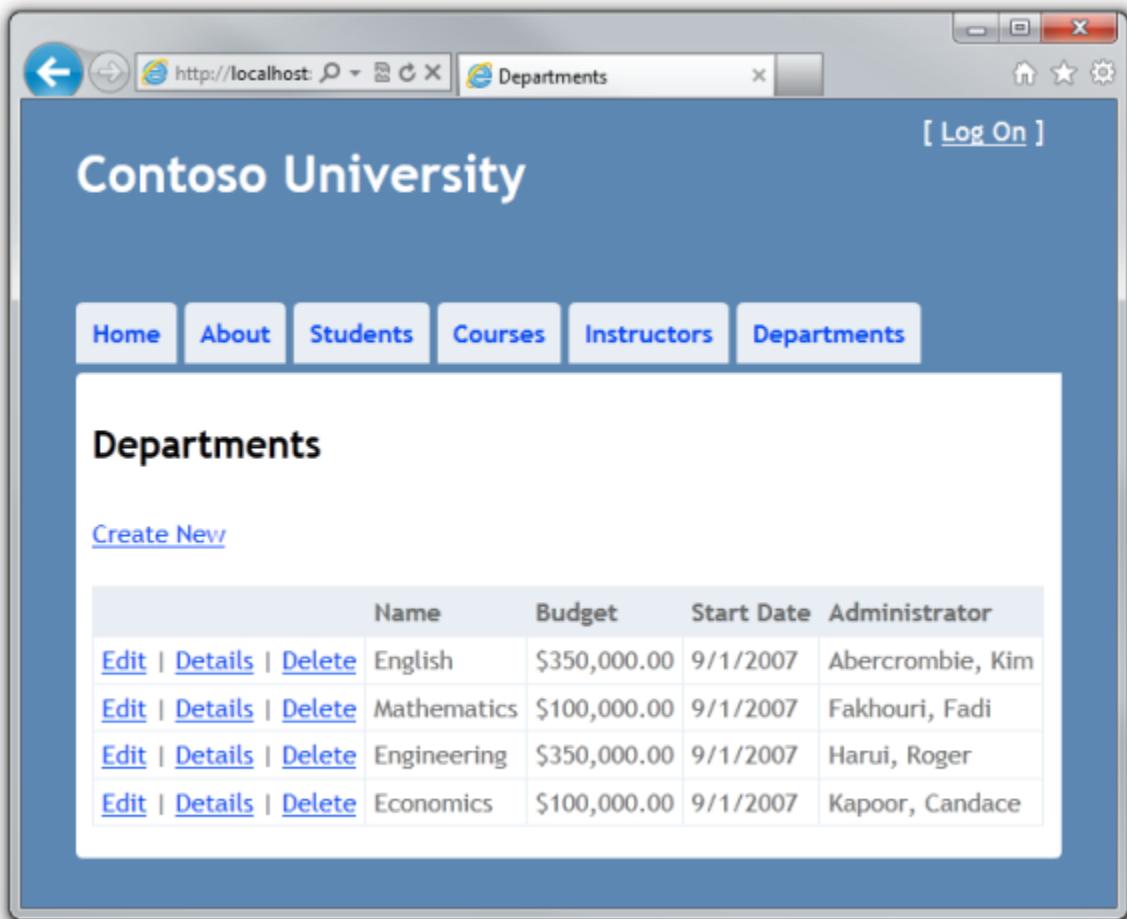
Ubah beberapa nilai pada checkbox pada course, kemudian klik tombol **Save**. Maka perubahan itu akan dapat dilihat pada halaman Index.

Sekarang telah lengkap tutorial pengenalan untuk bekerja dengan data menggunakan Entity Framework. pada tutorial yang telah ada sampai bagian ini telah diajarkan seluruh operasi CRUD, tetapi kita belum

menangani masalah concurrency. Pada tutorial berikutnya akan diperkenalkan topik yang membahas concurrency.

# Penanganan Concurrency dengan Entity Framework pada Aplikasi ASP.NET MVC

Tutorial ini akan memaparkan penanganan concurrency dengan Entity Framework. Pada tutorial ini kita akan membuat halaman web yang menampilkan data Department, dan akan digunakan penanganan error concurrency pada halaman yang menangani aksi edit dan menghapus data. Pada gambar berikut diperlihatkan halaman Index dan Delete, termasuk pesan yang akan ditampilkan jika terjadi konflik concurrency.



The screenshot shows a web browser window with the URL <http://localhost:43551/>. The title bar says "Edit". The main content area displays the Contoso University logo and navigation links for Home, About, Students, Courses, Instructors, and Departments. Below this, a heading "Edit" is followed by a red warning message: "The record you attempted to edit was modified by another user after you got the original value. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again. Otherwise click the Back to List hyperlink." A form titled "Department" contains fields for Name (English), Budget (999000.00), StartDate (9/1/2007), and Administrator (Abercrombie, Kim). A "Save" button is present at the bottom of the form. At the very bottom of the page is a link "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Edit

• The record you attempted to edit was modified by another user after you got the original value. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again. Otherwise click the Back to List hyperlink.

**Department**

Name

Budget  
 Current value: \$0.00

StartDate

Administrator

**Save**

[Back to List](#)

## Konflik Concurrency

Konflik concurrency terjadi ketika seorang user sedang mengedit data dan kemudian ada user lain yang mengedit data yang sama kemudian menyimpan data tersebut sebelum user pertama menyimpan datanya ke database. Apabila kita tidak melakukan setting pada Entity Framework untuk mendeteksi konflik seperti itu maka yang akan terjadi adalah user yang melakukan data yang akan disimpan ke dalam database adalah data yang disimpan paling akhir. Pada banyak aplikasi, resiko seperti ini masih bisa diterima jika pengguna aplikasi tidak banyak, update yang dilakukan masih sedikit atau jika kasus overwrite seperti kasus di atas bukan hal penting. Jika kasusnya seperti itu maka kita tidak perlu menangani konflik concurrency.

## Pessimistic Concurrency (Locking)

Salah satu cara untuk menangani konflik concurrency adalah dengan pessimistic concurrency yaitu dengan cara mengunci database. Sebagai contoh, sebelum kita membaca data pada database, maka terlebih dahulu akan melakukan permintaan untuk mengunci row pada database agar hanya bisa dibaca saja atau tertutup untuk akses operasi update.

Pengelolaan penguncian seperti itu mempunyai kekurangan, karena membuat program menjadi lebih rumit dan membutuhkan resource database yang tidak sedikit, dan hal ini tentu akan berakibat pada performansi jika jumlah pengguna aplikasi meningkat. Karena hal itu tidak semua database management system yang mendukung fitur ini. Entity Framework tidak memberikan dukungan built-in untuk fitur ini, dan pada tutorial ini tidak menunjukkan kepada kita bagaimana untuk menerapkannya.

## Optimistic Concurrency

Optimistic concurrency mengijinkan konflik concurrency terjadi, tetapi jika hal itu terjadi akan ada aksi-aksi yang dilakukan untuk menanganinya. Sebagai contoh jika John menjalankan halaman Edit Department dan menganti nilai **Budget** untuk English Department dari \$350,000.00 menjadi \$100,000.00.

[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Edit

**Department**

Name

Budget

StartDate

Administrator

[Back to List](#)

Sebelum John mengklik tombol **Save**, Jane menjalankan halaman yang sama dan mengedit data yang sama dan mengubah nilai Start Date dari 9/1/1970 menjadi 1/1/1999.

[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Edit

**Department**

Name

Budget

StartDate  
 1/1/1999

Administrator

**Save**

[Back to List](#)

John mengklik tombol **Save** terlebih dahulu dan melihat hasil yang yang telah dia ubah sesuai dengan data yang telah John masukkan. Kemudian Jane mengklik tombol **Save**. Apa yang terjadi setelah itu tergantung bagaimana kita menangani konflik concurrency ini. Berikut ini beberapa pilihan untuk menangani konflik ini :

- Kita dapat memeriksa property-property apa saja yang dimodifikasi oleh user dan hanya mengupdate nilai kolom itu saja ke database. Pada contoh kasus seperti di atas, dipastikan tidak ada data yang akan hilang, karena yang diupdate oleh kedua user tersebut adalah nilai dari dua property yang berbeda. Metode ini dapat mengurangi kehilangan data jika konflik terjadi, tetapi tidak dapat menghindari jika proses update terjadi pada property yang sama. Metode seperti ini tidak praktis jika digunakan pada aplikasi web, karena akan membutuhkan perhatian kita untuk menangani state dalam jumlah besar untuk menyimpan seluruh nilai awal dan nilai baru. Pengelolaan sejumlah besar state seperti itu tentu saja akan berdampak pada performansi aplikasi karena hal itu membutuhkan resource server atau berdampak pada halaman web itu sendiri karena akan menggunakan banyak hidden field untuk menyimpan nilai-nilai tersebut.
- Kita dapat mengijinkan perubahan yang dilakukan oleh Jane menimpa nilai-nilai yang telah dimasukkan oleh John. Maka data terbaru yang disimpan adalah sebagai berikut, yaitu nilai 1/1/1999 dan \$350,000.00. Metode ini disebut sebagai skenario *Client Wins* atau *Last in Win*. Metode seperti ini sudah diterapkan secara otomatis walaupun kita tidak menangani konflik concurrency.
- Kita dapat mencegah agar perubahan yang Jane lakukan terupdate ke database. Kita bisa memberikan pesan kesalahan, dan menampilkan data yang ada sekarang dan memberikan pilihan kepada Jane jika dia ingin tetap melakukan update. Skenario ini disebut *Store Wins*. Pada tutorial ini kita akan menggunakan skenario ini. Metode ini memastikan tidak ada perubahan yang menimpa tanpa pesan peringatan terlebih dahulu.

## Mendeteksi Konflik Concurrency

Kita dapat menyelesaikan konflik dengan menangani exception

`OptimisticConcurrencyException` yang dilemparkan oleh Entity Framework. Oleh karena itu kita harus melakukan konfigurasi pada database dan data model secara tepat. Berikut ini adalah pilihan yang dapat kita lakukan untuk mengaktifkan deteksi konflik yang terjadi :

- Pada tabel di database, terdapat kolom pelacakan yang dapat digunakan untuk menentukan kapan sebuah row diubah. Kita dapat mengkonfigurasi Entity Framework untuk mengikuti sertakan kolom tersebut dalam `Where` saat perintah `Update` dan `Delete` dijalankan. Tipe data yang akan digunakan pada kolom tersebut adalah `timestamp`, walaupun nilainya belum tentu berisi tanggal atau waktu. Tetapi dapat berisi nomor urut yang bertambah setiap row tersebut diperbarui. (Dalam versi SQL terbaru terdapat tipe data `rowversion` yang dapat digunakan untuk kebutuhan tersebut.) Pada perintah `Update` dan `Delete` akan dilakukan pemeriksaan kolom tersebut dengan menggunakan klausula `Where`, jika nilai kolom tersebut berbeda antara nilai awal dengan nilai yang tersimpan pada database sekarang maka perintah `Update` dan `Delete` tidak akan bisa dilakukan, karena row yang dimaksud tidak ditemukan. Jika Entity

Framework tidak menemukan row tersebut saat proses `Update` atau `Delete` maka hal itu dianggap telah terjadi konflik concurrency.

- Kita bisa mengkonfigurasi agar Entity Framework menyertakan nilai awal dari setiap kolom dari row pada klausanya `Where` saat menjalankan perintah `Update` dan `Delete`.

Opsi pertama adalah jika ada perubahan pada row sejak pertama kali dibaca maka klausanya `Where` tidak akan mengembalikan row untuk diupdate, yang artinya terdapat konflik concurrency. Tetapi jika tabel tersebut mempunyai banyak kolom, maka akan menghasilkan klausanya `Where` yang sangat panjang, dan perlu dilakukan penanganan state yang sangat besar. Hal ini tentu saja akan berimbas pada performansi di sisi server dan halaman web itu sendiri. Solusi dengan pendekatan seperti ini tidak dianjurkan dan tidak akan digunakan pada tutorial ini.

Pada entity Department akan ditambahkan property pelacakan, membuat controller dan view.

**Note :**

Jika kita ingin melakukan implementasi concurrency tanpa menggunakan kolom pelacakan, maka kita harus menandai seluruh property non-primary-key pada entity untuk pelacakan concurrency dengan menambahkan atribut `ConcurrencyCheck`. Perubahan tersebut membuat Entity Framework pengecekan seluruh kolom pada klausanya `Where` saat menjalankan perintah `Update`.

## Menambahkan Property Tracking pada Entity Department

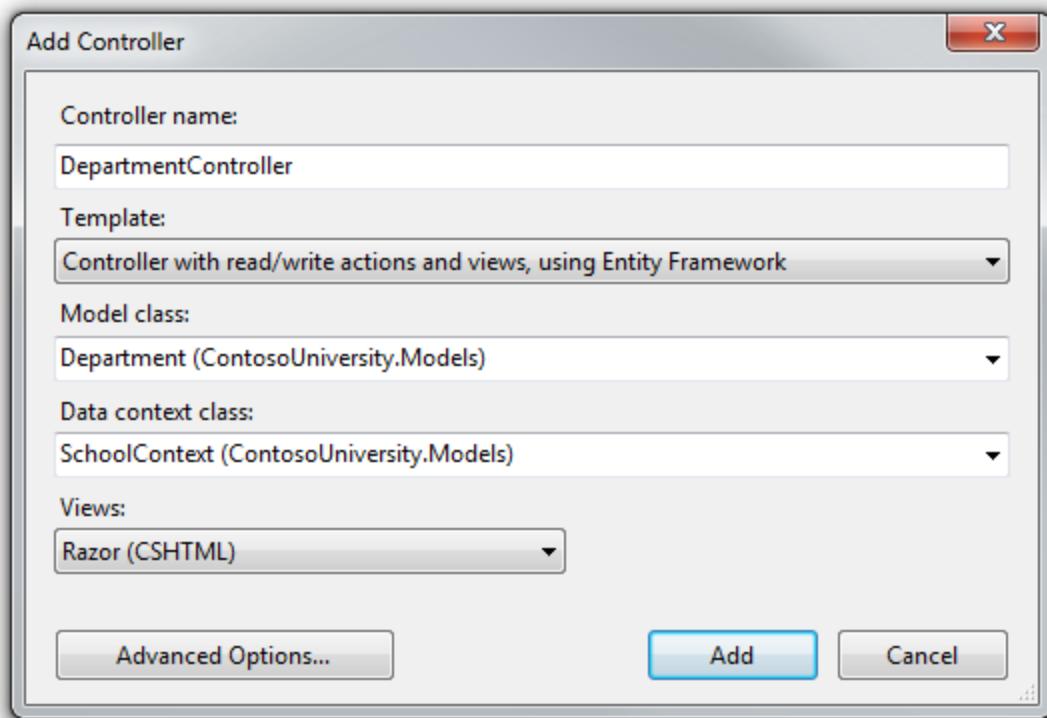
Pada file `Models\Department.cs`, tambahkan property pelacakan seperti berikut :

```
[Timestamp]  
public Byte[] Timestamp { get; set; }
```

Atribut TimeStamp menyatakan bahwa kolom akan digunakan pada klausanya `Where` pada perintah `Update` dan `Delete` untuk dikirim ke database.

## Membuat Controller Department

Buat controller Department dengan setting seperti berikut :



Pada file *Controllers\DepartmentController.cs*, tambahkan statement `using` seperti pada kode berikut ini :

```
using System.Data.Entity.Infrastructure;
```

Ubah seluruh “LastName” yang ada di file tersebut menjadi “FullName”, sehingga dropdown list untuk menampilkan administrator pada department akan menampilkan nama lengkap dari instructor.

Ganti kode pada method `HttpPost Edit` dengan kode berikut :

```
[HttpPost]  
  
public ActionResult Edit(Department department)  
  
{  
  
    try  
  
    {  
  
        if (ModelState.IsValid)
```

```
{  
    db.Entry(department).State = EntityState.Modified;  
  
    db.SaveChanges();  
  
    return RedirectToAction("Index");  
}  
}  
  
catch (DbUpdateConcurrencyException ex)  
{  
  
    var entry = ex.Entries.Single();  
  
    var databaseValues = (Department)entry.GetDatabaseValues().ToObject();  
  
    var clientValues = (Department)entry.Entity;  
  
    if (databaseValues.Name != clientValues.Name)  
  
        ModelState.AddModelError("Name", "Current value: "  
            + databaseValues.Name);  
  
    if (databaseValues.Budget != clientValues.Budget)  
  
        ModelState.AddModelError("Budget", "Current value: "  
            + String.Format("{0:c}", databaseValues.Budget));  
  
    if (databaseValues.StartDate != clientValues.StartDate)  
  
        ModelState.AddModelError("StartDate", "Current value: "  
            + String.Format("{0:d}", databaseValues.StartDate));  
  
    if (databaseValues.InstructorID != clientValues.InstructorID)  
  
        ModelState.AddModelError("InstructorID", "Current value: "  
            + db.Instructors.Find(databaseValues.InstructorID).FullName);  
  
    ModelState.AddModelError(string.Empty, "The record you attempted to edit "  
        + "was modified by another user after you got the original value. The "  
        + "edit operation was canceled and the current values in the database "  
        + "have been displayed. If you still want to edit this record, click "  
        + "the Save button again. Otherwise click the Back to List hyperlink.");  
}
```

```

        department.Timestamp = databaseValues.Timestamp;

    }

    catch (DataException)
    {

        //Log the error (add a variable name after Exception)

        ModelState.AddModelError(string.Empty, "Unable to save changes. Try again, and
if the problem persists contact your system administrator.");
    }

    ViewBag.InstructorID = new SelectList(db.Instructors, "InstructorID", "FullName",
department.InstructorID);

    return View(department);
}

```

View akan menyimpan nilai timestamp awal pada hidden field. Ketika model binder membuat instance `department`, objek tersebut akan memiliki property nilai asli atau nilai seperti di awa untuk property `TimeStamp` dan nilai baru untuk masing-masing property yang lain, yang nilainya sesuai dengan yang dimasukkan oleh user pada halaman Edit. Kemudian Entity Framework akan membuat perintah SQL `UPDATE` yang berisi klausa `WHERE` yang mencari row yang mempunyai nilai `TimeStamp` asli.

Jika tidak ada row yang terpengaruh dengan dengan perintah `UPDATE` tersebut maka Entity Framework akan melemparkan exception `DbUpdateConcurrencyException`, dan blok `catch` akan mengeksekusi kode berikut :

```

var entry = ex.Entries.Single();

var databaseValues = (Department)entry.GetDatabaseValues().ToObject();

var clientValues = (Department)entry.Entity;

```

Kode berikutnya berfungsi untuk menambahkan pesan error yang memberikan informasi jika terdapat perbedaan nilai pada setiap kolom yang ada di database dengan nilai pada halaman Edit :

```
if (databaseValues.Name != clientValues.Name)

    ModelState.AddModelError("Name", "Current value: " + databaseValues.Name);

// ...
```

Pesan kesalahan yang lebih panjang akan memberikan informasi tentang apa yang telah terjadi dan petunjuk hal apa yang mesti dilakukan oleh user :

```
ModelState.AddModelError(string.Empty, "The record you attempted to edit "

+ "was modified by another user after you got the original value. The "

+ "edit operation was canceled and the current values in the database "

+ "have been displayed. If you still want to edit this record, click "

+ "the Save button again. Otherwise click the Back to List hyperlink.");
```

Terakhir, nilai `TimeStamp` akan diset dengan nilai terbaru dari database. Nilai tersebut akan disimpan pada hidden field saat halaman Edit ditampilkan ulang.

Pada file `Views\Department>Edit.cshtml`, tambahkan hidden field untuk menyimpan nilai property `TimeStamp` :

```
@Html.HiddenFor(model => model.Timestamp)
```

Pada file `Views\Department\Index.cshtml`, ganti kode yang ada dengan kode berikut ini.

```
@model IEnumerable<ContosoUniversity.Models.Department>

@{
    ViewBag.Title = "Departments";
}

<h2>Departments</h2>
```

```
<p>

    @Html.ActionLink("Create New", "Create")

</p>

<table>

    <tr>

        <th></th>

        <th>Name</th>

        <th>Budget</th>

        <th>Start Date</th>

        <th>Administrator</th>

    </tr>

    @foreach (var item in Model) {

        <tr>

            <td>

                @Html.ActionLink("Edit", "Edit", new { id=item.DepartmentID }) |

                @Html.ActionLink("Details", "Details", new { id=item.DepartmentID }) |

                @Html.ActionLink("Delete", "Delete", new { id=item.DepartmentID })

            </td>

            <td>

                @Html.DisplayFor(modelItem => item.Name)

            </td>

            <td>

                @Html.DisplayFor(modelItem => item.Budget)

            </td>

            <td>

                @Html.DisplayFor(modelItem => item.StartDate)

            </td>

        </tr>
    }
}
```

```

        </td>
    <td>
        @Html.DisplayFor(modelItem => item.Administrator.FullName)
    </td>
</tr>
}

</table>

```

## Testing Penanganan Optimistic Concurrency

Jalankan halaman **Department** :

	Name	Budget	Start Date	Administrator
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	English	\$350,000.00	9/1/2007	Abercrombie, Kim
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Mathematics	\$100,000.00	9/1/2007	Fakhouri, Fadi
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Engineering	\$350,000.00	9/1/2007	Harui, Roger
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Economics	\$100,000.00	9/1/2007	Kapoor, Candace

Klik link **Edit**, kemudian buka browser baru dan lakukan hal yang sama seperti di atas. Maka kedua halaman tersebut akan menampilkan informasi yang sama.

The screenshot shows a web browser window with the URL <http://localhost:43551/>. The page title is "Contoso University". A navigation bar at the top includes links for Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Edit" and contains a form for editing a "Department". The form fields are: Name (English), Budget (350000.00), StartDate (9/1/2007), and Administrator (Abercrombie, Kim). A "Save" button is present at the bottom of the form. Below the form is a link to "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Edit

**Department**

Name

Budget

StartDate

Administrator

**Save**

[Back to List](#)

Pada browser pertama, ubah nilai yang diinginkan kemudian klik tombol **Save**.

The screenshot shows a web browser window with the URL <http://localhost:43551/>. The title bar says "Edit". The main content area displays the Contoso University logo and navigation menu with links to Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Edit" is displayed. A form titled "Department" contains fields for Name (with value "English"), Budget (with value "0.00" highlighted by a red box), StartDate (with value "9/1/2007"), and Administrator (with value "Abercrombie, Kim"). A "Save" button is at the bottom of the form. At the very bottom of the page is a link "Back to List".

[ Log On ]

# Contoso University

Home About Students Courses Instructors Departments

## Edit

**Department**

Name  
English

Budget  
0.00

StartDate  
9/1/2007

Administrator  
Abercrombie, Kim ▾

**Save**

[Back to List](#)

Maka dapat dilihat browser akan menampilkan data terbaru :

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title bar says "Departments". The main content area displays the "Contoso University" logo and a navigation menu with links to Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Departments" is highlighted. A link "Create New" is visible. A table lists four departments with columns for Name, Budget, Start Date, and Administrator. Each row has "Edit | Details | Delete" links.

	Name	Budget	Start Date	Administrator
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	English	\$0.00	9/1/2007	Abercrombie, Kim
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Mathematics	\$100,000.00	9/1/2007	Fakhouri, Fadi
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Engineering	\$350,000.00	9/1/2007	Harui, Roger
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Economics	\$100,000.00	9/1/2007	Kapoor, Candace

Pada browser kedua, ganti nilai pada field yang sama dengan nilai yang berbeda.

The screenshot shows a web browser window with the URL `http://localhost:43551/`. The title bar says "[ Log On ]". The main content area displays the Contoso University logo and navigation menu with links to Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Edit" is displayed. A form titled "Department" is shown, containing fields for Name (value: English), Budget (value: 999000.00, highlighted with a red border), StartDate (value: 9/1/2007), and Administrator (value: Abercrombie, Kim). A "Save" button is at the bottom of the form. At the bottom of the page, there is a link "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Edit

**Department**

Name

Budget

StartDate

Administrator

**Save**

[Back to List](#)

Kemudian klik tombol **Save**, maka akan dapat kita lihat pesan error seperti berikut.

The screenshot shows a web browser window with the URL <http://localhost:43551/>. The title bar says "Edit". The main content area displays the Contoso University logo and navigation links for Home, About, Students, Courses, Instructors, and Departments. Below this, a heading "Edit" is followed by a red warning message: "The record you attempted to edit was modified by another user after you got the original value. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again. Otherwise click the Back to List hyperlink." A form titled "Department" contains fields for Name (English), Budget (999000.00), StartDate (9/1/2007), and Administrator (Abercrombie, Kim). A "Save" button is present at the bottom of the form. At the very bottom of the page is a link "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Edit

• The record you attempted to edit was modified by another user after you got the original value. The edit operation was canceled and the current values in the database have been displayed. If you still want to edit this record, click the Save button again. Otherwise click the Back to List hyperlink.

**Department**

Name

Budget  
 Current value: \$0.00

StartDate

Administrator

**Save**

[Back to List](#)

Klik Save lagi, maka nilai yang kita masukkan pada browser kedua akan disimpan ke database dan halaman Index akan ditampilkan.

	Name	Budget	Start Date	Administrator
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	English	\$999,000.00	9/1/2007	Abercrombie, Kim
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Mathematics	\$100,000.00	9/1/2007	Fakhouri, Fadi
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Engineering	\$350,000.00	9/1/2007	Harui, Roger
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Economics	\$100,000.00	9/1/2007	Kapoor, Candace

## Menambahkan Halaman Delete

Untuk proses penghapusan data, Entity Framework akan mendeteksi konflik concurrency sama seperti sebelumnya. Ketika method `HttpGet Delete` menampilkan view konfirmasi, view akan menyimpan nilai asli `TimeStamp` pada hidden field. Ketika Entity Framework membuat perintah SQL `DELETE`, perintah tersebut akan berisi klausa `WHERE` dengan nilai asli `TimeStamp`. Jika hasil perintah tersebut tidak berpengaruh pada row, karena row dengan nilai `TimeStamp` asli tidak ditemukan, maka exception concurrency akan dilemparkan dan method `HttpGet Delete` akan dipanggil dengan error flag diset dengan nilai `true`, agar ditampilkan kembali halaman konfirmasi dengan tambahan pesan error.

Pada file `DepartmentController.cs`, ganti method `HttpGet Delete` dengan kode berikut ini :

```
public ActionResult Delete(int id, bool? concurrencyError)
{
```

```

if (concurrencyError.GetValueOrDefault())
{
    ViewBag.ConcurrencyErrorMessage = "The record you attempted to delete "
        + "was modified by another user after you got the original values. "
        + "The delete operation was canceled and the current values in the "
        + "database have been displayed. If you still want to delete this "
        + "record, click the Delete button again. Otherwise "
        + "click the Back to List hyperlink.";
}

Department department = db.Departments.Find(id);
return View(department);
}

```

Method tersebut menerima parameter yang menunjukkan apakah halaman akan ditampilkan ulang setelah terjadi konflik concurrency. Jika flag bernilai `true`, maka pesan kesalahan akan dikirim ke view dengan menggunakan property `ViewBag`.

Ganti kode method `HttpGet Delete` (bernama `DeleteConfirmed`) dengan kode berikut :

```

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(Department department)
{
    try
    {
        db.Entry(department).State = EntityState.Deleted;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    catch (DbUpdateConcurrencyException)

```

```
{  
    return RedirectToAction("Delete",  
        new System.Web.Routing.RouteValueDictionary { { "ConcurrencyError", true }  
    });  
}  
  
catch (DataException)  
{  
    //Log the error (add a variable name after Exception)  
  
    ModelState.AddModelError(string.Empty, "Unable to save changes. Try again, and  
    if the problem persists contact your system administrator.");  
  
    return View(department);  
}  
}
```

Pada kode sebelum diganti, method ini hanya akan menerima sebuah ID saja :

```
public ActionResult DeleteConfirmed(int id)
```

Kemudian kode tersebut diganti dengan kode di bawah ini. Kode di bawah ini memungkinkan kita untuk mengakses nilai property `TimeStamp` disamping record key.

```
public ActionResult DeleteConfirmed(Department department)
```

Jika error concurrency ditemukan, kode akan menampilkan ulang halaman konfirmasi Delete dan memberikan flag yang menunjukkan untuk menampilkan pesan error concurrency.

Pada file Views\Department\Delete.cshtml, ganti kode yang sudah ada menjadi seperti berikut :

```
@model ContosoUniversity.Models.Department

@{
    ViewBag.Title = "Delete";
}

<h2>Delete</h2>

<p class="error">@ViewBag.ConcurrencyErrorMessage</p>

<h3>Are you sure you want to delete this?</h3>

<fieldset>
    <legend>Department</legend>

    <div class="display-label">
        @Html.LabelFor(model => model.Name)
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>

    <div class="display-label">
        @Html.LabelFor(model => model.Budget)
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Budget)
    </div>
```

```

<div class="display-label">
    @Html.LabelFor(model => model.StartDate)
</div>

<div class="display-field">
    @Html.DisplayFor(model => model.StartDate)
</div>

<div class="display-label">
    @Html.LabelFor(model => model.InstructorID)
</div>

<div class="display-field">
    @Html.DisplayFor(model => model.Administrator.FullName)
</div>

</fieldset>

@using (Html.BeginForm()) {
    @Html.HiddenFor(model => model.DepartmentID)
    @Html.HiddenFor(model => model.Timestamp)

    <p>
        <input type="submit" value="Delete" /> |
        @Html.ActionLink("Back to List", "Index")
    </p>
}

}

```

Kode di atas menambahkan pesan error antara heading h2 dan h3 :

```
<p class="error">@ViewBag.ConcurrencyErrorMessage</p>
```

Pada area `Administrator`, ganti `LastName` dengan `FullName` :

```
<div class="display-label">  
    @Html.LabelFor(model => model.InstructorID)  
</div>  
  
<div class="display-field">  
    @Html.DisplayFor(model => model.Administrator.FullName)  
</div>
```

Terakhir, tambahkan hidden field untuk property `DepartmentID` dan `TimeStamp` setelah statement `Html.BeginForm`:

```
@Html.HiddenFor(model => model.DepartmentID)  
@Html.HiddenFor(model => model.Timestamp)
```

Jalankan halaman Department Index dan buka browser kedua untuk mengakses URL tersebut juga.

Pada browser pertama, klik **Edit** pada data department dan ubah salah satu nilainya tapi tombol **Save** jangan diklik dulu :

[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Edit

**Department**

Name

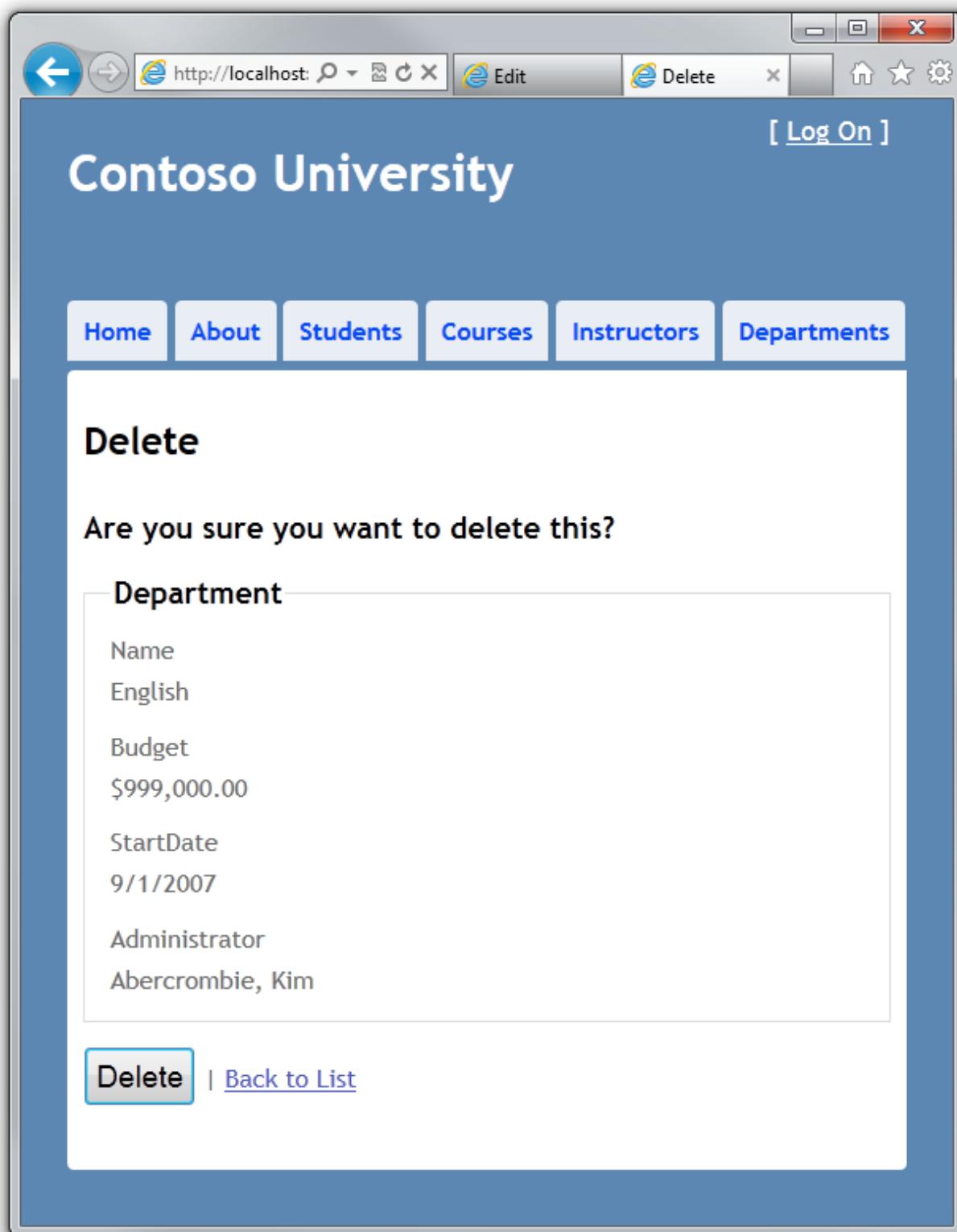
Budget

StartDate

Administrator

[Back to List](#)

Pada browser kedua, pilih Delete pada data department yang sama seperti pada browser pertama, kemudian halaman konfirmasi akan ditampilkan :



Klik Save pada browser pertama maka dapat dilihat perubahan pada halaman Index :

The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost>. The title bar says "Departments". The main content area displays the "Contoso University" logo and a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Departments" is highlighted in bold. A link "Create New" is visible. A table lists four departments with columns: Name, Budget, Start Date, and Administrator. Each row has "Edit | Details | Delete" links.

	Name	Budget	Start Date	Administrator
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	English	\$100,000.00	9/1/2007	Abercrombie, Kim
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Mathematics	\$100,000.00	9/1/2007	Fakhouri, Fadi
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Engineering	\$350,000.00	9/1/2007	Harui, Roger
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	Economics	\$100,000.00	9/1/2007	Kapoor, Candace

Sekarang klik tombol Delete pada browser kedua, maka kita akan melihat pesan error concurrency seperti di bawah ini.

The screenshot shows a web browser window with the following details:

- Address Bar:** http://localhost
- Toolbar:** Includes Back, Forward, Stop, Refresh, Home, Favorites, and Settings buttons.
- Header:** [ Log On ]
- Page Title:** Contoso University
- Navigation Menu:** Home, About, Students, Courses, Instructors, Departments (the 'Departments' menu item is highlighted).
- Main Content Area:**
  - Section Header:** Delete
  - Text Message:** The record you attempted to delete was modified by another user after you got the original values. The delete operation was canceled and the current values in the database have been displayed. If you still want to delete this record, click the Delete button again. Otherwise click the Back to List hyperlink.
  - Confirmation Question:** Are you sure you want to delete this?
  - Form Data:** A table showing department details:

Department	
Name	English
Budget	\$100,000.00
StartDate	9/1/2007
Administrator	Abercrombie, Kim
  - Buttons:** A blue "Delete" button and a link to "Back to List".

Jika kita klik tombol Delete lagi, maka data akan dihapus dan akan ditampilkan halaman Index.

Sebagai informasi mengenai skenario-skenario untuk menangani concurrency dapat dilihat post dari blog berikut ini Optimistic Concurrency Patterns

(<http://blogs.msdn.com/b/adonet/archive/2011/02/03/using-dbcontext-in-ef-feature-ctp5-part-9-optimistic-concurrency-patterns.aspx>) dan Working with Property Values

(<http://blogs.msdn.com/b/adonet/archive/2011/01/30/using-dbcontext-in-ef-feature-ctp5-part-5-working-with-property-values.aspx>). Tutorial berikutnya akan memperlihatkan implementasi table-per-hierarchy inheritance pada entity Instructor dan Student.

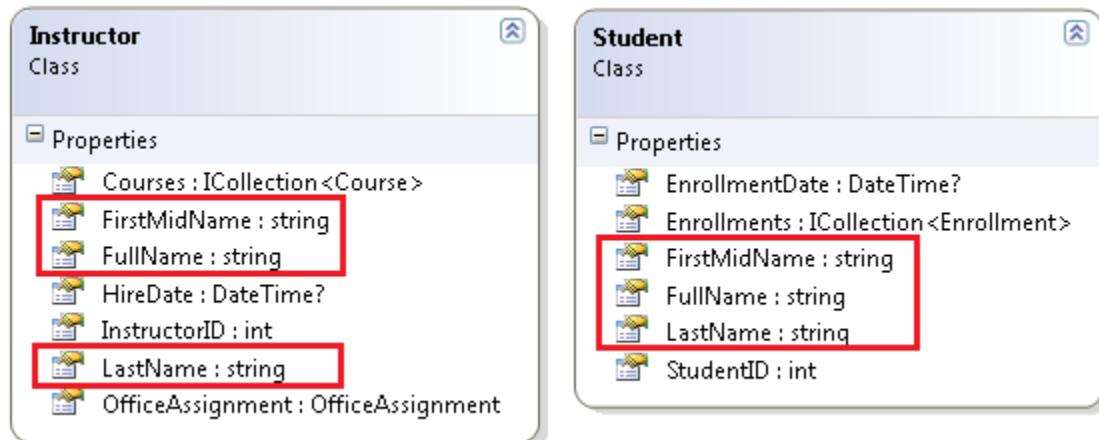
# Implementasi Inheritance dengan Entity Framework pada Aplikasi ASP.NET MVC

Pada tutorial sebelumnya berisi paparan untuk menangani konflik concurrency. Selanjutnya, pada tutorial ini akan ditunjukkan bagaimana implementasi inheritance pada data model.

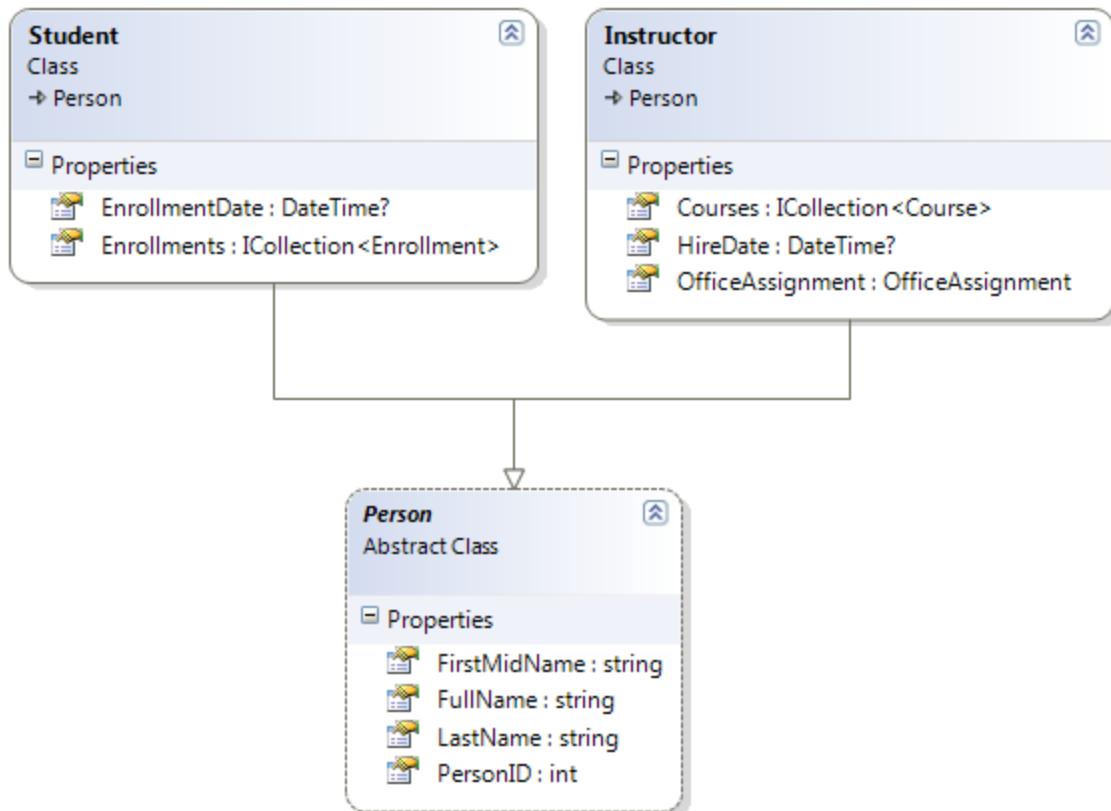
Pada konsep pemrograman berbasis objek, kita dapat menggunakan inheritance untuk mengilangkan kode yang berlebihan. Pada tutorial ini, kita akan mengubah class `Instructor` dan `Student` agar keduanya berasal dari base class `Person` yang telah mempunyai property seperti `LastName` dan property-property lain yang dimiliki oleh instruktur dan pelajar. Kita tidak akan perlu melakukan perubahan pada halaman-halaman web yang sudah ada, tetapi kita akan mengubah beberapa kode dan perubahan tersebut nantinya akan secara otomatis berpengaruh ke database.

## Table-per-Hierarchy vs Table-per-Type Inheritance

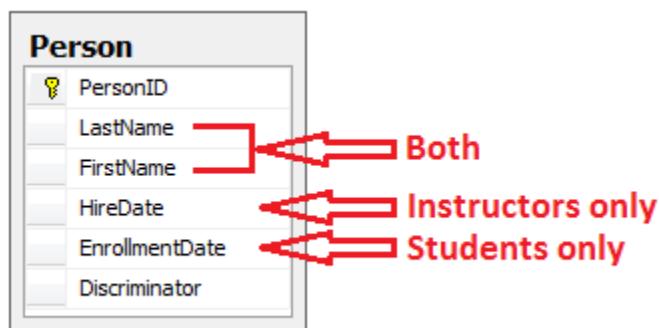
Pada pemrograman berbasis objek, kita dapat menggunakan konsep inheritance untuk membuat pekerjaan lebih mudah. Sebagai contoh, class `Instructor` dan `Student` pada data model `School` memiliki beberapa property yang sama seperti yang dapat dilihat pada gambar berikut :



Hal tersebut tentu saja membuat kode yang ditulis terjadi pengulangan yang mubazir. Untuk menghilangkan hal tersebut kita dapat membuat base class `Person` yang mempunyai property-property yang sama tersebut, kemudian membuat entity `Instructor` dan `Student` menjadi turunan dari base class tersebut.

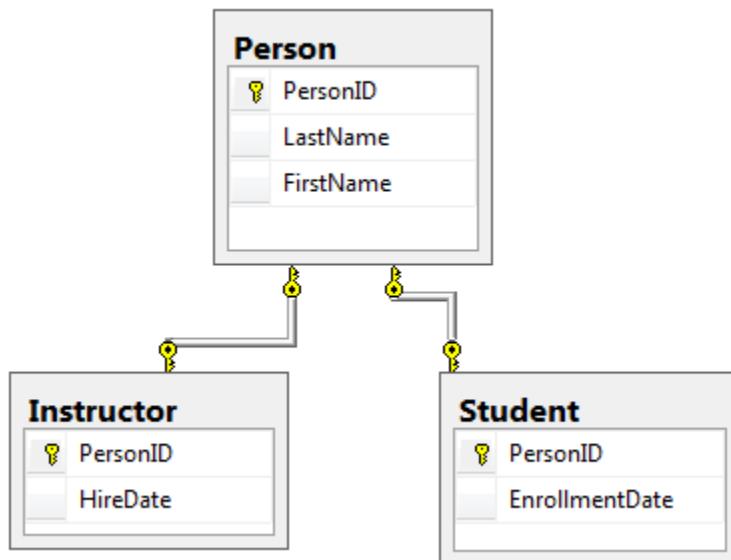


Ada beberapa cara untuk merepresentasikan struktur inheritance ini ke dalam database. Kita dapat membuat tabel **Person** yang memuat informasi dari pelajar dan instruktur ke dalam sebuah tabel. Ada kolom yang hanya digunakan oleh instruktur, ada kolom yang hanya digunakan oleh pelajar, dan ada juga kolom yang akan digunakan oleh keduanya. Selain itu dibutuhkan kolom pembeda untuk menentukan row mana yang berisi data pelajar dan row mana yang berisi data instruktur. Kolom tersebut mungkin akan berisi nilai “Instructor” untuk data instruktur dan “Student” untuk data pelajar.



Pattern seperti ini disebut inheritance table-per-hierarchy (TPH).

Alternatif lain adalah membuat database sesuai dengan struktur inheritance. Sebagai contoh, kita dapat mempunyai tabel Person yang berisi field nama dan tabel terpisah untuk Instructor dan Student dengan field yang sesuai seperti yang dapat dilihat pada gambar berikut ini.



Pattern seperti di atas disebut inheritance tabel-per-type (TPT).

Pattern inheritance TPH biasanya mempunyai performansi yang lebih baik jika diimplementasikan pada Entity Framework jika dibandingkan menggunakan pattern inheritance TPT, karena pattern TPT dapat menghasilkan query join yang komplek. Pada tutorial ini akan didemonstrasikan bagaimana melakukan implementasi inheritance TPH. Untuk itu kita mesti melakukan hal-hal berikut ini :

- Buat class `Person` dan ubah class `Instructor` dan `Student` agar menjadi class turunan dari `Person`.
- Tambahkan kode mapping model-to-database ke class database context.
- Ubah reference `InstructorID` dan `StudentID` menjadi `PersonID` disetiap file yang ada pada project.

## Membuat Class Person

Pada folder `Models`, buat file `Person.cs` dengan isi file seperti kode berikut ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
```

```
{  
  
    public abstract class Person  
  
    {  
  
        [Key]  
  
        public int PersonID { get; set; }  
  
  
        [Required(ErrorMessage = "Last name is required.")]  
        [Display(Name = "Last Name")]  
        [MaxLength(50)]  
  
        public string LastName { get; set; }  
  
  
        [Required(ErrorMessage = "First name is required.")]  
        [Column("FirstName")]  
        [Display(Name = "First Name")]  
        [MaxLength(50)]  
  
        public string FirstMidName { get; set; }  
  
  
        public string FullName  
  
        {  
            get  
            {  
                return LastName + ", " + FirstMidName;  
            }  
        }  
    }  
}
```

Pada file *Instructor.cs*, modifikasi class *Instructor* agar menjadi turunan dari class *Person* dan hilangkan field key dan name seperti kode di bawah ini :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public class Instructor : Person
    {
        [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
        [Required(ErrorMessage = "Hire date is required.")]
        [Display(Name = "Hire Date")]
        public DateTime? HireDate { get; set; }

        public virtual ICollection<Course> Courses { get; set; }

        public virtual OfficeAssignment OfficeAssignment { get; set; }
    }
}
```

Hal yang sama juga dilakukan pada file *Student.cs*, berikut kodennya :

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
```

```

public class Student : Person
{
    [Required(ErrorMessage = "Enrollment date is required.")]
    [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
    [Display(Name = "Enrollment Date")]
    public DateTime? EnrollmentDate { get; set; }

    public virtual ICollection<Enrollment> Enrollments { get; set; }
}

```

## Menambah Tipe Entity Person pada Model

Pada `SchoolContext.cs`, tambahkan property `DbSet` untuk tipe entity `Person` :

```

public DbSet<Person> People { get; set; }

```

Hal tersebut dibutuhkan oleh Entity Framework untuk mengkonfigurasi inheritance table-per-hierarchy. Kita akan melihat saat database dibuat ulang kita akan memiliki tabel `Person`.

## Changing InstructorID and StudentID menjadi PersonID

Pada `SchoolContext.cs`, pada statement Instructor-Course, ganti `MapRightKey("InstructorID")` menjadi `MapRightKey("PersonID")` :

```

modelBuilder.Entity<Course>()
    .HasMany(c => c.Instructors).WithMany(i => i.Courses)
    .Map(t => t.MapLeftKey("CourseID"))
    .MapRightKey("PersonID")
    .ToTable("CourseInstructor"));

```

Perubahan ini sebenarnya tidak harus dilakukan, tanpa ada perubahan tersebut aplikasi tetap akan berjalan dengan baik. Perubahan tersebut hanya berfungsi untuk mengganti nama kolom InstructorID pada tabel join many-to-many.

Selanjutnya, melakukan perubahan global (pada semua file pada project) yaitu mengubah `InstructorID` menjadi `PersonID` dan `StudentID` menjadi `PersonID`. Perhatikan case sensitive dari hal-hal yang kita ubah.

## Menyesuaikan Nilai Primary Key pada Initializer

Pada `SchoolInitializer.cs`, pada kode ini nilai primary key pada entity `Student` dan `Instructor` diberikan nilai dengan angka secara terpisah. Untuk nilai pada entity `Student` masih mempunyai nilai yang benar yaitu 1 sampai 8, sedangkan pada entity `Instructor` nilainya akan dimulai dengan 9 sampai 13, bukan bernilai 1 sampai 5. Hal ini jelas terjadi karena kode untuk menambahkan data instruktur dijalankan setelah kode untuk menambah data student. Ganti kode pada entity `Department` dan `OfficeAssignment` dengan kode yang menggunakan nilai ID baru untuk instruktur.

```
var departments = new List<Department>
{
    new Department { Name = "English",      Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 9 },
    new Department { Name = "Mathematics",   Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 10 },
    new Department { Name = "Engineering",   Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 11 },
    new Department { Name = "Economics",     Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 12 }
};
```

```
var officeAssignments = new List<OfficeAssignment>
{
    new OfficeAssignment { PersonID = 9, Location = "Smith 17" },
    new OfficeAssignment { PersonID = 10, Location = "Gowan 27" },
    new OfficeAssignment { PersonID = 11, Location = "Thompson 304" },
};
```

## Mengubah OfficeAssignment menjadi Lazy Loading

Versi saat ini dari Entity Framework belum mendukung eager loading untuk relasi one-to-zero-or-one ketika property navigation pada class turunan dari struktur inheritance TPH. Seperti yang terjadi pada property `OfficeAssignment` pada entity `Instructor`. Untuk mengatasi hal ini, kita akan menghapus kode yang sebelumnya ditambahkan agar dapat dilakukan eager loading pada property ini :

Pada file `InstructorController.cs`, hapus tiga baris dari baris kode berikut ini :

```
.Include(i => i.OfficeAssignment)
```

## Testing

Jalankan site dan coba untuk mengakses halaman-halaman yang ada di site untuk memastikan semua berjalan seperti sebelumnya.

Pada **Solution Explorer**, klik dua kali pada **School.sdf** untuk membuka database pada **Server Explorer**. Buka **School.sdf**, kemudian pada bagian Tables kita akan dapat melihat tabel **Student** dan **Instructor** sudah diganti dengan tabel **Person**. Buka tabel **Person** dan kita akan dapat melihat kolom-kolom seperti yang dilihat pada gambar berikut.

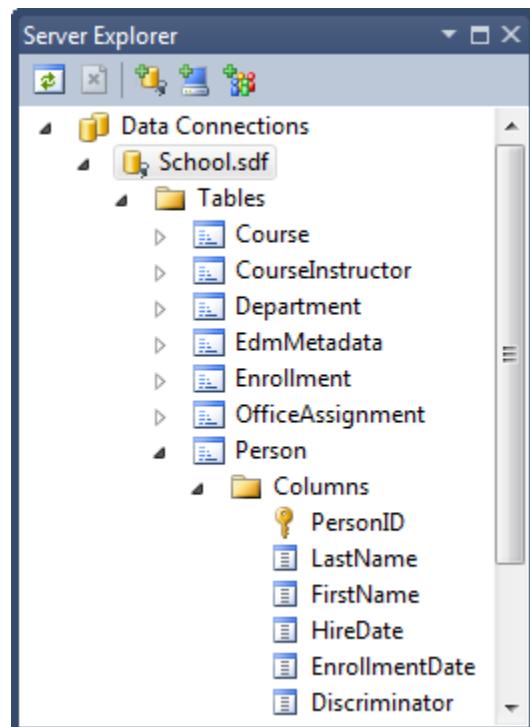
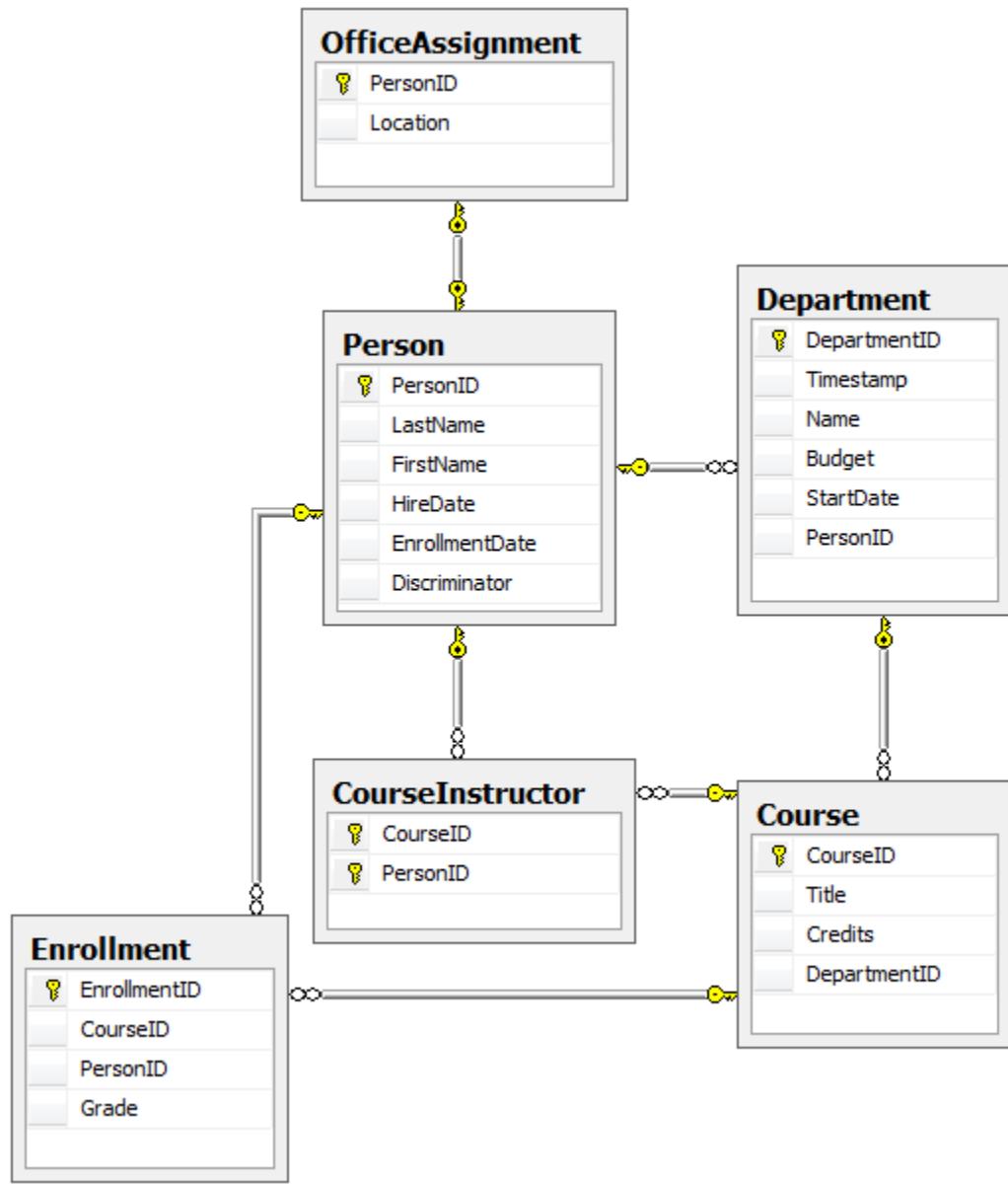


Diagram di bawah ini memberikan gambaran struktur tabel pada database School yang baru.



Inheritance table-per-hierarchy telah diimplementasikan pada class Person, Student dan Instructor. Untuk mengetahui lebih lanjut tentang struktur inheritance yang lain bisa mengunjungi blog milik Morteza Manavi dengan judul Inheritance Mapping Strategies (<http://weblogs.asp.net/manavi/archive/2010/12/24/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-1-table-per-hierarchy-tph.aspx>). Pada tutorial selanjutnya beberapa cara untuk implementasi pattern repository dan unit of work.

# Implementasi Pattern Repository dan Unit of Work pada Aplikasi ASP.NET MVC

---

Pada tutorial sebelumnya, kita sudah menggunakan inheritance untuk mengurangi kode sama sama dan membuat mubazir yang terdapat pada class entity `Student` dan `Instructor`. Pada tutorial ini kita akan menggunakan pattern repository dan unit of work untuk operasi CRUD. Seperti pada tutorial sebelumnya, pada tutorial ini kita akan mengubah kode yang sudah ada pada file-file pada project, dan bukan membuat kode dengan file baru.

## Pattern Repository dan Unit of Work

Repository dan Unit of Work berfungsi untuk membuat abstraction layer antara layer data access dan layer logic dari aplikasi. Penerapan pattern ini dapat membantu untuk melindungi aplikasi dari perubahan yang mungkin terjadi dalam penyimpanan dan dapat memfasilitasi untuk penerapan automated unit testing atau test-driven-development (TDD).

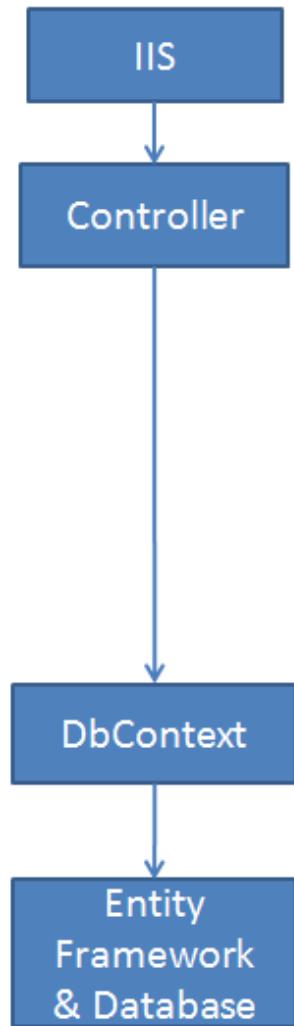
Pada tutorial ini kita akan melakukan implementasi class repository untuk setiap type entity. Untuk entity `Student` kita akan membuat interface repository dan class repository. Jika kita ingin menggunakan repository pada controller, kita harus menggunakan interface agar controller dapat menerima referensi ke objek yang mengimplementasi interface repository. Ketika controller berjalan dibawah class unit test, maka controller tersebut akan menerima repository yang bekerja dengan data yang disimpan dengan cara yang dapat kita manipulasi dengan mudah saat pengujian, seperti data yang ada di memory.

Pada tutorial ini kita akan menggunakan class-class repository dan unit of work untuk tipe entity `Course` dan `Department` pada controller `Course`. Class Unit of Work mengkoordinasikan pekerjaan beberapa repository dengan membuat sebuah class database context. Jika kita ingin dapat melakukan automated unit test, kita harus membuat dan menggunakan interface untuk class-class tersebut seperti yang kita lakukan pada repository `Student`. Tetapi untuk membuat tutorial ini sederhana, kita akan membuat class-class tanpa interface.

Gambar di bawah ini memperlihatkan perbandingan hubungan antara controller dan class context yang mengimplementasikan penggunaan repository dan yang tidak menggunakan repository.

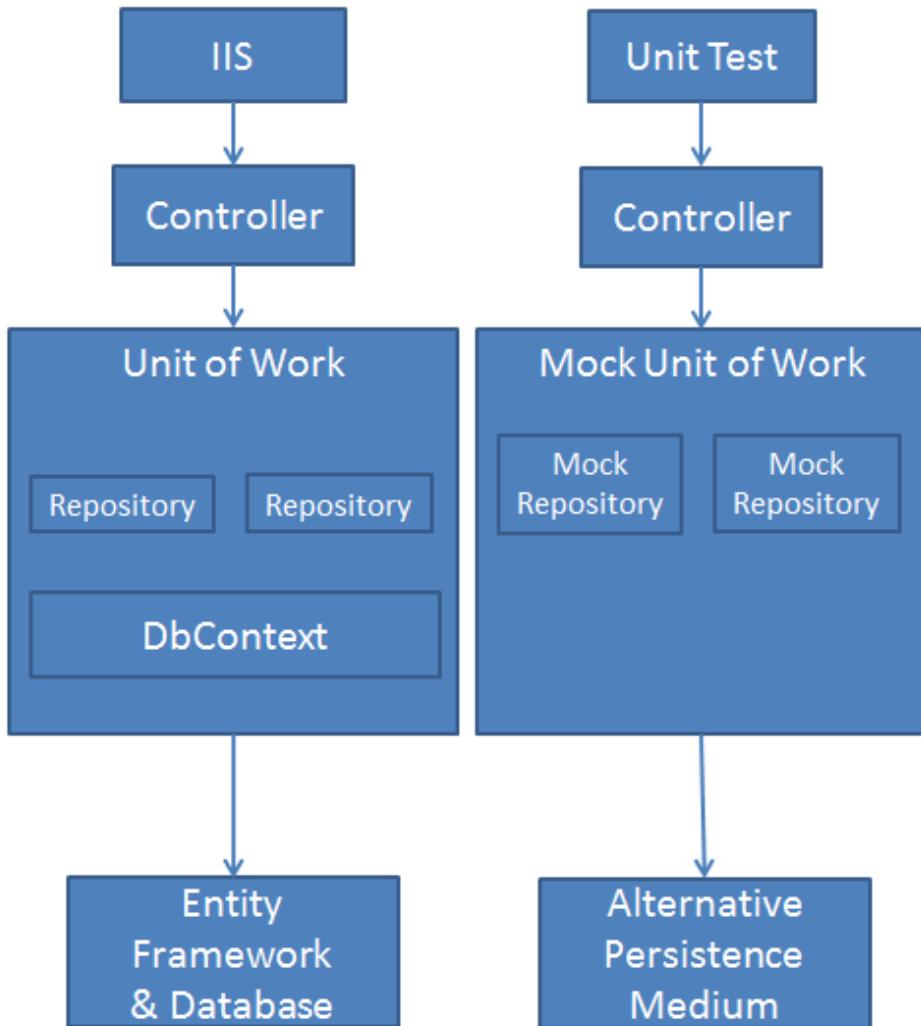
## No Repository

Direct access to database context from controller.



## With Repository

Abstraction layer between controller and database context. Unit tests can use a custom persistence layer to facilitate testing.



Pada tutorial ini tidak ada pembuatan unit test. Untuk pendahuluan TDD pada aplikasi MVC yang menggunakan pattern repository dapat melihat posting pada MSDN library ini Walkthrough: Using TDD with ASP.NET MVC (<http://msdn.microsoft.com/en-us/library/ff847525.aspx>). Untuk informasi lebih lanjut tentang pattern repository dapat membaca posting berikut Using Repository and Unit of Work patterns with Entity Framework 4.0 (<http://blogs.msdn.com/b/adonet/archive/2009/06/16/using-repository-and-unit-of-work-patterns-with-entity-framework-4.0.aspx>) dan Agile Entity Framework 4 Repository (<http://thedatafarm.com/blog/data-access/agile-entity-framework-4-repository-part-1-model-and-poco-classes/>).

**Note :**

Ada banyak cara implementasi pattern Repository dan Unit of Work. Kita dapat menggunakan class-

class Repository dengan atau tanpa class Unit of Work. Kita dapat mengimplementasi satu repository saja untuk digunakan oleh seluruh tipe entity atau satu repository untuk satu tipe entity. Jika kita menggunakan satu repository untuk setiap tipe entity, kita dapat menggunakan class terpisah, class generic utama (generic base class) dan class turunan. Kita dapat memasukkan business logic ke dalam repository atau membatasinya ke data access logic. Kita juga dapat membuat layer abstraksi pada dalam class database context dengan menggunakan interface `IDbSet` selain tipe `DbSet` untuk entity yang kita miliki. Pendekatan dengan implementasi layer abstraksi yang dijelaskan pada tutorial ini adalah salah satu pilihan yang dapat kita pertimbangkan, tapi tidak direkomendasikan untuk digunakan pada seluruh skenario dan lingkungan sistem.

## Membuat Class Student Repostory

Pada folder `DAL`, buat file `IStudentRepository.cs` dan isi file tersebut dengan kode di bawah ini :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public interface IStudentRepository : IDisposable
    {
        IEnumerable<Student> GetStudents();
        Student GetStudentByID(int studentId);
        void InsertStudent(Student student);
        void DeleteStudent(int studentID);
        void UpdateStudent(Student student);
        void Save();
    }
}
```

Kode tersebut mendeklarasikan method-method yang biasa ada pada operasi CRUD.

Pada folder *DAL*, buat class dengan *nama StudentRepository.cs*. Dan gunakan kode di bawah ini sebagai isinya. Class ini menggunakan interface *IStudentRepository*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public class StudentRepository : IStudentRepository, IDisposable
    {
        private SchoolContext context;

        public StudentRepository(SchoolContext context)
        {
            this.context = context;
        }

        public IEnumerable<Student> GetStudents()
        {
            return context.Students.ToList();
        }

        public Student GetStudentByID(int id)
        {
```

```
        return context.Students.Find(id);

    }

    public void InsertStudent(Student student)
    {
        context.Students.Add(student);
    }

    public void DeleteStudent(int studentID)
    {
        Student student = context.Students.Find(studentID);
        context.Students.Remove(student);
    }

    public void UpdateStudent(Student student)
    {
        context.Entry(student).State = EntityState.Modified;
    }

    public void Save()
    {
        context.SaveChanges();
    }

    private bool disposed = false;

    protected virtual void Dispose(bool disposing)
    {
```

```

        if (!this.disposed)
    {
        if (disposing)
        {
            context.Dispose();
        }
    }

    this.disposed = true;
}

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
}

```

Pada kode di atas, dapat dilihat terdapat variable pada class yang didefinisikan dari database context. Kemudian pada contructor class tersebut memiliki parameter yang dapat diisi dengan objek context :

```

private SchoolContext context;

public StudentRepository(SchoolContext context)
{
    this.context = context;
}

```

Kita bisa menginstansiasi context baru dalam repository, tapi jika kita nanti menggunakan beberapa repository dalam satu controller, maka nantinya akan memiliki context masing-masing. Nanti kita akan

menggunakan beberapa repository pada controller `Course`, dan kita akan dapat melihat bagaimana class Unit of Work dapat memastikan bahwa semua repository menggunakan context yang sama.

Pada repository dapat dilihat terdapat penggunaan interface `IDisposable`, sehingga dapat melakukan proses penghilangan database context seperti yang dapat dilihat pada kode di atas. Selain itu kita juga dapat melihat bagaimana method-method operasi CRUD dipanggil dengan cara yang sama seperti sebelumnya.

## Mengubah Student Controller untuk Menggunakan Repository

Pada file `StudentController.cs`, ganti kode yang sudah ada dengan kode berikut :

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ContosoUniversity.Models;
using ContosoUniversity.DAL;
using PagedList;

namespace ContosoUniversity.Controllers
{
    public class StudentController : Controller
    {
        private IStudentRepository studentRepository;

        public StudentController()
        {
```

```
        this.studentRepository = new StudentRepository(new SchoolContext());  
    }  
  
    public StudentController(IStudentRepository studentRepository)  
    {  
        this.studentRepository = studentRepository;  
    }  
  
    //  
    // GET: /Student/  
  
    public ViewResult Index(string sortOrder, string currentFilter, string  
searchString, int? page)  
    {  
        ViewBag.CurrentSort = sortOrder;  
        ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "Name desc" : "";  
        ViewBag.DateSortParm = sortOrder == "Date" ? "Date desc" : "Date";  
  
        if (Request.HttpMethod == "GET")  
        {  
            searchString = currentFilter;  
        }  
        else  
        {  
            page = 1;  
        }  
        ViewBag.CurrentFilter = searchString;
```

```
var students = from s in studentRepository.GetStudents()

    select s;

if (!String.IsNullOrEmpty(searchString))
{
    students = students.Where(s =>
s.LastName.ToUpper().Contains(searchString.ToUpper())
||

s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));
}

switch (sortOrder)
{
    case "Name desc":
        students = students.OrderByDescending(s => s.LastName);
        break;
    case "Date":
        students = students.OrderBy(s => s.EnrollmentDate);
        break;
    case "Date desc":
        students = students.OrderByDescending(s => s.EnrollmentDate);
        break;
    default:
        students = students.OrderBy(s => s.LastName);
        break;
}

int pageSize = 3;
int pageNumber = (page ?? 1);

return View(students.ToPagedList(pageNumber, pageSize));
```

```
}

//  
// GET: /Student/Details/5  
  
  
public ViewResult Details(int id)  
{  
    Student student = studentRepository.GetStudentByID(id);  
    return View(student);  
}  
  
  
//  
// GET: /Student/Create  
  
  
public ActionResult Create()  
{  
    return View();  
}  
  
  
//  
// POST: /Student/Create  
  
  
[HttpPost]  
public ActionResult Create(Student student)  
{  
    try  
    {
```

```
        if (ModelState.IsValid)
    {
        studentRepository.InsertStudent(student);
        studentRepository.Save();
        return RedirectToAction("Index");
    }
}

catch (DataException)
{
    //Log the error (add a variable name after DataException)

    ModelState.AddModelError("", "Unable to save changes. Try again, and if
the problem persists see your system administrator.");
}

return View(student);
}

// 
// GET: /Student/Edit/5

public ActionResult Edit(int id)
{
    Student student = studentRepository.GetStudentByID(id);
    return View(student);
}

// 
// POST: /Student/Edit/5
```

```
[HttpPost]

public ActionResult Edit(Student student)

{
    try
    {
        if (ModelState.IsValid)
        {
            studentRepository.UpdateStudent(student);
            studentRepository.Save();
            return RedirectToAction("Index");
        }
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        ModelState.AddModelError("", "Unable to save changes. Try again, and if
the problem persists see your system administrator.");
    }
    return View(student);
}

// 
// GET: /Student/Delete/5


public ActionResult Delete(int id, bool? saveChangesError)
{
    if (saveChangesError.GetValueOrDefault())
    {
        ViewBag.ErrorMessage = "Unable to save changes. Try again, and if the
```

```
problem persists see your system administrator.";

}

Student student = studentRepository.GetStudentByID(id);

return View(student);

}

// POST: /Student/Delete/5

[HttpPost, ActionName("Delete")]

public ActionResult DeleteConfirmed(int id)

{

    try

    {

        Student student = studentRepository.GetStudentByID(id);

        studentRepository.DeleteStudent(id);

        studentRepository.Save();

    }

    catch (DataException)

    {

        //Log the error (add a variable name after DataException)

        return RedirectToAction("Delete",

            new System.Web.Routing.RouteValueDictionary {

                { "id", id },

                { "saveChangesError", true } });

    }

    return RedirectToAction("Index");
}
```

```
}

protected override void Dispose(bool disposing)
{
    studentRepository.Dispose();
    base.Dispose(disposing);
}

}

}
```

Pada controller terdapat deklarasi variabel yang mengimplementasikan interface IStudentRepository, bukan dari class context.

```
private IStudentRepository studentRepository;
```

Pada class ini terdapat dua constructor yang dapat digunakan :

```
public StudentController()
{
    this.studentRepository = new StudentRepository(new SchoolContext());
}

public StudentController(IStudentRepository studentRepository)
{
    this.studentRepository = studentRepository;
}
```

Sedangkan untuk pemanggilan method operasi CRUD dapat dilihat seperti di bawah ini :

```
var students = from s in studentRepository.GetStudents()
               select s;
```

```
Student student = studentRepository.GetStudentByID(id);
```

```
studentRepository.InsertStudent(student);
studentRepository.Save();
```

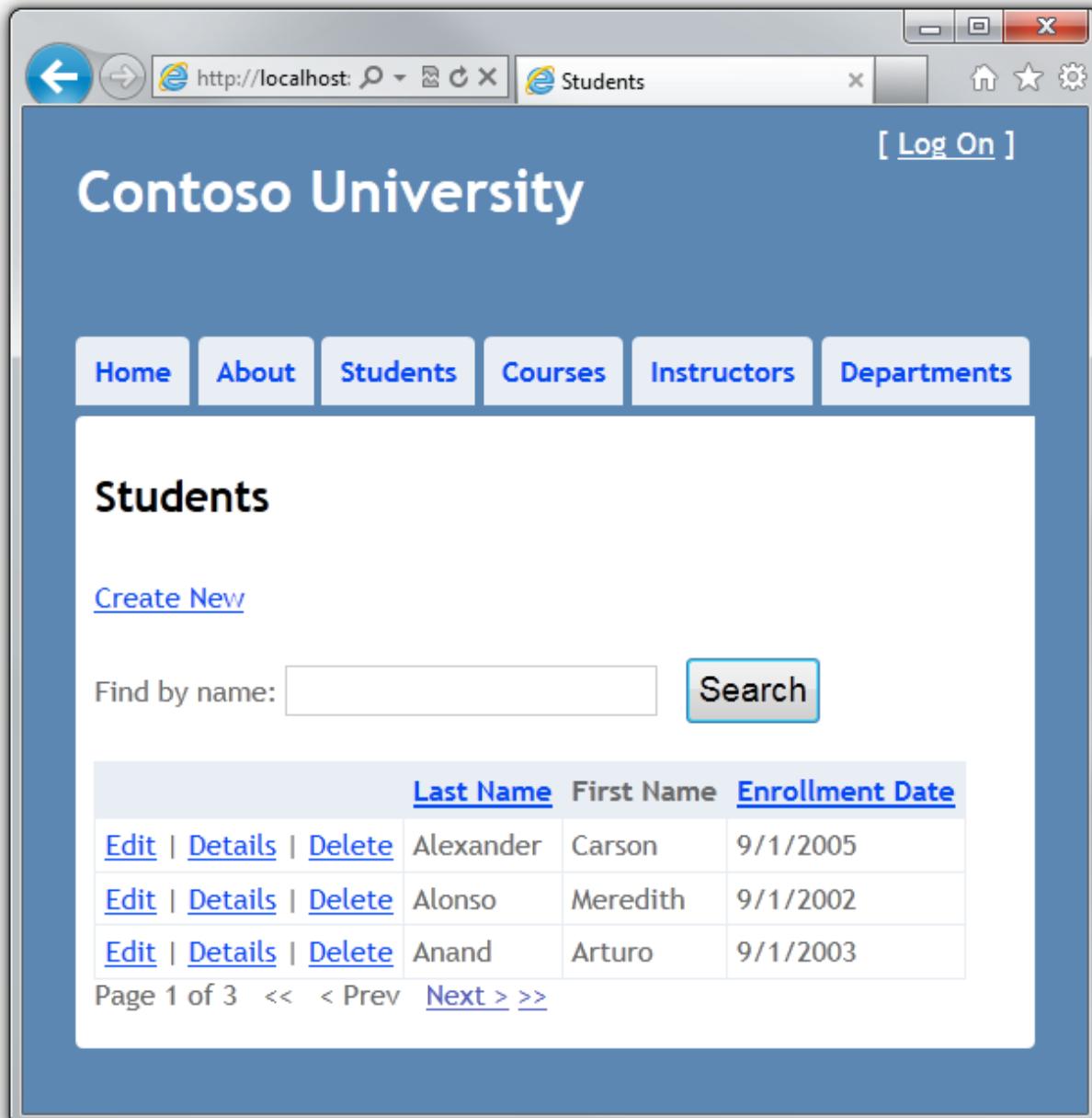
```
studentRepository.UpdateStudent(student);
studentRepository.Save();
```

```
studentRepository.DeleteStudent(id);
studentRepository.Save();
```

Dan method Dispose sekarang berfungsi untuk menghilangkan repository, bukan context :

```
studentRepository.Dispose();
```

Selanjutkan kita coba untuk menjalankan site kemudian pilih tab **Student**.



Dapat dilihat sepertinya halaman masih berjalan dengan baik seperti sebelum kode kita lakukan modifikasi untuk menggunakan repository, begitu juga halaman-halaman student yang lainnya.

Walaupun ada perbedaan yang penting pada method Index dari controller dalam melakukan filter dan pengurutan. Pada kode sebelumnya method ini mempunyai kode seperti berikut :

```
var students = from s in context.Students  
                select s;  
  
if (!String.IsNullOrEmpty(searchString))  
{  
    students = students.Where(s =>  
        s.LastName.ToUpper().Contains(searchString.ToUpper())  
        ||  
        s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));  
}  
}
```

Pada kode sebelumnya, digunakan object `IQueryable`. Query tidak akan dikirimkan ke database sebelum ada proses convert ke collection, sebagai contoh setelah pemanggilan method `ToList()`. Pada kode yang baru, variabel student adalah collection `IEnumerable`. walaupun hasilnya sama dengan kode sebelumnya tetapi cara kerjanya berbeda, proses akan dilakukan di memory web server bukan di database. Untuk data yang banyak hal ini tentu saja tidak effisien. Pada bagian selanjutnya akan diperlihatkan bagaimana implementasi method repository yang memungkinkan melakukan proses oleh database.

Kita telah membuat layer abstraksi antara controller dan database context Entity Framework. Jika kita akan melakukan automated unit testing pada aplikasi, maka kita dapat membuat class repository alternatif dalam project unit test yang mengimplementasikan `IStudentRepository`. Class repository dapat memanipulasi collection pada memory untuk menguji fungsi controller, sebagai pengganti pemanggilan context untuk membaca dan menulis data.

## Implementasi Class Generic Repository dan Unit of Work

Membuat class repository untuk setiap tipe entity dapat menyebabkan banyak penulisan kode yang sama dan mubazir, dan hal itu bisa mengakibatkan update parsial. Sebagai contoh, kita harus mengupdate dua tipe entity yang berbeda sebagai bagian dari sebuah transaksi. Jika masing-masing menggunakan instance database context yang terpisah maka ada kemungkinan salah satu dapat diupdate dan yang lain mungkin mengalami kegagalan. Salah satu cara mengatasi kode yang sama dan mubazir adalah dengan menggunakan repository generic, dan salah satu cara untuk memastikan bahwa semua repository menggunakan database context yang sama adalah dengan menggunakan class Unit of Work.

Pada tutorial bagian ini, kita akan membuat class `GenericRepository` dan class `UnitOfWork` dan nantinya akan digunakan dalam controller `Course` untuk mengakses entity `Department` dan `Course`.

## Membuat Generic Repository

Pada folder `DAL`, buat `GenericRepository.cs` dan gunakan kode di bawah ini :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using System.Data.Entity;
using ContosoUniversity.Models;
using System.Linq.Expressions;

namespace ContosoUniversity.DAL
{
    public class GenericRepository<TEntity> where TEntity : class
    {
        internal SchoolContext context;
        internal DbSet<TEntity> dbSet;

        public GenericRepository(SchoolContext context)
        {
            this.context = context;
            this.dbSet = context.Set<TEntity>();
        }

        public virtual IEnumerable<TEntity> Get(
            Expression<Func<TEntity, bool>> filter = null,
```

```
Func<IQueryable< TEntity >, IOrderedQueryable< TEntity >> orderBy = null,  
        string includeProperties = "")  
  
{  
    IQueryable< TEntity > query = dbSet;  
  
    if (filter != null)  
    {  
        query = query.Where(filter);  
    }  
  
    foreach (var includeProperty in includeProperties.Split  
        (new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))  
    {  
        query = query.Include(includeProperty);  
    }  
  
    if (orderBy != null)  
    {  
        return orderBy(query).ToList();  
    }  
    else  
    {  
        return query.ToList();  
    }  
}  
  
public virtual TEntity GetByID(object id)  
{
```

```
        return dbSet.Find(id);

    }

    public virtual void Insert(TEntity entity)
    {
        dbSet.Add(entity);
    }

    public virtual void Delete(object id)
    {
        TEntity entityToDelete = dbSet.Find(id);
        Delete(entityToDelete);
    }

    public virtual void Delete(TEntity entityToDelete)
    {
        if (context.Entry(entityToDelete).State == EntityState.Detached)
        {
            dbSet.Attach(entityToDelete);
        }
        dbSet.Remove(entityToDelete);
    }

    public virtual void Update(TEntity entityToUpdate)
    {
        dbSet.Attach(entityToUpdate);
        context.Entry(entityToUpdate).State = EntityState.Modified;
    }
}
```

```
    }  
}
```

Dapat dilihat terdapat deklarasi dua variable seperti di bawah ini :

```
internal SchoolContext context;  
internal DbSet< TEntity > dbSet;
```

Pada constructor menerima instance database context sebagai parameter :

```
public GenericRepository(SchoolContext context)  
{  
    this.context = context;  
    this.dbSet = context.Set< TEntity >();  
}
```

Pada method Get digunakan lambda expression agar dimungkinkan melakukan pemanggilan kode berdasarkan kondisi filter dan kolom yang diinginkan, selain itu juga terdapat parameter string yang dapat diisikan dengan daftar property navigation yang dipisahkan oleh koma untuk eager loading.

```
public virtual IEnumerable< TEntity > Get(  
    Expression< Func< TEntity, bool > > filter = null,  
    Func< IQueryable< TEntity >, IOrderedQueryable< TEntity > > orderBy = null,  
    string includeProperties = "")
```

Kode `Expression< Func< TEntity, bool > > filter` artinya dapat diberikan lambda expression berdasarkan tipe `TEntity type`, expression ini akan mengembalikan nilai Boolean.

Sebagai contoh, untuk tipe entity `Student` maka parameter filernya adalah `student => student.LastName == "Smith".`

Sedangkan bagian `Func<IQueryable< TEntity >, IOrderedQueryable< TEntity >>` `orderBy` mempunyai arti dapat diberikan lambda expression sebagai input. Pada kasus ini, inputnya adalah objek `IQueryable` untuk tipe `TEntity`. Expression akan mengembalikan versi object `IQueryable` yang telah diurut. Sebagai contoh, untuk tipe entity `Student`, maka parameter inputnya `orderBy` adalah `q => q.OrderBy(s => s.LastName).`

Kode pada method Get membuat objet `IQueryable` dan proses filter akan dilakukan jika nilai `filter` bukan `null`.

```
if (filter != null)
{
    query = query.Where(filter);
}
```

Selanjutnya diterapkan expression eager loading setelah melakukan parsing daftar nilai yang dipisahkan oleh tanda koma.

```
foreach (var includeProperty in includeProperties.Split
    (new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))
{
    query = query.Include(includeProperty);
}
```

Yang terakhir, dilakukan proses pengurutan dengan kode berikut ini :

```
if (orderBy != null)
{
    return orderBy(query).ToList();
}
```

```
else
{
    return query.ToList();
}
```

Ketika kita memanggil method Get, kita dapat melakukan proses filter dan pengurutan dengan memanfaatkan collection `IEnumerable`, tetapi tentunya proses tersebut akan dilakukan pada memory di web server. Dengan kode di atas, dipastikan kita melakukan proses di database. Sebagai alternatif, kita dapat membuat method yang mempunyai fungsi khusus seperti `GetStudentsInNameOrder` atau `GetStudentsByName`. Walaupun pada aplikasi yang rumit, hal ini membuat lebih banyak hal yang harus dikelola.

Kode pada method GetById, Insert, dan Update berisi sama dengan yang kita lihat di dalam non-generic repository.

Terdapat dua method Delete yang bisa digunakan :

```
public virtual void Delete(object id)
{
    TEntity entityToDelete = dbSet.Find(id);
    Delete(entityToDelete);
}

public virtual void Delete(TEntity entityToDelete)
{
    if (context.Entry(entityToDelete).State == EntityState.Detached)
    {
        dbSet.Attach(entityToDelete);
    }
    dbSet.Remove(entityToDelete);
}
```

Generic repository ini menangani kebutuhan operasi CRUD standar. Jika pada perkembangannya dibutuhkan proses filter dan order yang lebih rumit maka kita dapat menurunkan class ini untuk menambah method yang diinginkan.

## Membuat Class Unit of Work

Class unit of work ini berfungsi untuk memastikan ketika kita menggunakan beberapa repository, maka repository-repository tersebut menggunakan satu database context. Setelah unit of work ini selesai, maka kita dapat memanggil method SaveChanges dan semua perubahan pada entity yang terkait akan dilakukan.

Pada folder DAL, buat class dengan nama UnitOfWork.cs dan gunakan kode di bawah ini :

```
using System;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public class UnitOfWork : IDisposable
    {
        private SchoolContext context = new SchoolContext();
        private GenericRepository<Department> departmentRepository;
        private GenericRepository<Course> courseRepository;

        public GenericRepository<Department> DepartmentRepository
        {
            get
            {
                if (this.departmentRepository == null)
                {
                    this.departmentRepository = new
                    GenericRepository<Department>(context);
                }
            }
        }
    }
}
```

```
        return departmentRepository;
    }

}

public GenericRepository<Course> CourseRepository
{
    get
    {

        if (this.courseRepository == null)
        {
            this.courseRepository = new GenericRepository<Course>(context);
        }
        return courseRepository;
    }
}

public void Save()
{
    context.SaveChanges();
}

private bool disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!this.disposed)
    {
```

```
        if (disposing)
    {
        context.Dispose();
    }
}

this.disposed = true;
}

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
}
```

Berikut adalah variabel-variabel untuk database context dan repository.

```
private SchoolContext context = new SchoolContext();

private GenericRepository<Department> departmentRepository;

private GenericRepository<Course> courseRepository;
```

Pemeriksaan keberadaan repository dan memastikan seluruh repository menggunakan instance context yang sama.

```
public GenericRepository<Department> DepartmentRepository
{
    get
    {
```

```
    if (this.departmentRepository == null)
    {
        this.departmentRepository = new GenericRepository<Department>(context);
    }
    return departmentRepository;
}
}
```

Method `Save` memanggil `SaveChanges` dari database context.

```
public void Save()
{
    context.SaveChanges();
}
```

Class `UnitOfWork` mengimplement `IDisposable` untuk menghapus context.

## Mengubah Course Controller untuk Menggunakan Class UnitOfWork dan Repositories

Pada file `CourseController.cs`, ganti kode yang sudah ada dengan kode berikut ini :

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ContosoUniversity.Models;
using ContosoUniversity.DAL;
```

```
namespace ContosoUniversity.Controllers

{
    public class CourseController : Controller
    {
        private UnitOfWork unitOfWork = new UnitOfWork();

        //
        // GET: /Course/

        public ViewResult Index()
        {
            var courses = unitOfWork.CourseRepository.Get(includeProperties:
"Department");
            return View(courses.ToList());
        }

        //
        // GET: /Course/Details/5

        public ViewResult Details(int id)
        {
            Course course = unitOfWork.CourseRepository.GetByID(id);
            return View(course);
        }

        //
        // GET: /Course/Create
```

```
public ActionResult Create()
{
    PopulateDepartmentsDropDownList();
    return View();
}

[HttpPost]
public ActionResult Create(Course course)
{
    try
    {
        if (ModelState.IsValid)
        {
            unitOfWork.CourseRepository.Insert(course);
            unitOfWork.Save();
            return RedirectToAction("Index");
        }
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        ModelState.AddModelError("", "Unable to save changes. Try again, and if
the problem persists, see your system administrator.");
    }
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}
```

```
public ActionResult Edit(int id)
{
    Course course = unitOfWork.CourseRepository.GetByID(id);
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}

[HttpPost]
public ActionResult Edit(Course course)
{
    try
    {
        if (ModelState.IsValid)
        {
            unitOfWork.CourseRepository.Update(course);
            unitOfWork.Save();
            return RedirectToAction("Index");
        }
    }
    catch (DataException)
    {
        //Log the error (add a variable name after DataException)

        ModelState.AddModelError("", "Unable to save changes. Try again, and if
the problem persists, see your system administrator.");
    }
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}
```

```
private void PopulateDepartmentsDropDownList(object selectedDepartment = null)
{
    var departmentsQuery = unitOfWork.DepartmentRepository.Get(
        orderBy: q => q.OrderBy(d => d.Name));

    ViewBag.DepartmentID = new SelectList(departmentsQuery, "DepartmentID",
    "Name", selectedDepartment);
}

// GET: /Course/Delete/5

public ActionResult Delete(int id)
{
    Course course = unitOfWork.CourseRepository.GetByID(id);

    return View(course);
}

// POST: /Course/Delete/5

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
{
    Course course = unitOfWork.CourseRepository.GetByID(id);

    unitOfWork.CourseRepository.Delete(id);

    unitOfWork.Save();

    return RedirectToAction("Index");
}
```

```
}

protected override void Dispose(bool disposing)
{
    unitOfWork.Dispose();
    base.Dispose(disposing);
}

}

}
```

Pada kode di atas dapat dilihat penambahan variabel untuk class **UnitOfWork**.

```
private UnitOfWork unitOfWork = new UnitOfWork();
```

Referensi ke database context diganti dengan referensi ke repository yang sesuai menggunakan property-property **UnitOfWork**.

```
var courses = unitOfWork.CourseRepository.Get(includeProperties: "Department");

// ...

Course course = unitOfWork.CourseRepository.GetByID(id);

// ...

unitOfWork.CourseRepository.Insert(course);

unitOfWork.Save();

// ...

Course course = unitOfWork.CourseRepository.GetByID(id);

// ...

unitOfWork.CourseRepository.Update(course);

unitOfWork.Save();

// ...
```

```
var departmentsQuery = unitOfWork.DepartmentRepository.Get(
    orderBy: q => q.OrderBy(d => d.Name));

// ...

Course course = unitOfWork.CourseRepository.GetByID(id);

// ...

unitOfWork.CourseRepository.Delete(id);

unitOfWork.Save();

// ...

unitOfWork.Dispose();
```

Jalankan site dan klik tab **Course**.

The screenshot shows a web browser window with the URL <http://localhost> in the address bar. The title of the page is "Courses". The header features the "Contoso University" logo and a "Log On" link. Below the header is a navigation menu with links for Home, About, Students, Courses, Instructors, and Departments. The main content area is titled "Courses" and contains a "Create New" link. Below this is a table listing course details:

	Number	Title	Credits	Department
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2021	Composition	3	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2042	Literature	4	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1045	Calculus	4	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	3141	Trigonometry	4	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1050	Chemistry	3	Engineering
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4022	Microeconomics	3	Economics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4041	Macroeconomics	3	Economics

Lakukan test akses untuk halaman-halaman yang ada, untuk memastikan seluruh halaman mempunyai tampilan yang sama dan bekerja tanpa error seperti sebelum kita melakukan perubahan.

Sekarang kita telah mengimplementasikan pattern repository dan unit of work. Kita telah menggunakan lambda expression sebagai parameter pada generic repository. Untuk informasi lebih banyak mengenai penggunaan expression dengan objek IQueryable, dapat mengunjungi MSDN Library berikut ini [IQueryable\(T\) Interface \(System.Linq\)](http://msdn.microsoft.com/en-us/library/bb351562.aspx) (<http://msdn.microsoft.com/en-us/library/bb351562.aspx>).

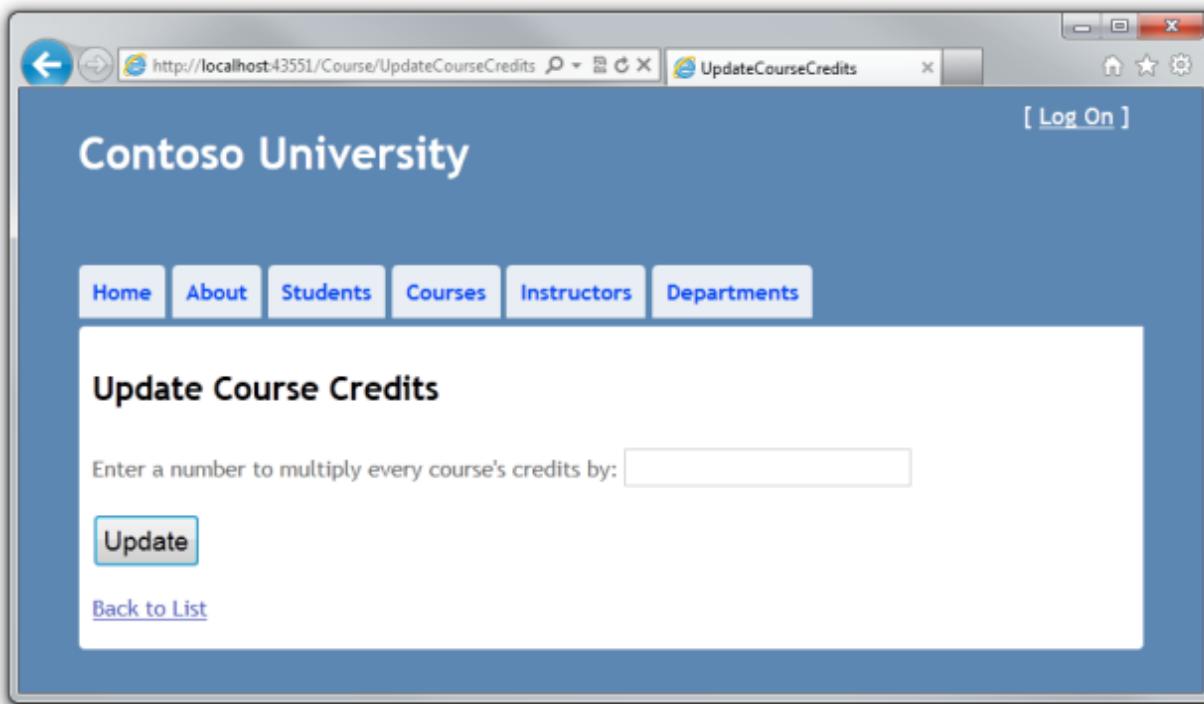
Pada tutorial selanjutnya, kita akan belajar bagaimana menangani skenario-skenario lainnya.

# Kasus-Kasus Lanjutan Entity Framework untuk Aplikasi Web MVC

Pada tutorial ini akan dibahas beberapa topic sebagai berikut :

- Melakukan query dengan Raw SQL.
- Melakukan query no-tracking.
- Memeriksa query yang dikirim ke database.
- Bekerja dengan class-class proxy.
- Menonaktifkan deteksi otomatis perubahan.
- Menonaktifkan validasi saat menyimpan perubahan.

Pada tutorial sebelumnya, kita bekerja dengan halaman-halaman yang telah dibuat sebelumnya. Untuk tutorial yang menjelaskan penggunaan query dengan Raw SQL, kita akan membuat halaman baru untuk mengupdate nomor kredit pada data course di database.



Sedangkan pada bahasan melakukan query non-tracking, kita akan menambahkan logika validasi baru pada halaman Edit Department :

[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Edit

• Instructor Fadi Fakhouri is already administrator of the Mathematics department.

**Department**

Name

Budget

StartDate

Administrator

[Back to List](#)

## Melakukan query dengan Raw SQL.

Pada API Entity Framework terdapat method-method untuk menjalankan perintah SQL langsung ke database. Berikut adalah beberapa pilihan method yang dapat digunakan :

- Method `DbSet.SqlQuery` untuk melakukan query dengan kembalian tipe entity.
- Method `DbDatabase.SqlQuery` untuk melakukan query dengan kembalian yang bukan tipe entity.
- Method `DbDatabase.SqlCommand` untuk menjalankan perintah bukan query.

Salah satu keuntungan menggunakan Entity Framework adalah dapat menghindari kode kita terlalu dengan metode penyimpanan data tertentu, atau terlalu dekat dengan perintah SQL database. Tetapi pada kasus tertentu ada kalanya kode kita harus mengeksekusi query SQL database yang spesifik.

Jika kode kita langsung mengeksekusi perintah SQL database secara langsung dari aplikasi web, maka kita harus melakukan perhatian lebih terhadap kode yang kita tulis agar tidak memiliki celah keamanan serangan SQL Injection. Salah satu solusi untuk permasalah ini adalah dengan menggunakan parameter pada query yang akan dijelaskan pada tutorial bagian ini.

### Memanggil Query yang Mengembalikan Entity

Misalkan pada class `GenericRepository` ingin dimiliki method tambahan untuk proses filter dan pengurutan yang fleksibel tanpa mesti membuat class turunan, maka salah satu solusinya adalah membuat method yang dapat menerima query SQL.

Buat method `GetWithRawSql` pada file `GenericRepository.cs` dengan isi seperti berikut :

```
public virtual IEnumerable< TEntity> GetWithRawSql(string query, params object[] parameters)
{
    return dbSet.SqlQuery(query, parameters).ToList();
}
```

Kemudian pada `CourseController.cs`, kita panggil method tersebut dari method `Detail` seperti contoh berikut ini :

```
public ActionResult Details(int id)
{
    var query = "SELECT * FROM Course WHERE CourseID = @p0";
    return View(unitOfWork.CourseRepository.GetWithRawSql(query, id).Single());
```

```
}
```

Pada kasus seperti ini, sebenarnya kita cukup menggunakan method `GetByID` untuk mendapatkan hasil yang sama, tetapi disini kita akan mencoba method `GetWithRawSql` untuk memastikan method itu dapat bekerja dengan baik.

Untuk memastikan perubahan yang kita buat berjalan dengan baik, maka jalankan site, kemudian pilih tab Course dan pilih Detail pada salah satu course yang ada.

A screenshot of a web browser window displaying the Contoso University website. The browser's address bar shows the URL `http://localhost`. The title bar of the browser window also displays "Details". The main content area features a blue header with the text "[ Log On ]" and the "Contoso University" logo. Below the header is a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. The "Courses" link is highlighted. The main content area has a white background and contains a section titled "Details" under the heading "Course". This section lists course details: Number (2021), Title (Composition), Credits (3), and Department (English). At the bottom of the content area are two links: "Edit" and "Back to List".

[ Log On ]

# Contoso University

Home   About   Students   Courses   Instructors   Departments

## Details

**Course**

Number  
2021

Title  
Composition

Credits  
3

Department  
English

[Edit](#) | [Back to List](#)

## Memanggil Query yang Mengembalikan Tipe Object

Sebelumnya kita telah membuat statistik data student yang menampilkan jumlah student untuk setiap tanggal pendaftaran. Kode tersebut menggunakan LINQ seperti terlihat pada contoh di bawah ini, kode ini terdapat pada file *HomeController.cs*.

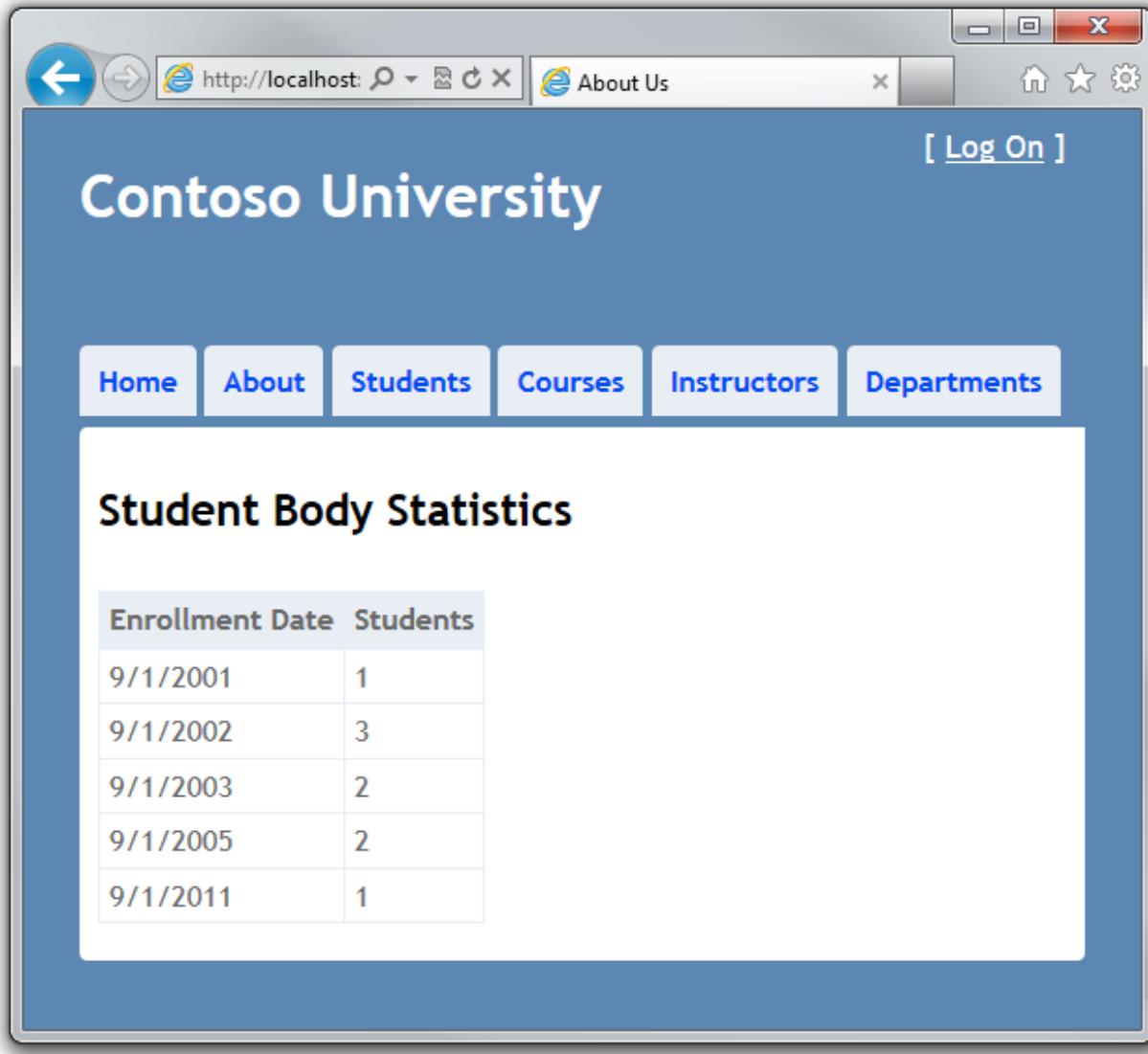
```
var data = from student in db.Students  
          group student by student.EnrollmentDate into dateGroup  
          select new EnrollmentDateGroup()  
  
{  
    EnrollmentDate = dateGroup.Key,  
    StudentCount = dateGroup.Count()  
};
```

Misalkan kita ingin menulis kode yang langsung mengakses langsung data langsung dengan menggunakan query SQL bukan dengan menggunakan LINQ seperti contoh di atas, maka kita dapat menggunakan method `Database.SqlQuery`:

Pada file *HomeController.cs* ganti kode yang ditulis dengan LINQ dengan kode berikut ini :

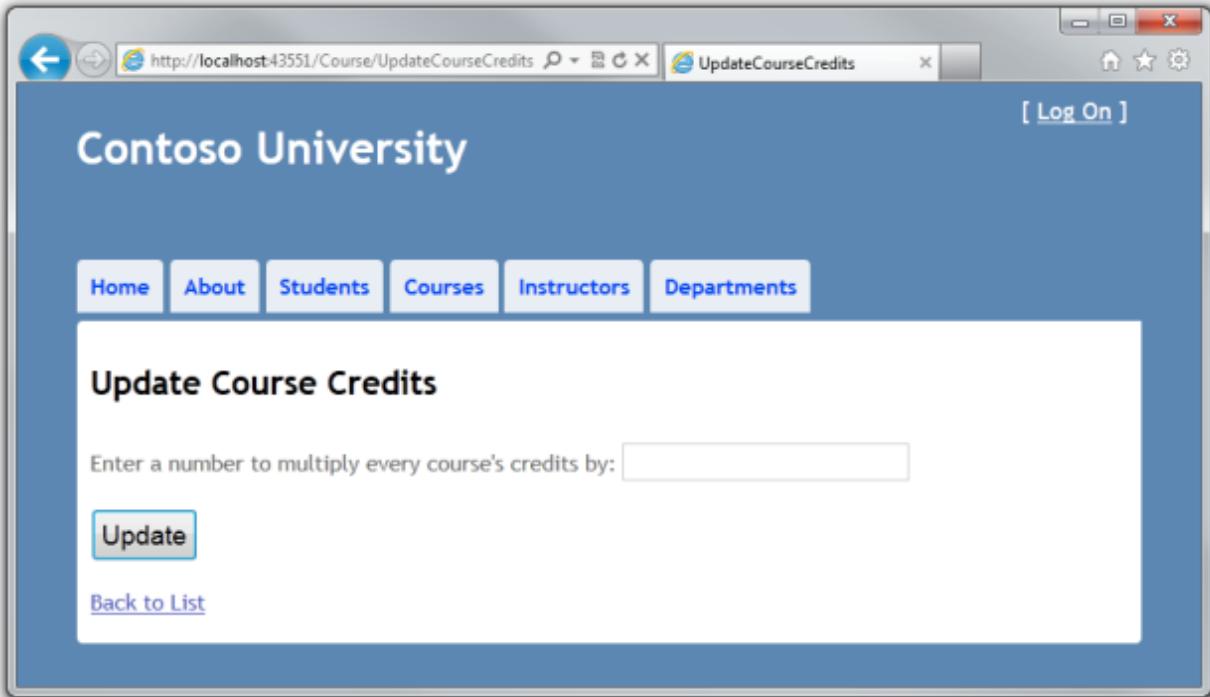
```
var query = "SELECT EnrollmentDate, COUNT(*) AS StudentCount "  
+ "FROM Person "  
+ "WHERE EnrollmentDate IS NOT NULL "  
+ "GROUP BY EnrollmentDate";  
  
var data = db.Database.SqlQuery<EnrollmentDateGroup>(query);
```

Jalankan halaman About, dan dapat kita lihat kita masih mendapatkan informasi yang sama seperti sebelum kode diganti.



## Memanggil Query untuk Update

Pada bagian ini akan dibuat halaman web yang memungkinkan user untuk mengubah nomor kredit untuk semua course, dan untuk proses itu akan digunakan cara dengan mengeksekusi perintah UPDATE SQL. Berikut adalah halaman web yang akan kita buat untuk fungsionalitas tersebut.



Untuk operasi update seperti kasus ini, kita akan repository baru dengan nama class `CourseRepository` yang merupakan turuan dari class `GenericRepository`.

Pada folder *DAL*, buat file `CourseRepository.cs` dan gunakan kode di bawah ini :

```
using System;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public class CourseRepository : GenericRepository<Course>
    {
        public CourseRepository(SchoolContext context)
            : base(context)
        {
        }
    }
}
```

```
public int UpdateCourseCredits(int multiplier)
{
    return context.Database.ExecuteSqlCommand("UPDATE Course SET Credits =
Credits * {0}", multiplier);
}

}

}
```

Pada file `UnitOfWork.cs`, ganti tipe repository `Course` dari `GenericRepository<Course>` menjadi `CourseRepository`.

```
private CourseRepository courseRepository;
```

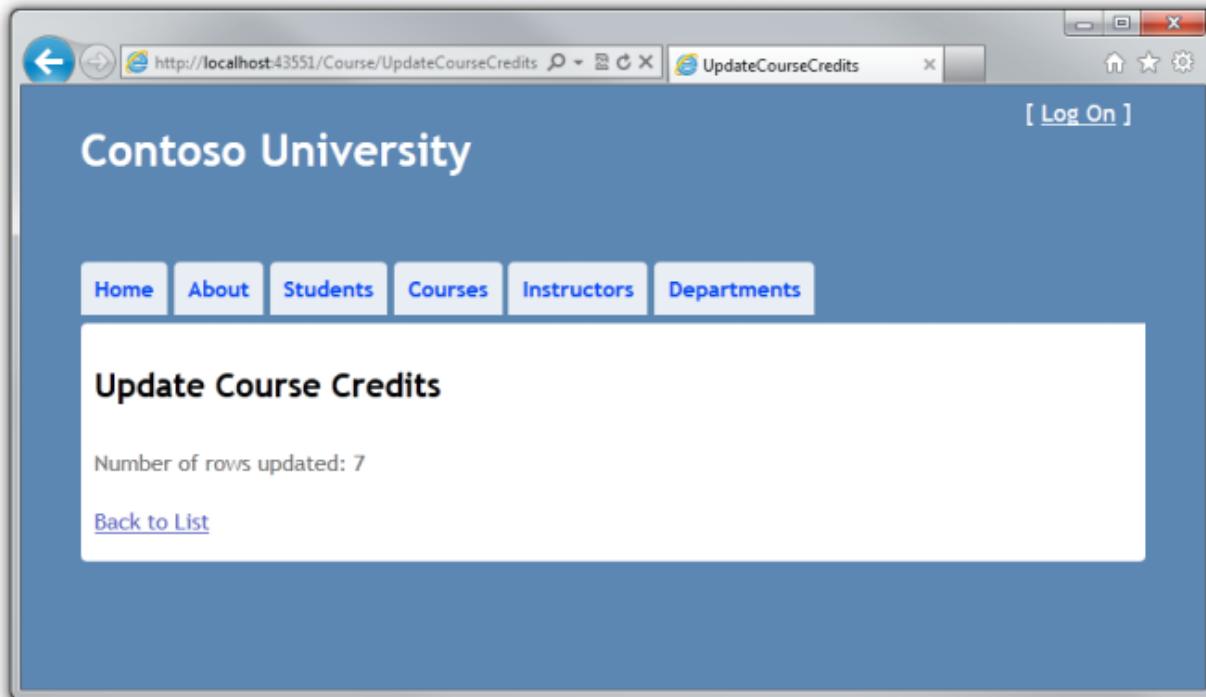
```
public CourseRepository CourseRepository
{
    get
    {
        if (this.courseRepository == null)
        {
            this.courseRepository = new CourseRepository(context);
        }
        return courseRepository;
    }
}
```

Pada `CourseController.cs` tambahkan method `UpdateCourseCredits`:

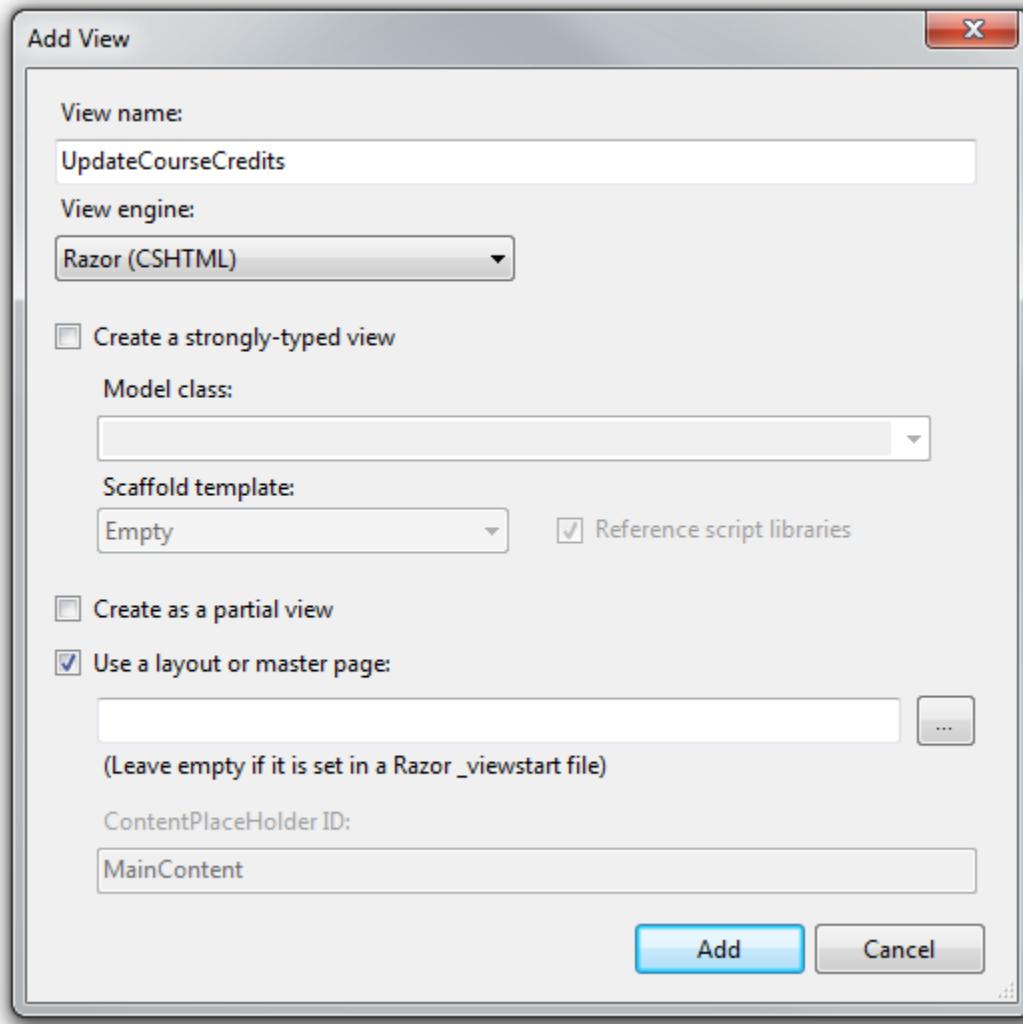
```
public ActionResult UpdateCourseCredits(int? multiplier)
{
    if (multiplier != null)
    {
        ViewBag.RowsAffected =
unitOfWork.CourseRepository.UpdateCourseCredits(multiplier.Value);
    }
    return View();
}
```

Method ini akan digunakan oleh `HttpPost` dan `HttpGet`. Ketika method `HttpGet` `UpdateCourseCredits` dieksekusi, variabel `multiplier` akan bernilai null dan view akan menampilkan textbox kosong dan sebuah tombol submit, seperti yang kita lihat pada gambar di atas.

Ketika tombol `Update` diklik dan method `HttpPost` dijalankan, `multiplier` akan mempunyai nilai sesuai yang diisikan pada textbox. Kemudian akan dipanggil method `UpdateCourseCredit` dan akan mengembalikan jumlah row yang berhasil diupdate, seperti pada gambar di bawah ini.



Buat view pada folder Views\Course untuk halaman Update Course Credit :



Ganti kode pada file Views\Course\UpdateCourseCredits.cshtml dengan kode berikut :

```
@model ContosoUniversity.Models.Course

@{
    ViewBag.Title = "UpdateCourseCredits";
}

<h2>Update Course Credits</h2>
```

```

@if (ViewBag.RowsAffected == null)

{
    using (Html.BeginForm())
    {
        <p>
            Enter a number to multiply every course's credits by:
        @Html.TextBox("multiplier")

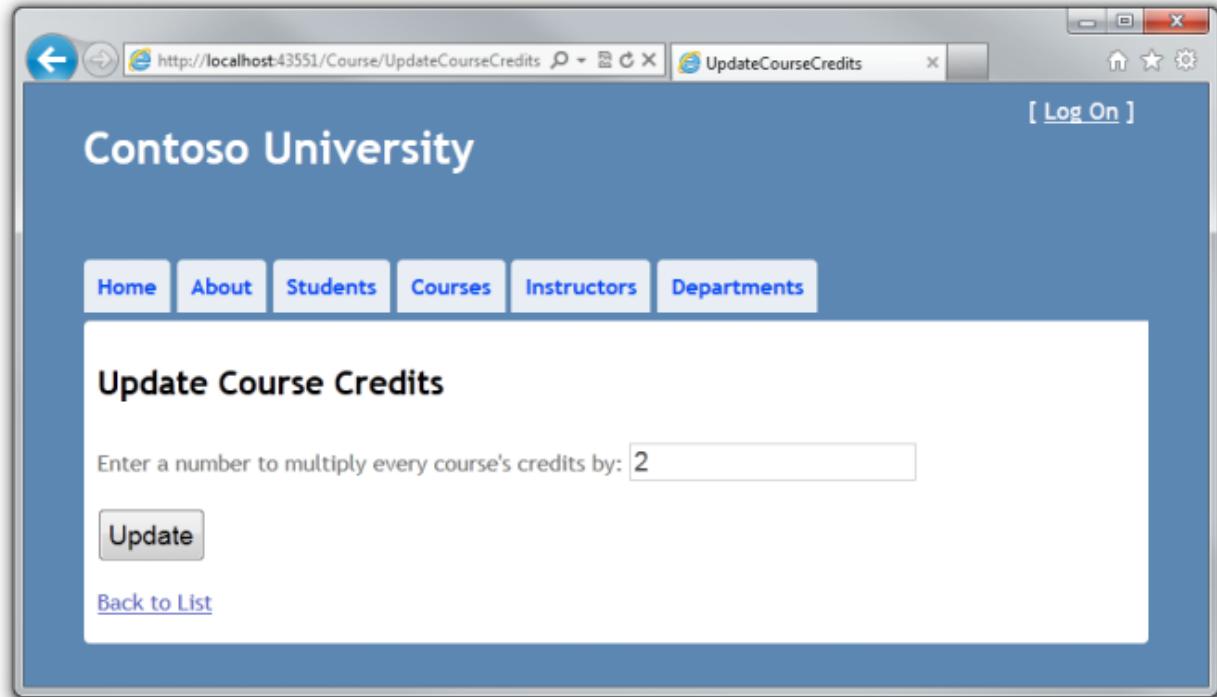
        </p>
        <p>
            <input type="submit" value="Update" />
        </p>
    }
}

@if (ViewBag.RowsAffected != null)
{
    <p>
        Number of rows updated: @ViewBag.RowsAffected
    </p>
}

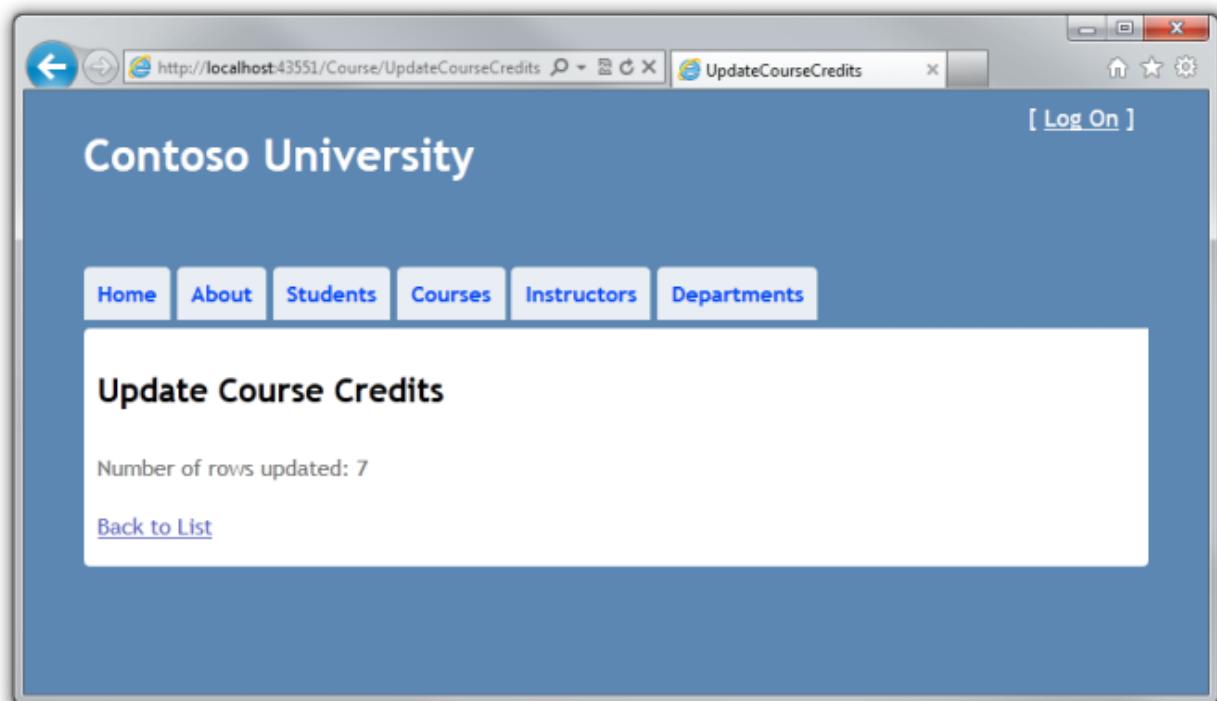
<div>
    @Html.ActionLink("Back to List", "Index")
</div>

```

Jalankan halaman dan pilih tab **Course**, kemudian tambahkan "/UpdateCourseCredits" pada akhir URL pada address bar (sebagai contoh : <http://localhost:50205/Course/UpdateCourseCredits>). Kemudian masukkan angka pada textbox.



Klik tombol **Update** dan dapat dilihat hasilnya seperti berikut.



Klik tombol **Back to List** dan dapat kita lihat hasilnya seperti berikut :

The screenshot shows a web browser window with the URL <http://localhost:43551/Cou> in the address bar. The title bar says "Courses". The main content area displays the "Contoso University" logo and a navigation menu with links: Home, About, Students, Courses, Instructors, and Departments. Below the menu, the word "Courses" is displayed in large bold letters. A link "Create New" is visible. A table lists course information with columns: Number, Title, Credits, and Department. Each row includes edit, details, and delete links.

	Number	Title	Credits	Department
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2021	Composition	6	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2042	Literature	8	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1045	Calculus	8	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	3141	Trigonometry	8	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1050	Chemistry	6	Engineering
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4022	Microeconomics	6	Economics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4041	Macroeconomics	6	Economics

Untuk informasi yang lebih banyak tentang query SQL, dapat dilihat pada post blog Raw SQL Queries (<http://blogs.msdn.com/b/adonet/archive/2011/02/04/using-dbcontext-in-ef-feature-ctp5-part-10-raw-sql-queries.aspx>).

## Melakukan query no-tracking.

Ketika database context mengambil row dan membuat objek entity maka secara default dilakukan pelacakan untuk menentukan apakah entity dalam memory sinkron dengan apa yang ada di database. Data di dalam memory berperan sebagai cache dan digunakan ketika kita memperbarui suatu entity. Caching seperti ini sering tidak diperlukan dalam aplikasi web karena intance context biasanya berumur

pendek karena instance biasanya dibuat saat ada pemintaan dan langsung dihancurkan setelah permintaan selesai dilakukan.

Kita dapat menentukan apakah context melakukan mengaktifkan dan menonaktifkan fitur pelacakan objek entity pada query dengan menggunakan method `AsNoTracking`. Ada beberapa kasus dimana kita mungkin melakukan query no-tracking, yaitu :

- Query yang mengambil data yang sangat besar, maka fitur pelacakan ini dapat dimatikan untuk meningkatkan kinerja.
- Kita ingin menggunakan entity untuk diupdate, tetapi sebelumnya kita sudah menggunakan entity tersebut untuk kebutuhan yang lain. Karena entity tersebut sudah digunakan (tracked) oleh database context maka kita tidak bisa menggunakan (attach) entity tersebut untuk diupdate. Salah satu solusi dari masalah ini adalah dengan menggunakan query dengan opsi `AsNoTracking`.

Pada bagian ini kita akan mengimplementasikan bussiness logic yang diilustrasikan pada kasus kedua di atas.

Pada file `DepartmentController.cs` tambahkan method baru yang memastikan tidak ada department yang mempunyai administrator yang sama, method ini nantinya akan dipanggil dari method `Edit` dan `Create`.

```
private void ValidateOneAdministratorAssignmentPerInstructor(Department department)
{
    if (department.PersonID != null)
    {
        var duplicateDepartment = db.Departments
            .Include("Administrator")
            .Where(d => d.PersonID == department.PersonID)
            .FirstOrDefault();

        if (duplicateDepartment != null && duplicateDepartment.DepartmentID != department.DepartmentID)
        {
            var errorMessage = String.Format(
                "Instructor {0} {1} is already administrator of the {2} department.",
                duplicateDepartment.Administrator.FirstMidName,
                duplicateDepartment.Administrator.LastName,
```

```
        duplicateDepartment.Name);

        ModelState.AddModelError(string.Empty, errorMessage);

    }

}

}
```

Tambahkan kode berikut pada block `try` pada method `HttpPost Edit` untuk memanggil method baru tersebut jika tidak ditemukan error. Berikut adalah isi dari block `try` :

```
if (ModelState.IsValid)

{
    ValidateOneAdministratorAssignmentPerInstructor(department);
}

if (ModelState.IsValid)
{
    db.Entry(department).State = EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Jalankan halaman Department Edit, dan coba untuk mengubah administrator pada department dengan nama administrator yang telah menjadi administrator di department lain. Maka kita akan mendapatkan pesan kesalahan sebagai berikut :

[\[ Log On \]](#)

# Contoso University

[Home](#) [About](#) [Students](#) [Courses](#) [Instructors](#) [Departments](#)

## Edit

• Instructor Fadi Fakhouri is already administrator of the Mathematics department.

**Department**

Name

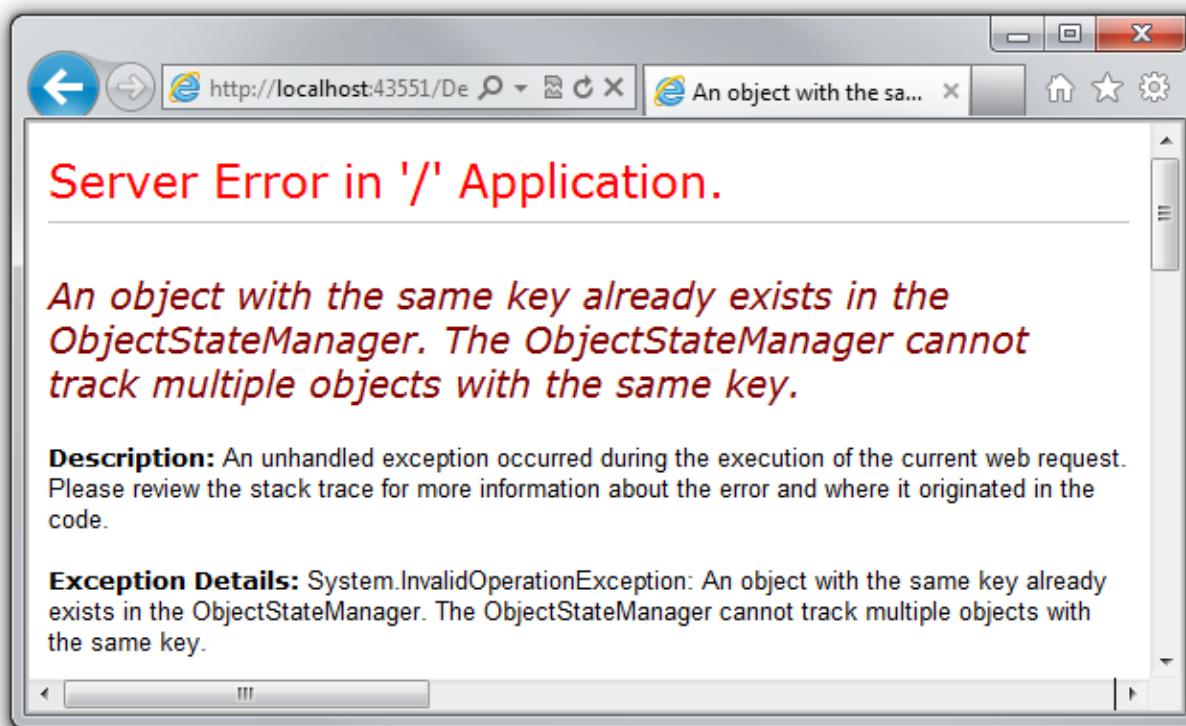
Budget

StartDate

Administrator

[Back to List](#)

Sekarang jalankan kembali halaman Department Edit dan sekarang ganti nilai pada **Budget**. Setelah mengklik tombol Save maka kita akan melihat pesan error seperti berikut :



Pesan error “*An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot track multiple objects with the same key.*” terjadi karena urutan kejadian berikut :

- Method `Edit` memanggil method `ValidateOneAdministratorAssignmentPerInstructor`, yang melakukan pengambilan data seluruh department yang nama administratornya adalah Kim Abercrombie. Sebagai hasil dari operasi ini maka entity English department yang dibaca dari database mempunyai status tracked oleh database context.
- Kemudian method `Edit` melakukan penggantian flag `Modified` pada entity English department yang telah dibuat sebelumnya, hal ini menyebabkan kegagalan karena context telah melacak (tracked) entity English department.

Solusi dari kasus di atas dapat diatasan dengan melakukan langkah berikut. Pada file `DepartmentController.cs` pada method `ValidateOneAdministratorAssignmentPerInstructor` akan ditambahkan method agar query yang dilakukan tidak dilacak (tracking) dengan cara seperti berikut :

```
var duplicateDepartment = db.Departments
```

```
.Include("Administrator")  
.Where(d => d.PersonID == department.PersonID)  
.AsNoTracking()  
.FirstOrDefault();
```

Ulangi langkah untuk melakukan update data Budget pada department, seperti yang telah kita lakukan di atas. Maka akan kita lihat proses yang kita lakukan saat ini tidak menampilkan error lagi seperti sebelumnya.

## Memeriksa query yang dikirim ke database.

Terkadang akan sangat membantu jika kita bisa melihat query SQL yang dikirimkan ke database. Untuk melihat query tersebut kita dapat menggunakan variabel query dengan menggunakan debugger atau menampilkannya dengan menggunakan method `ToString()`.

Pada `Controller/CourseController` ganti method Index dengan kode berikut :

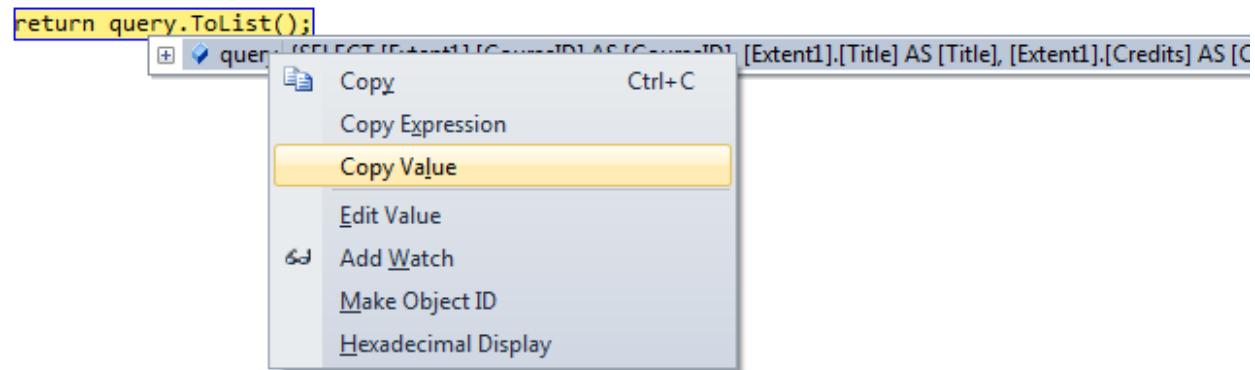
```
public ViewResult Index()  
{  
    var courses = unitOfWork.CourseRepository.Get();  
    return View(courses.ToList());  
}
```

Kemudian pada file `GenericRepository.cs` set breakpoint pada baris `return query.ToList();` dan pada baris `return orderBy(query).ToList();` pada method `Get`. Jalankan project pada mode debug dan pilih halaman Course Index. Saat kode pada breakpoint yang telah kita tentukan dijalankan periksa variabel `query`. Maka kita akan dapat melihat pernah SQL seperti berikut :

```
{SELECT  
[Extent1].[CourseID] AS [CourseID],  
[Extent1].[Title] AS [Title],  
[Extent1].[Credits] AS [Credits],  
[Extent1].[DepartmentID] AS [DepartmentID]  

```

Query yang ditampilkan mungkin bisa sangat panjang untuk ditampilkan pada window debugging di Visual Studio, untuk melihat seluruh query tersebut kita dapat menyalin nilai variabel tersebut ke text editor dengan cara berikut ini :



Sekarang kita akan tambahkan dropdown list pada halaman Course Index sehingga user dapat melakukan filter berdasarkan department. Kita akan mengurutkan data berdasarkan title course dan kita akan menggunakan eager loading pada property navigation `Department`.

Pada file `CourseController.cs`, ganti method `Index` dengan kode berikut :

```
public ActionResult Index(int? SelectedDepartment)
{
    var departments = unitOfWork.DepartmentRepository.Get(
        orderBy: q => q.OrderBy(d => d.Name));

    ViewBag.SelectedDepartment = new SelectList(departments, "DepartmentID", "Name",
SelectedDepartment);

    int departmentID = SelectedDepartment.GetValueOrDefault();

    return View(unitOfWork.CourseRepository.Get(
        filter: d => !SelectedDepartment.HasValue || d.DepartmentID == departmentID,
        orderBy: q => q.OrderBy(d => d.CourseID),
        includeProperties: "Department"));
}
```

Method ini akan menerima nilai yang dipilih dari dropdown list yang akan dimasukkan menjadi nilai parameter `SelectedDepartment`.

Collection `SelectList` berisi seluruh department dikirimkan ke view untuk ditampilkan pada dropdown list. Parameter-parameter yang dikirimkan ke constructor `SelectList` adalah nilai dari nama value field, nama text field dan item yang dipilih.

Pada method Get dari repository `Course`, terdapat kode untuk menentukan untuk ekspresi filter, pengurutan dan eager loading untuk property navigation `Department`. Ekspresi filter akan selalu mengembalikan nilai `true` jika tidak ada item yang dipilih pada dropdown list.

Pada file `Views\Course\Index.cshtml`, sebelum tag awal table tambahkan kode berikut untuk membuat dropdown list dan tombol submit.

```
@using (Html.BeginForm())
{
    <p>Select Department: @Html.DropDownList("SelectedDepartment", "All")
    <input type="submit" value="Filter" /></p>
}
```

Dengan breakpoint yang masih ditentukan sebelumnya pada class `GenericRepository`, jalankan halaman Course Index. Kemudian pilih department yang diinginkan dan klik tombol **Filter**.

The screenshot shows a web browser window with the URL <http://localhost:43551/Courses>. The page title is "Courses". At the top, there is a navigation bar with links for Home, About, Students, Courses, Instructors, and Departments. A "Log On" link is also present. Below the navigation bar, the main content area displays the heading "Courses" and a "Create New" link. A dropdown menu labeled "Select Department: Economics" is shown, along with a "Filter" button. A table lists seven courses with columns for Number, Title, Credits, and Department. Each row includes edit, details, and delete links.

	Number	Title	Credits	Department
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1045	Calculus	8	Mathematics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	1050	Chemistry	6	Engineering
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2021	Composition	6	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	2042	Literature	8	English
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4041	Macroeconomics	6	Economics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	4022	Microeconomics	6	Economics
<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>	3141	Trigonometry	8	Mathematics

Pada waktu breakpoint pertama, yang terjadi adalah proses query department untuk dropdown list. Lewati hal itu dan pada breakpoint kedua terjadi lihat nilai variabel `query`, dan kita akan dapat melihat query SQL seperti berikut :

```
{SELECT  
[Extent1].[CourseID] AS [CourseID],
```

```

[Extent1].[Title] AS [Title],
[Extent1].[Credits] AS [Credits],
[Extent1].[DepartmentID] AS [DepartmentID],
[Extent2].[DepartmentID] AS [DepartmentID1],
[Extent2].[Name] AS [Name],
[Extent2].[Budget] AS [Budget],
[Extent2].[StartDate] AS [StartDate],
[Extent2].[PersonID] AS [PersonID],
[Extent2].[Timestamp] AS [Timestamp]
FROM [Course] AS [Extent1]
INNER JOIN [Department] AS [Extent2] ON [Extent1].[DepartmentID] = [Extent2].[DepartmentID]
WHERE (@p__linq_0 IS NULL) OR ([Extent1].[DepartmentID] = @p__linq_1)}

```

Dapat dilihat query di atas merupakan query **JOIN** yang mengambil data **Department** dan **Course** dan terdapat klausa **WHERE** pada query tersebut.

## Bekerja dengan class-class proxy.

Ketika Entity Framework membuat instance entity (sebagai contoh, ketika kita menjalankan query), sering kali dibuat instance dari sebuah tipe turunan yang dihasilkan secara otomatis yang berfungsi sebagai proxy untuk entity. Proxy ini meng-override virtual property dari entitas untuk memasukkan hook yang akan melakukan tindakan otomatis ketika property diakses. Sebagai contoh, mekanisme ini dipakai untuk mendukung lazy loading.

Dalam prakteknya kita tidak perlu memperdulikan penggunaan proxy ini, tetapi ada beberapa pengecualian :

- Pada kasus tertentu kita mungkin ingin mencegah Entity Framework membuat intance proxy. Sebagai contoh, proses serialize intance non-proxy lebih effisien dibandingkan melakukan serialize intance proxy.
- Ketika kita membuat instance dari class entity dengan operator new, tetapi kita tidak mendapatkan instance proxy. Hal ini bisa terjadi jika entity yang dibuat tidak berada di dalam database.
- Jika kita ingin mendapatkan tipe entity sebenarnya dari tipe proxy, maka kita bisa gunakan method **GetObjectTipe** dari class **ObjectContext**.

Untuk informasi lebih lanjut mengenai ini bisa mengunjungi post blog Working with Proxies (<http://blogs.msdn.com/b/adonet/archive/2011/02/02/using-dbcontext-in-ef-feature-ctp5-part-8-working-with-proxies.aspx>).

## **Menonaktifkan deteksi otomatis perubahan.**

Entity Framework dapat menentukan suatu entity sudah berubah dengan cara membandingkan nilai-nilai saat ini suatu entity dengan nilai-nilai aslinya. Nilai-nilai asli di dapat ketika suatu entity digunakan di awal. Method-method yang dapat menyebabkan deteksi perubahan secara otomatis adalah sebagai berikut :

- `DbSet.Find`.
- `DbSet.Local`.
- `DbSet.Remove`.
- `DbSet.Add`.
- `DbSet.Attach`.
- `DbContext.SaveChanges`.
- `DbContext.GetValidationErrors`.
- `DbContext.Entry`.
- `DbChangeTracker.Entries`.

Jika kita melakukan pelacakan sejumlah besar entity dan memanggil salah satu method di atas dalam suatu pengulangan, kita mungkin akan mendapatkan peningkatan kinerja yang signifikan jika untuk sementara kita mematikan deteksi perubahan secara otomatis ini, caranya dengan menggunakan property `AutoDetectChangesEnabled` ([http://msdn.microsoft.com/en-us/library/system.data.entity.infrastructure.dbcontextconfiguration.autodetectchangesenabled\(VS.103\).aspx](http://msdn.microsoft.com/en-us/library/system.data.entity.infrastructure.dbcontextconfiguration.autodetectchangesenabled(VS.103).aspx)). Untuk informasi lebih lanjut dapat membaca post blog [Automatically Detecting Changes](http://blogs.msdn.com/b/adonet/archive/2011/02/06/using-dbcontext-in-ef-feature-ctp5-part-12-automatically-detecting-changes.aspx) (<http://blogs.msdn.com/b/adonet/archive/2011/02/06/using-dbcontext-in-ef-feature-ctp5-part-12-automatically-detecting-changes.aspx>).

## **Menonaktifkan validasi saat menyimpan perubahan.**

Saat kita memanggil method `SaveChanges`, secara default Entity Framework akan melakukan validasi atas semua data pada seluruh property-property dari seluruh entity yang diubah sebelum melakukan update ke database. jika kita ingin mematikan fitur ini maka kita dapat mempergunakan property `ValidateOnSaveEnabled` ([http://msdn.microsoft.com/en-us/library/system.data.entity.infrastructure.dbcontextconfiguration.validateonsaveenabled\(VS.103\).aspx](http://msdn.microsoft.com/en-us/library/system.data.entity.infrastructure.dbcontextconfiguration.validateonsaveenabled(VS.103).aspx)). Untuk informasi lebih lanjut tentang ini dapat mengunjungi blog berikut ini Validation (<http://blogs.msdn.com/b/adonet/archive/2010/12/15/ef-feature-ctp5-validation.aspx>).

## **Materi-Materi Entity Framework**

Berikut adalah materi-materi tentang Entity Framework :

- Introduction to the Entity Framework 4.1 (Code First) ([http://msdn.microsoft.com/en-us/library/gg696172\(VS.103\).aspx](http://msdn.microsoft.com/en-us/library/gg696172(VS.103).aspx)).
- The Entity Framework Code First Class Library API Reference ([http://msdn.microsoft.com/en-us/library/gg696095\(VS.103\).aspx](http://msdn.microsoft.com/en-us/library/gg696095(VS.103).aspx)).

- Entity Framework FAQ (<http://social.technet.microsoft.com/wiki/contents/articles/entity-framework-faq.aspx>).
- The Entity Framework Team Blog (<http://blogs.msdn.com/b/adonet/>).
- Entity Framework in the MSDN Library (<http://msdn.microsoft.com/en-us/library/bb399572.aspx>).
- Entity Framework in the MSDN Data Developer Center (<http://msdn.microsoft.com/data/ef>).
- Entity Framework Forums on MSDN (<http://social.msdn.microsoft.com/forums/en-US/adodotnetentityframework/>).
- Julie Lerman's blog (<http://thedatafarm.com/blog/>).
- Code First DataAnnotations Attributes (<http://msdn.microsoft.com/en-us/data/gg193958>).
- Maximizing Performance with the Entity Framework in an ASP.NET Web Application (<http://www.asp.net/entity-framework/tutorials/maximizing-performance-with-the-entity-framework-in-an-asp-net-web-application>).
- Profiling Database Activity in the Entity Framework (<http://msdn.microsoft.com/en-us/magazine/gg490349.aspx>).
- Entity Framework Power Tools (<http://blogs.msdn.com/b/adonet/archive/2011/05/18/ef-power-tools-ctp1-released.aspx>).

Selain itu juga bisa mengunjungi posting pada blog Entity Framework Team berikut ini untuk mendapatkan informasi lanjutan dari topik-topik yang ada pada tutorial seri ini :

- Fluent API Samples (<http://blogs.msdn.com/b/adonet/archive/2010/12/14/ef-feature-ctp5-fluent-api-samples.aspx>).
- Connections and Models (<http://blogs.msdn.com/b/adonet/archive/2011/01/27/using-dbcontext-in-ef-feature-ctp5-part-2-connections-and-models.aspx>).
- Pluggable Conventions (<http://blogs.msdn.com/b/adonet/archive/2011/01/10/ef-feature-ctp5-pluggable-conventions.aspx>).
- Finding Entities (<http://blogs.msdn.com/b/adonet/archive/2011/01/28/using-dbcontext-in-ef-feature-ctp5-part-3-finding-entities.aspx>).
- Loading Related Entities (<http://blogs.msdn.com/b/adonet/archive/2011/01/31/using-dbcontext-in-ef-feature-ctp5-part-6-loading-related-entities.aspx>).
- Load and AsNoTracking (<http://blogs.msdn.com/b/adonet/archive/2011/02/05/using-dbcontext-in-ef-feature-ctp5-part-11-load-and-asnotracking.aspx>).

Untuk mendapatkan informasi penggunaan LINQ dengan Entity Framework dapat mengunjungi LINQ to Entities (<http://msdn.microsoft.com/en-us/library/bb386964.aspx>) pada MSDN Library.

Tutorial yang menggunakan lebih banyak lagi fitur-fitur MVC dan Entity Framework dan melihat MVC Music Store (<http://www.asp.net/mvc/tutorials/mvc-music-store-part-1>).

Untuk informasi bagaimana proses deploy aplikasi web yang kita buat dapat mengunjungi MSDN Library berikut ASP.NET Deployment Content Map (<http://msdn.microsoft.com/en-us/library/bb386521.aspx>).