بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

خواندن و نمایش داده های بار گذاری شده در Navigation Property

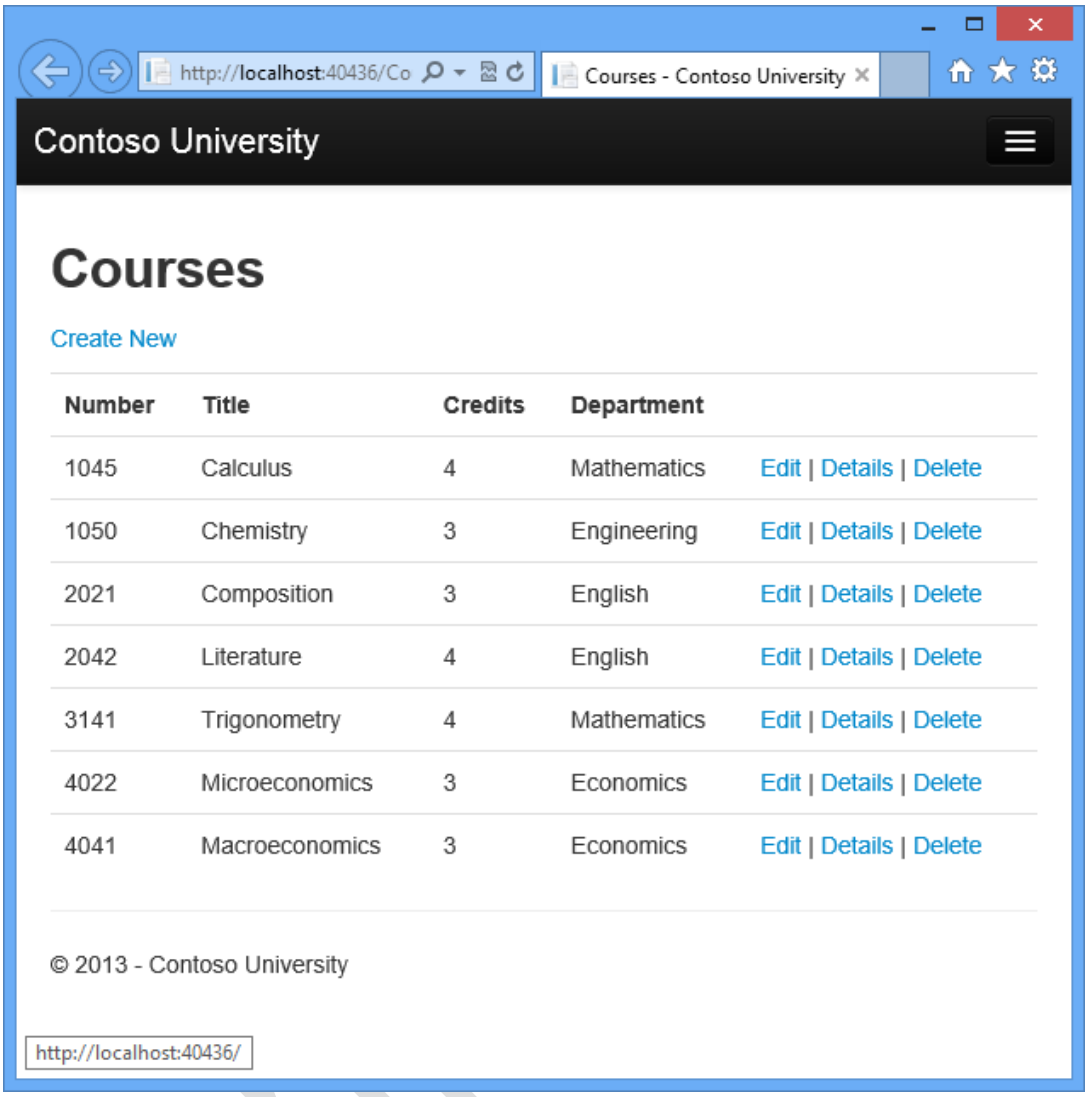
مدرس : مهندس افشین رفوآ

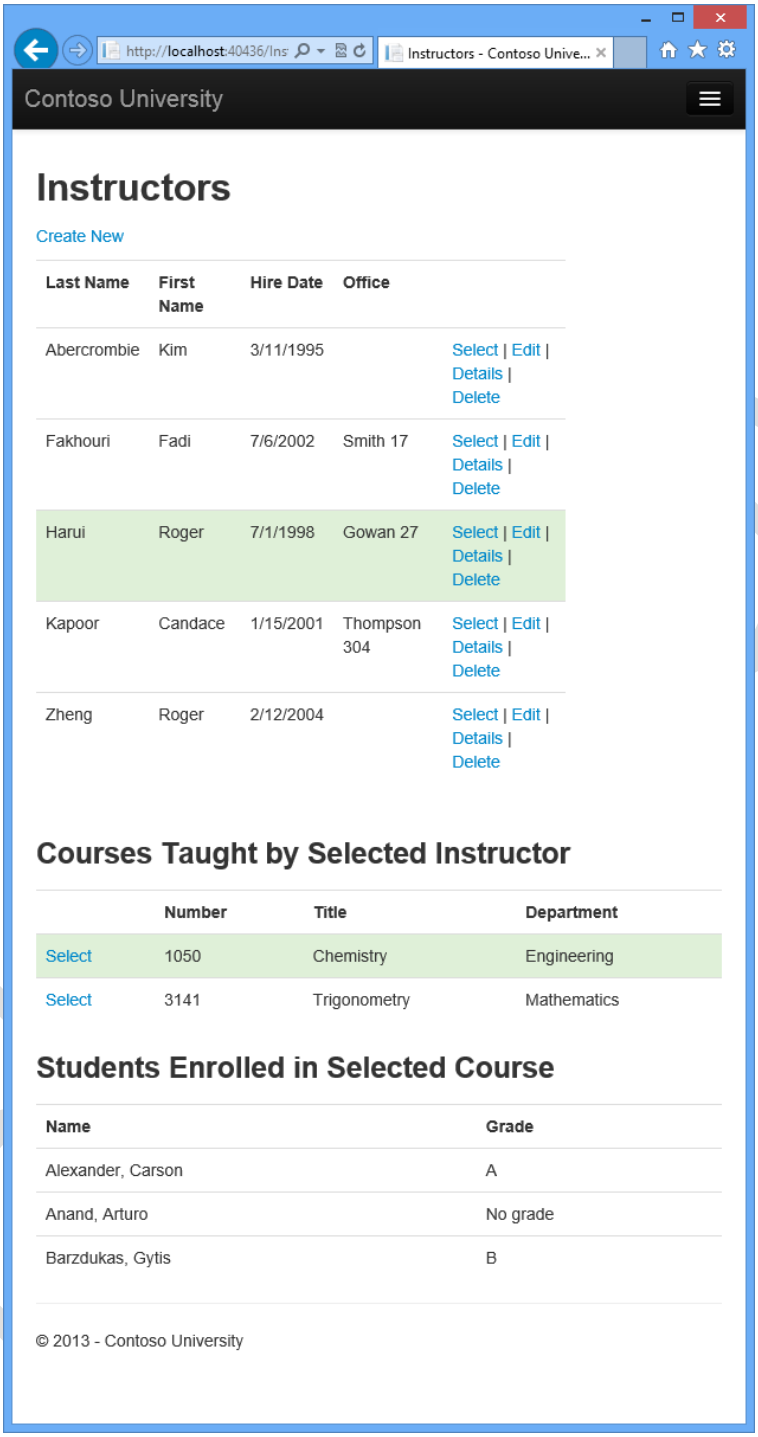
دوره آموزش MVC

خواندن و نمایش داده های بار گذاری شده در Navigation Property با EF در ASP.NET MVC

در آموزش قبلی data model ای که Student نام گذاری شده بود را تکمیل کردیم. در این آموزش داده های مربوط – داده هایی که EF در navigation property لود می کند– را خوانده و نمایش می دهیم.

تصویر زیر صفحاتی را که با آن ها کار خواهید کرد را نمایش می دهد.


http://localhost:40436/Co
Courses - Contoso University
Contoso University
Courses
Create New
Number Title Credits Department
1045 Calculus 4 Mathematics Edit | Details | Delete
1050 Chemistry 3 Engineering Edit | Details | Delete
2021 Composition 3 English Edit | Details | Delete
2042 Literature 4 English Edit | Details | Delete
3141 Trigonometry 4 Mathematics Edit | Details | Delete
4022 Microeconomics 3 Economics Edit | Details | Delete
4041 Macroeconomics 3 Economics Edit | Details | Delete
© 2013 - Contoso University
http://localhost:40436/



Lazy، Eager و Explicit Loading داده های بارگذاری شده در navigation property ها

روش های مختلفی وجود دارد که EF می تواند داده های مربوط را در navigation property یک موجودیت بارگذاری کند:

1. Lazy Loading (بار گذاری با تاخیر). هنگامی که entity مربوطه برای اولین بار خوانده می شود، داده های مربوط همراه با آن بازیابی نمی شوند. اما اولین باری که تلاش می کنید به یک navigation property دسترسی داشته باشید، داده های مورنیاز آن navigation property به صورت خودکار بازیابی می شوند. در این حالت چندین query به پایگاه داده ارسال می شود – یکی برای خود موجودیت مورد نظر و دیگری هر بار که داده های مربوطه ی آن موجودیت بایستی بازگردانی شود. کلاس DbContext به صورت پیش فرض قابلیت Lazy Loading را فعال می سازد.

```
departments = context.Departments
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

2. Eager Loading (بارگذاری یکجا). هنگامی که موجودیت خوانده می شود، داده های مربوطه همراه با آن بازیابی می شوند. در این حالت تنها یک join query به پایگاه داده فرستاده می شود که طی آن کلیه ی اطلاعات مورد نیاز بازیابی می شوند. می توان به وسیله ی متد Include، قابلیت eager loading را تعریف کرد و مورد استفاده قرار داد.

```
departments = context.Departments.Include(x => x.Courses)
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows and related Course rows

3. Explicit loading (بارگذاری با تاخیر صریح). بسیار شبیه به lazy loading است با این تفاوت که شما در آن داده های مربوطه را به طور صریح در کد بازیابی می کنید؛ بدین معنا که هنگام دسترسی به navigation property این بازیابی به صورت خودکار اتفاق نمی افتد. داده های مربوطه را به صورت دستی و با بازیابی object state manager entry برای یک موجودیت و فراخوانی متد Collection.Load برای مجموعه ها یا متد Reference.Load برای property هایی که یک تک

موجودیت را نگه می دارند، بارگذاری کنید. (در مثال زیر اگر می خواهید navigation property را بار گذاری کنید، می بایست Collection(x => x.Courses) را با Reference(x => x.Administrator) جایگزین کنید). از explicit loading معمولا زمانی استفاده می شود که قبلش lazy loading را غیرفعال کرده باشید.

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

Query: all Department rows

Query: Course rows related to Department d

از آنجایی که هر دو حالت بارگذاری مقادیر property ها را با تاخیر لود می کنند، هر دو به عنوان deferred loading (برگذاری دیرهنگام) نیز شناخته می شود.

ملاحظاتی درباره ی کارایی

اگر می دانید که به داده های مرتبط با هر entity بازیابی شده احتیاج پیدا خواهید کرد، در آن صورت استفاده از eager loading بهترین گزینه بوده و بالاترین کارایی را ارائه می دهد. دلیل آن هم این است که ارسال یک query به پایگاه داده برای بازیابی هر entity کارامد تر از چندین query جدا به ازای هر entity می باشد. به عنوان نمونه، در مثال های بالا، فرض بگیرید هر دپارتمان دارای 10 دوره ی آموزشی (course) مربوطه می باشد. در مثال eager loading، تنها یک query (join) به پایگاه داده ارسال می شد و فقط یک رفت و برگشت (round trip) صورت می گرفت. در حالی که اگر به مثال های lazy loading و explicit loading مراجعه کنید می بینید که در هر کدام یازده query به پایگاه داده فرستاده شده و یازده رفت و برگشت به آن انجام پذیرفته. رفت و برگشت های اضافی بر سازمان در شرایطی که latency (نهفتگی) بالا است، باعث افت شدید کارایی می شود.

از سوی دیگر، می توان گفت در برخی شرایط lazy loading کارآمدتر است. eager loading ممکن است باعث ایجاد یک join پیچیده شود که SQL Server قادر به پردازش کارآمد آن نیست. همچنین در مواقعی که نیاز

دارید فقط به navigation property های یک موجودیت از میان مجموعه ای از entity های در حال پردازش دسترسی داشته باشید، Lazy loading گزینه ی برتر می باشد، زیرا eager loading داده های بیشتری را نسبت به نیاز شما بازیابی می کند که در نهایت منجر به افت کارایی می شود. چنانچه میزان کارایی از اهمیت خاصی برخوردار است، توصیه می شود کارایی را به هر دو روش تست کرده و بهترین گزینه را انتخاب نمایید.

Lazy loading می تواند آن کدهایی که افت کارایی به دنبال دارند را مخفی سازد. برای مثال، آن کدی که روش بارگذاری (lazy loading و eager loading) را مشخص نمی کند ولی تعداد زیادی entity را پردازش کرده و navigation property های متعددی را در هر گام تکرار مورد استفاده قرار می دهد، به خاطر رفت و برگشت های مکرر به پایگاه داده، بسیار ناکارآمد باشد. یک برنامه ممکن است زمانی که هنوز در دست توسعه بوده و از سیستم های مستقر بر سرورهای محلی (on-premise SQL Server) استفاده می کند دارای کارایی مناسب باشد اما به هنگام انتقال آن به Azure SQL Database، به خاطر lazy loading و افزایش زمان تاخیر (latency) با افت کارایی مواجه شود. نظارت بر اجرای query ها و ردگیری آن ها (profiling) از طریق یک load test (یک نوع تست کارایی نرم افزار که با فشار آوردن و اضافه بارگذاری سیستم نرم افزاری، پاسخ سیستم و کارایی آن را می سنجد.) واقعی به شما کمک می کند دریابید آیا lazy loading مناسب نیاز شما هست یا خیر.

غیرفعال سازی lazy loading پیش از serialization

فرایند تبدیل یک شی به رشته ای از بایت ها جهت ذخیره در فایل را serialization می گویند. چنانچه lazy loading را پیش از serialization غیرفعال نکرده باشید، در آن صورت ممکن است از داده های بیشتری (نسبت به مقدار مدنظر) query بگیرید. Serialization با دسترسی به نمونه propertyها از یک نوع کار می کند. دسترسی به property باعث می شود قابلیت lazy loading فعال شده و آن موجودیت هایی که توسط lazy loading بار گذاری شده اند، serialize می شوند. فرایند serialization سپس به هر یک از property های موجودیت هایی که با lazy loading بارگذاری شده اند، دسترسی پیدا می کند، که در نهایت سبب بارگذاری با تاخیر و serialization بیشتر می شود. برای برطرف ساختن این مشکل، می بایست lazy loading را پیش از serialize کردن موجودیت غیرفعال کنید.

Serialization همچنین ممکن است توسط کلاس های proxy که EF مورد استفاده قرار می دهد، پیچیده تر شود.

یکی از روش های برطرف ساختن مشکلات ناشی از serialization این است که DTO ها (شی انتقال داده بین پروسه ها) را بجای entity object ها، serialize کنیم.

روش های غیرفعال سازی lazy loading:

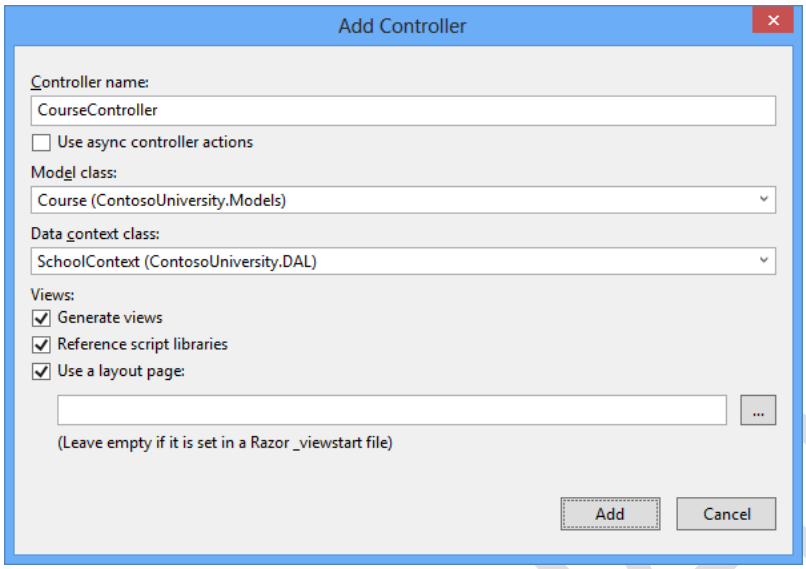
1. هنگام اعلان برخی از navigation property ها، کلیدواژه ی virtual را از تعریف آن حذف کنید.
2. خاصیت LazyLoadingEnabled را در تعریف تمامی navigation property ها بر روی false تنظیم کرده و کد زیر را در constructor (سازنده) کلاس context جای گذاری کنید:

```
this.Configuration.LazyLoadingEnabled = false;
```

ایجاد یک صفحه به نام Courses که اسم Department را نمایش می دهد

موجودیت Course یک navigation property را شامل می شود که خود دربردارنده ی آن موجودیت Department دپارتمانی است که دوره ی آموزشی (course) به آن تخصیص یافته است. جهت نمایش اسم دپارتمان تخصیص یافته در فهرست دوره ها، بایستی خاصیت Name را از موجودیت Department موجود در navigation property به نام Course.Department بازیابی کنید.

یک controller ویژه ی entity type که نام آن Course هست، ایجاد کرده و آن را CourseController (نه CoursesController) نام گذاری کنید. این کار را با استفاده از همان گزینه های MVC 5 Controller with views, using Entity Framework scaffolder که پیش تر برای ایجاد کنترلر Student مورد استفاده قرار دادید، انجام دهید.



فایل Controllers\CourseController.cs را باز کرده و متد Index را بررسی کنید:

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

Scaffolding ای که خودکار اجرا می شود، با استفاده از متد Include، قابلیت eager loading را برای خاصیت Department فعال می سازد.

فایل Views\Course\Index.cshtml را باز کرده و کد template را با کد زیر جایگزین کنید. این تغییرات در مثال زیر هایلایت شده اند:

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

    @{
  ViewBag.Title = "Courses";
    }

    <h2>Courses</h2>

    <p>
      @Html.ActionLink("Create New", "Create")
    </p>
    <table class="table">
```

```
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.CourseID)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Credits)
            </th>
            <th>
                Department
            </th>
            <th></th>
        </tr>

    @foreach (var item in Model)
    {
        <tr>
            <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>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id = item.CourseID }) |
                @Html.ActionLink("Details", "Details", new { id = item.CourseID }) |
                @Html.ActionLink("Delete", "Delete", new { id = item.CourseID })
            </td>
        </tr>
    }

</table>
```

شما تغییرات زیر را به کدی که توسط scaffolding ایجاد شده، اعمال کرده اید:

1. سرستون (heading) را از Index به Courses تغییر دادید.
2. یک ستون به نام Number اضافه کردید که مقدار خاصیت CourseID را نمایش می دهد. از آنجایی که primary key ها برای کاربران بی معنی به نظر می رسد، (primary keyها) به صورت پیش فرض

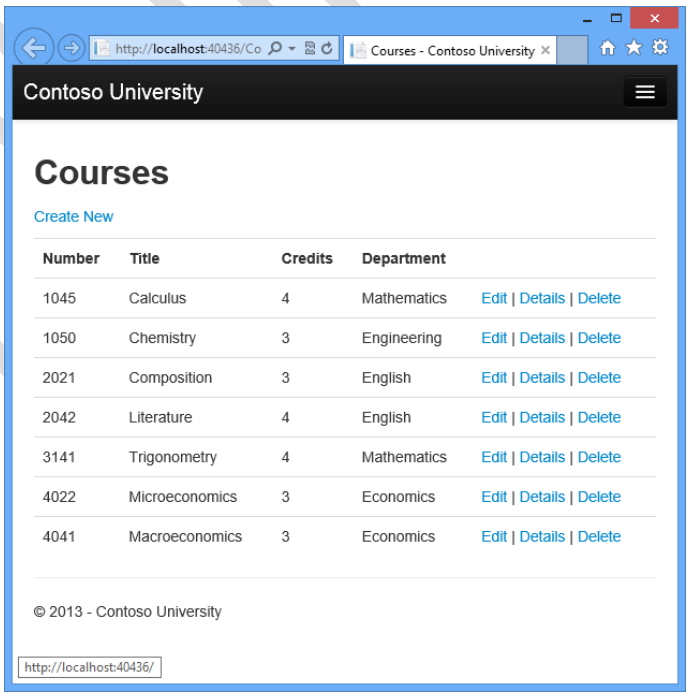
توسط پروسه ی scaffolding ایجاد نمی شوند. اما در این مثال، primary key معنی دار بوده و نمایش آن توصیه می شود.

3. ستون Department را به سمت راست انتقال داده و سرستون آن را ویرایش کردید. Scaffolder خاصیت Name را از موجودیت Department خوانده و نمایش می دهد. اما در این مثال، در صفحه ی Course page، سرستون بایستی بجای Name، Department باشد.

همان طور که در ستون Department مشاهده می کنید، کدی که توسط scaffolding ایجاد شده، خاصیت Name الصاق شده به موجودیت Department را نمایش می دهد. این property در خاصیت پیمایشی (navigation property) Department بارگذاری شده:

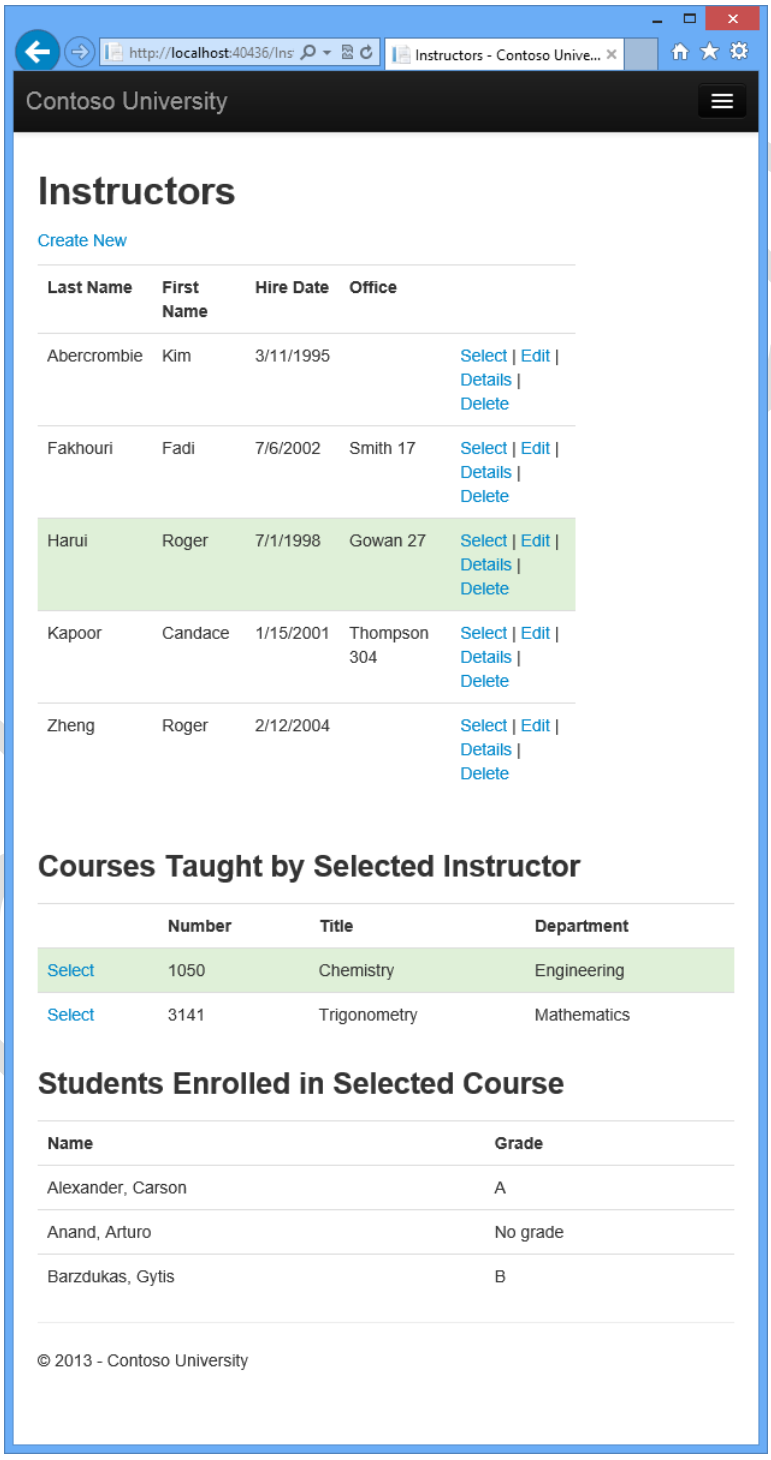
```
<td>
@Html.DisplayFor(modelItem => item.Department.Name)
</td>
```

صفحه را اجرا کرده (تب Courses را در صفحه ی اصلی Contoso University انتخاب کنید) تا لیست دربردارنده ی اسامی درپارتمان ها را مشاهده نمایید.



ایجاد یک صفحه به نام Instructors جهت نمایش دوره های آموزشی و ثبت نام دانشجویان

در این بخش یک controller و view ویژه ی موجودیت Instructor برای نمایش صفحه ی Instructor ایجاد می کنیم:



این صفحه داده های مربوطه را به شیوه های زیر خوانده و نمایش می دهد:

1. فهرستی که لیست مدرسان دانشگاه را نمایش می دهد، داده های مربوطه را از موجودیت OfficeAssignment می خواند. موجودیت های Instructor و OfficeAssignment در یک رابطه ی یک به صفر یا یک به یک مشارکت دارند. برای موجودیت های OfficeAssignment از قابلیت eager loading استفاده می کنیم. همان طور که قبل تشریح شد، در مواقعی که داده های مربوطه ی تمام سطرهای بازیابی شده از جدول اصلی (primary table) را احتیاج دارید، eager loading بهترین گزینه می باشد. در این نمونه، می خواهیم office assignment های تمامی instructor های نمایش داده شده را نشان دهیم.
2. پس از اینکه کاربر یک instructor را انتخاب می کند، تمامی موجودیت های مربوطه ی Course نمایش داده می شوند. موجودیت های instructor و Course با هم یک رابطه ی چند به چند دارند. برای موجودیت های Course و همچنین تمامی موجودیت های Department مربوطه ی آن، از eager loading استفاده می کنیم. لازم به ذکر است که در این نمونه ی خاص، lazy loading از کارایی بیشتری برخوردار می باشد زیرا تنها به دوره های آموزشی (course) مدرس (instructor) انتخابی احتیاج است. با این وجود، مثال حاضر نمایش می دهد چگونه برای navigation property هایی که در درون entityهای دیگر که آن ها نیز خود در navigation propertyها جای گرفته اند، از eager loading استفاده کنید.
3. زمانی که کاربر یک دوره ی آموزشی را انتخاب می کند، داده های مربوطه از entity set که Enrollments نام دارد، خوانده شده و نشان داده می شود. رابطه ی بین موجودیت های Course و Enrollment از نوع یک به چند می باشد. برای موجودیت های Enrollment و موجودیت های Student مربوطه ی آن قابلیت explicit loading را اعمال می کنیم. (از آنجایی که lazy loading فعال می باشد، نیازی به Explicit loading نیست با این حال جهت آموزش نحوه ی استفاده از آن در این بخش لحاظ شده.)

ایجاد یک View Model برای Instructor Index View

صفحه ی Instructors سه جدول مختلف را به نمایش می گذارد. به همین دلیل یک view model ایجاد می کنیم که هر سه خاصیت را دربرگیرد و هر یک حاوی اطلاعات یک جدول باشد.

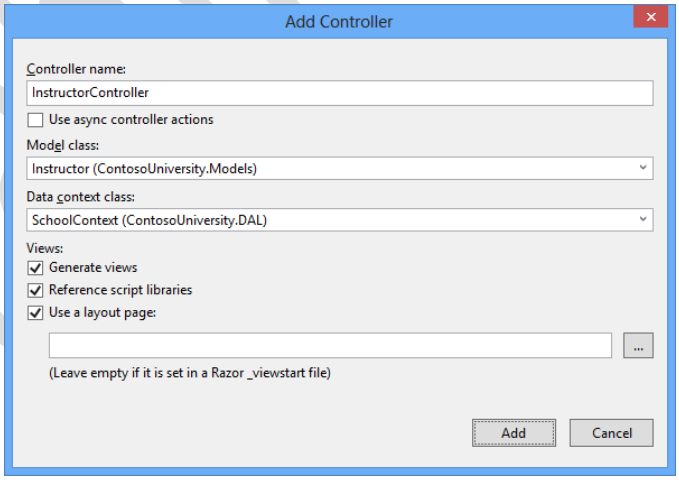
در پوشه ی ViewModels، فایل InstructorIndexData.cs را ایجاد کرده و سپس کد جاری را جایگزین کد زیر کنید:

```
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; }
  }
}
```

ایجاد view ها و Instructor Controller

یک controller به نام InstructorController (نه InstructorsController) با استفاده از عملیات EF read/write ایجاد کنید:



فایل Controllers\InstructorController.cs را باز کرده و یک دستور using به فضای نام (namespace) ViewModels اضافه نمایید:

```
using ContosoUniversity.ViewModels;
```

کدی که توسط قابلیت scaffolding در متد Index ایجاد شده، eager loading را تنها برای خاصیت پیمایشی OfficeAssignment تعریف می کند:

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

کد زیر را جایگزین متد Index کرده تا داده های مربوطه ی اضافی بارگذاری شده، سپس آن را در view model بریزید:

```
        public ActionResult Index(int? id, int? 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.ID == 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);
    }
```

این متد route data اختیاری (id) و یک پارامتر query string (courseID) می پذیرد که مقادیر ID مدرس (instructor) و دوره ی آموزشی (course) انتخابی را فراهم می نماید، سپس تمامی داده های مورد نیاز را به view مورد نظر ارسال می کند. پارامترهای مورد نظر توسط لینک های Select موجود در صفحه ارائه می شوند.

کد با ایجاد یک نمونه از view model و جای گذاری آن در فهرست مدرسان (instructors) آغاز می شود. کد مورد نظر قابلیت eager loading را برای Instructor.OfficeAssignment و خاصیت پیمایشی Instructor.Courses تعریف می کند.

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

دومین متد Include، Courses را بار گذاری می کند، همچنین به ازای هر course که لود می شود، eager loading را برای خاصیت پیمایشی Course.Department اجرا می نماید.

```
.Include(i => i.Courses.Select(c => c.Department))
```

همان طور که پیش تر توضیح دادیم، eager loading الزامی نیست، اما استفاده از آن به بهبود کارایی کمک می کند. از آنجایی که view همیشه به موجودیت OfficeAssignment وابسته است (به آن نیاز دارد)، روش کارآمد این است که آن را در query یکسان واکشی کنید. موجودیت های Course زمانی مورد نیاز می باشند که یک Instructor در صفحه ی وب انتخاب شده باشد، بنابراین eager loading تنها درشرایطی بهتر از lazy loading قلمداد می شود که صفحه ی مورد نظر اغلب با یک course نمایش داده می شود تا بدون آن.

چنانچه instructor ID انتخاب شده باشد، در آن صورت instructor انتخابی از لیست instructor ها در view model بازیابی می شود. خاصیت Courses الحاق شده به view model بعد به همراه موجودیت های Course از خاصیت پیمایشی Courses متعلق به آن instructor بارگذاری می شود.

```
  if (id != null)
{
  ViewBag.InstructorID = id.Value;
  viewModel.Courses = viewModel.Instructors.Where(i => i.ID == id.Value).Single().Courses;
}
```

در اصل متد Where یک مجموعه (collection) بازگردانی می نماید، اما در این مورد خاص معیار یا شرط (criteria) ارسالی به متد مذکور باعث بازگردانی تنها یک موجودیت Instructor می شود. متد Single

مجموعه مورد نظر را به یک موجودیت Instructor واحد تبدیل می کند، که دسترسی به خاصیت Courses آن موجودیت را برای شما فراهم می کند.

زمانی متد Single را بر روی یک مجموعه اعمال می کنیم که می دانیم آن مجموعه دربردانده ی تنها یک آیتم می باشد. چنانچه مجموعه ی ارسالی به متد Single، تهی بوده یا بیش از یک آیتم داشته باشد، متد نام برده یک پیام خطا (exception) صادر می کند. یک روش جایگزین استفاده از SingleOrDefault می باشد که در صورت خالی بودن مجموعه مقدار پیش فرض (null) را بازمی گرداند. اما در این نمونه باز به صدور یک پیام خطا منتهی می شد و پیام خطای صادرشده نیز به روشنی علت رخداد مشکل را شرح نمی داد. به هنگام صدا زدن متد Single، می توان بجای فراخوانی متد Where به صورت جداگانه، شرط Where را به عنوان پارامتر ورودی به متد Single پاس داد. بنویسیم:

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

بجای:

```
.Where(I => i.ID == id.Value).Single()
```

حال زمانی که course انتخاب می شد، course انتخابی از فهرست course ها موجود در view model بازیابی می گردید. سپس خاصیت الحاق شده به view model به همراه موجودیت های Enrollment از خاصیت پیمایشی Enrollments آن course لود می شد.

```
if (courseID != null)
{
  ViewBag.CourseID = courseID.Value;
  viewModel.Enrollments = viewModel.Courses.Where(
    x => x.CourseID == courseID).Single().Enrollments;
}
```

اصلاح Instructor Index View

فایل Views\Instructor\Index.cshtml را باز کرده، کد template را با کد زیر جایگزین کنید. همان طور که می بینید این تغییرات با رنگ زرد نمایش داده شده:

```
@model ContosoUniversity.ViewModels.InstructorIndexData
@{
  ViewBag.Title = "Instructors";
```

```
}
<h2>Instructors</h2>

<p>
   @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
   <tr>
      <th>Last Name</th>
      <th>First Name</th>
      <th>Hire Date</th>
      <th>Office</th>
      <th></th>
   </tr>

   @foreach (var item in Model.Instructors)
   {
      string selectedRow = "";
      if (item.ID == ViewBag.InstructorID)
      {
         selectedRow = "success";
      }
      <tr class="@selectedRow">
         <td>
            @Html.DisplayFor(modelItem => item.LastName)
         </td>
         <td>
            @Html.DisplayFor(modelItem => item.FirstMidName)
         </td>
         <td>
            @Html.DisplayFor(modelItem => item.HireDate)
         </td>
         <td>
            @if (item.OfficeAssignment != null)
            {
               @item.OfficeAssignment.Location
            }
         </td>
         <td>
            @Html.ActionLink("Select", "Index", new { id = item.ID }) |
            @Html.ActionLink("Edit", "Edit", new { id = item.ID }) |
            @Html.ActionLink("Details", "Details", new { id = item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id = item.ID })
         </td>
      </tr>
   }
</table>
```

اصلاحاتی که به کد بالا اعمال شده:

1. Model class به InstructorIndexData تغییر داده شد.
2. عنوان صفحه از Index به Instructors تغییر داده شد.
3. یک ستون به نام Office اضافه شد که تنها در صورتی که item.OfficeAssignment برابر null نباشد، item.OfficeAssignment.Location را نمایش می دهد. (به این خاطر که رابطه ی از نوع یک به صفر یا یک به یک می باشد، ممکن است موجودیت OfficeAssignment مرتبطی وجود نداشته باشد.)

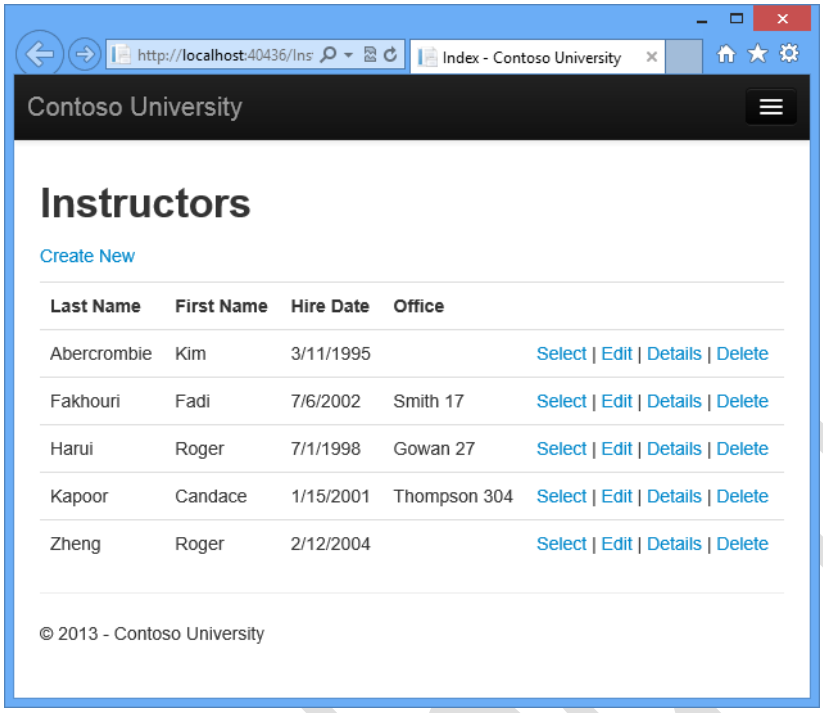
```
<td>
  @if (item.OfficeAssignment != null)
  {
    @item.OfficeAssignment.Location
  }
</td>
```

4. کدی اضافه شده که به صورت داینامیک class="success" را به المان tr، instructor انتخابی ضمیمه می کند. این کد با بهره گیری از کلاس Bootstrap، یک رنگ پس زمینه برای سطر انتخابی تنظیم می کند.

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

5. یک ActionLink که متن آن Select می باشد درست پیش از دیگر لینک ها در هر سطر اضافه شده که باعث می شود instructor ID (شناسه ی مدرس) انتخابی به متد Index ارسال گردد.

برنامه را اجرا کرده و تب ای که Instructors نام دارد را انتخاب کنید. صفحه ی مورد نظر خاصیت Location موجودیت های OfficeAssignment مرتبط و یک خانه ی جدول تهی در صورتی که موجودیت OfficeAssignment وجود ندارد (دفتری به مدرس تخصیص داده نشده) نمایش می دهد.



در فایل Views\Instructor\Index.cshtml، پس از المان بسته ی table (در انتهای فایل)، کد زیر را اضافه می کنیم. با انتخاب یک instructor، این کد یک فهرست از course های مربوط به آن instructor را به نمایش می گذارد.

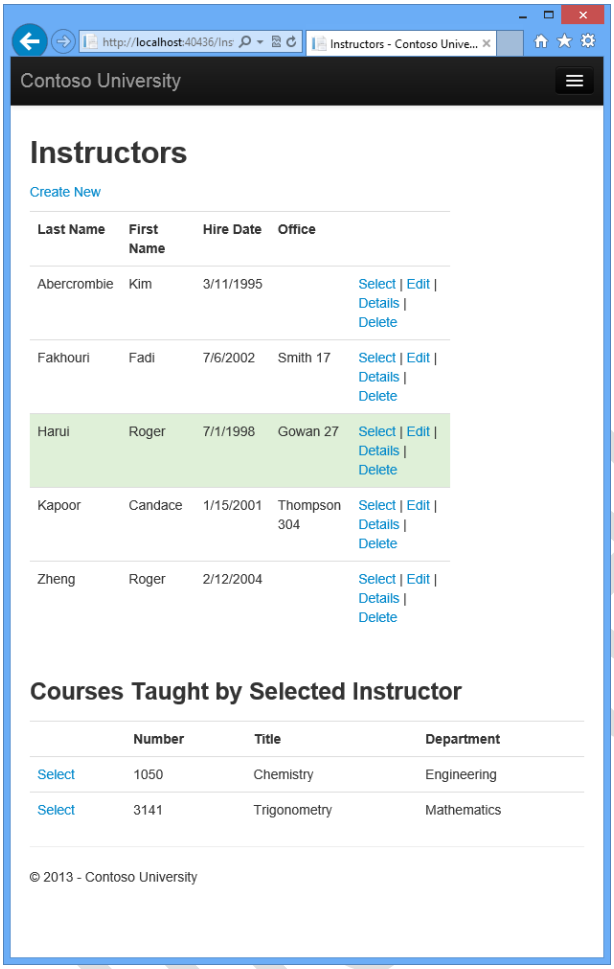
```
@if (Model.Courses != null)
{
  <h3>Courses Taught by Selected Instructor</h3>
  <table class="table">
    <tr>
      <th></th>
      <th>Number</th>
      <th>Title</th>
      <th>Department</th>
    </tr>

    @foreach (var item in Model.Courses)
    {
      string selectedRow = "";
      if (item.CourseID == ViewBag.CourseID)
      {
        selectedRow = "success";
      }
      <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>
}
```

کد حاضر با خواندن خاصیت Courses از view model، فهرستی از course ها را نمایش می دهد. این کد همچنین یک لینک که متن آن Select می باشد را ارائه می دهد که ID یا شناسه ی دوره ی انتخاب شده (Course) را به اکشن متد Index می فرستد.

صفحه را اجرا کرده و یک instructor را انتخاب کنید. یک grid ظاهر می شود که دوره های آموزشی تخصیص یافته به مدرس انتخابی را نمایش می دهد، همچنین اسم دپارتمان هر دوره یا course را مشاهده می کنید.



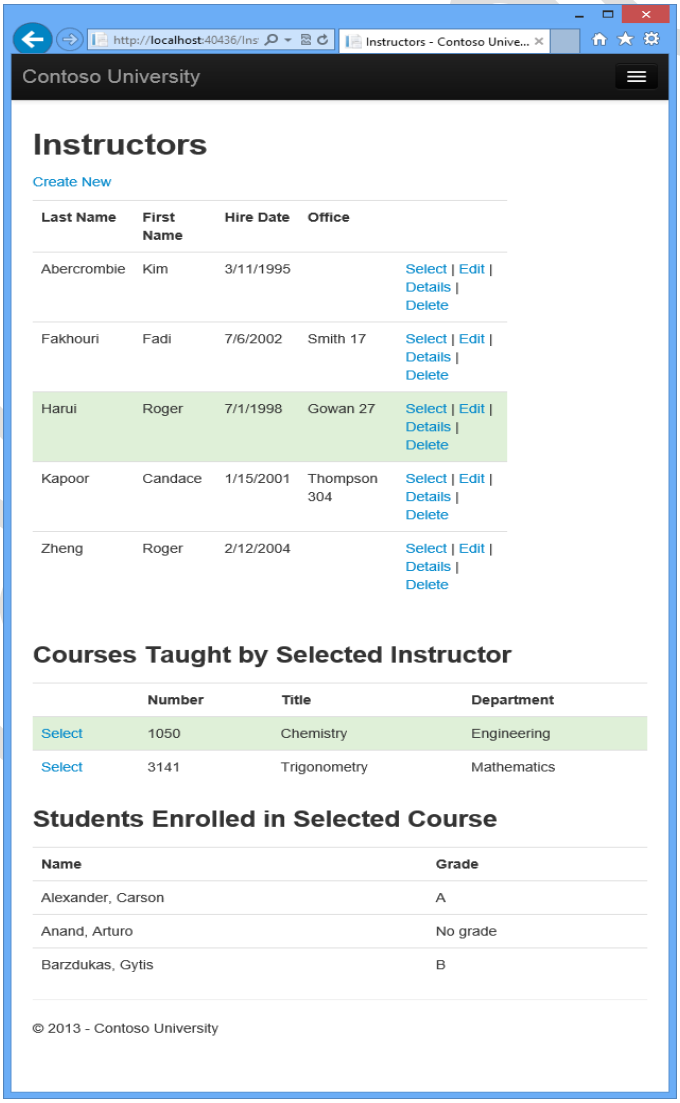
پس از قطعه کدی که هم اکنون اضافه کردید، کد زیر را اضافه کنید. این کد یک لیست از دانشجویان را نمایش می دهد که با انتخاب آن course در دوره ی آموزشی ثبت نام می شوند.

```
@if (Model.Enrollments != null)
{
  <h3>
    Students Enrolled in Selected Course
  </h3>
  <table class="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>
}
```

این کد خاصیت Enrollments از view model را خوانده تا بتواند فهرستی از دانشجویانی که در آن ثبت نام کرده اند را نمایش دهد.

صفحه را اجرا کرده و یکی از مدرسان را انتخاب کنید. سپس یکی از دوره های آموزشی را انتخاب کنید و فهرست دانشجویانی که ثبت نام کرده اند را به ضمیمه ی نمره ی آن ها مشاهده نمایید.



افزودن قابلیت Explicit Loading

فایل InstructorController.cs را باز کرده و ببینید چگونه متد Index فهرست ثبت نام های (enrollment list) یک دوره ی آموزشی انتخابی (course) را بازیابی می کند.

```
        if (courseID != null)
    {
      ViewBag.CourseID = courseID.Value;
      viewModel.Enrollments = viewModel.Courses.Where(
        x => x.CourseID == courseID).Single().Enrollments;
    }
```

پس از بازیابی فهرست مدرسان، eager loading را برای خاصیت پیمایشی Courses و خاصیت Department هر Course مشخص کردید. سپس مجموعه ی Courses را در view model جای گذاری کرده، شما در حال حاضر به خاصیت پیمایشی Enrollments از یک entity در آن مجموعه دسترسی دارید. به این خاطر که شما eager loading را برای خاصیت پیمایشی Course.Enrollments مشخص نکردید، داده های آن خاصیت درنتیجه ی تعریف Lazy loading در صفحه ی مورد نظر ظاهر می شود.

چنانچه lazy loading را بدون ایجاد تغییر در کد غیرفعال می کردید، خاصیت Enrollments صرف نظر از اینکه آن Course چه تعداد enrollment داشت، null می بود. در چنین موقعیتی، جهت بارگذاری خاصیت Enrollments، می بایست lazy loading یا explicit loading را اعمال می کردید. پیش تر با نحوه ی اجرای eager loading آشنا شدید. به منظور مشاهده ی مثالی از explicit loading، بایستی متد Index را با کد زیر جایگزین کنید. در این نوع بارگذاری، خاصیت Enrollments به صورت صریح بارگذاری می شود. کد تغییر یافته با رنگ زرد هایلایت شده است.

```
public ActionResult Index(int? id, int? 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.ID == id.Value).Single().Courses;
    }

    if (courseID != null)
    {
        ViewBag.CourseID = courseID.Value;
        // Lazy loading
        //viewModel.Enrollments = viewModel.Courses.Where(
        //    x => x.CourseID == courseID).Single().Enrollments;
        // Explicit loading
        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);
}
```

پس از بازیابی موجودیت Course انتخابی خواهید دید که کد جدید، خاصیت پیمایشی Enrollments آن course را به صورت صریح بارگذاری می کند.

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

سپس موجودیت Student مربوط به هر Enrollments را به صورت صریح بارگذاری می کند:

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

همان طور که مشاهده می کنید متد Collection را برای بارگذاری collection property بکار بردیم، اما برای خاصیتی که تنها یک enrollment دارد، متد Reference را مورد استفاده قرار می دهیم.

اکنون صفحه ی Instructor را اجرا کنید. خواهید دید که صفحه بدون هیچ گونه تغییری در محتوا نمایش داده می شود، اگرچه نحوه ی بازیابی اطلاعات تغییر پیدا کرده است.

در این مبحث از هر سه روش بارگذاری برای لود داده های مورد نظر در navigation property ها استفاده شد. در مبحث بعدی با چگونگی بروز آوری داده های مربوط آشنا خواهید شد.