بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

مدیریت سناریوهای پیچیده ی EF6 در برنامه های MVC

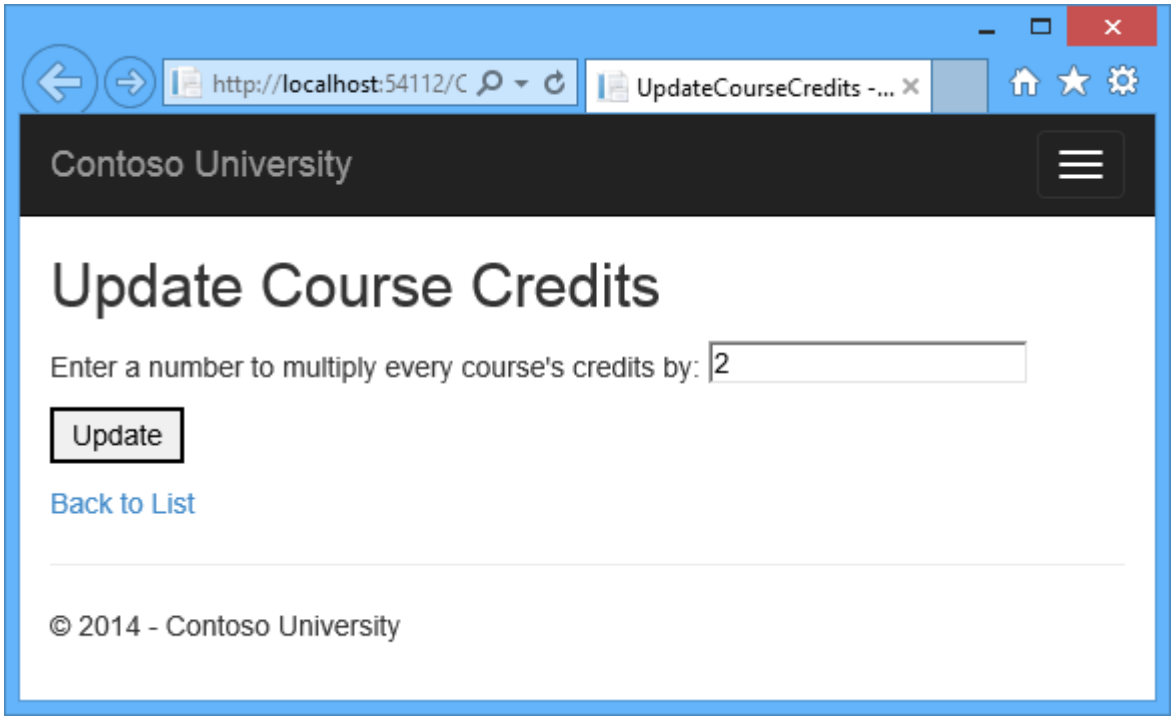
مدرس : مهندس افشین رفوآ

دوره آموزش MVC

مدیریت سناریوهای پیچیده ی EF6 در برنامه های MVC

این آموزش شامل نکاتی بسیار زیادی می باشد که در سناریوهای پیچیده توسعه ی برنامه های تحت وب MVC که از EF Code First بهره می گیرند، بکار شما می آید.

در بیشتر بخش ها از صفحاتی که در فصل های قبلی ایجاد کردیم، استفاده خواهیم کرد. جهت استفاده از دستورهای خام SQL برای اجرای عملیات بروز رسانی دسته جمعی (bulk)، می بایست یک صفحه ی جدید ایجاد کرده که تعداد credits (واحدهای پاس شده) تمام course های موجود در پایگاه داده را بروز آوری کند:



اجرای query های SQL در زبان LINQ (raw query)

رابط برنامه سازی کاربردی (API) EF Code First متدهایی را ارائه می دهد که به شما امکان می دهد دستورات SQL را مستقیم به پایگاه داده ی مورد نظر ارسال کنید. می توانید از روش های زیر استفاده کنید:

1. از متد DbSet.SqlQuery برای query هایی که نوع entity ها را برمی گرداند، استفاده کنید. اشیا بازگشتی بایستی از همان نوعی باشد که مورد انتظار شی DbSet است. این اشیا به صورت خودکار توسط database context ردیابی (track) می شوند، مگر این قابلیت tracking را خود غیر فعال کنید.

2. از متد Database.SqlQuery برای query هایی که نوع خروجی آن ها entity نیست استفاده کنید. داده های بازگشتی توسط database context ردیابی نمی شوند، حتی اگر از این متد برای بازیابی نوع entity استفاده کنید.

3. از Database.ExecuteSqlCommand برای دستورهایی که query نیستند، استفاده کنید.

یکی از مزایای استفاده از EF این است که کد شما را خیلی به یک روش ذخیره ی داده مشخص وابسته نمی کند. EF این کار را با ایجاد query ها و دستورات SQL برای شما انجام می دهد که یک مزیت جانبی آن رهایی

شما از نوشتن دستورات و query ها می باشد. اما گاهی شرایطی (استثنایی) پیش می آید که در آن شما مجبور به استفاده از query هایی هستید که خود به صورت دستی نوشته اید و این روش ها به شما کمک می کنند، چنین شرایطی را مدیریت کنید.

توجه داشته باشید که همیشه باید به هنگام اجرای دستورات SQL، اقدامات پیشگیرانه ای را جهت ایمن سازی سایت در برابر حملات SQL injection اتخاذ کنید. یکی از روش هایی که می توانید این کار را انجام دهید، استفاده از query های پارامتر دار است، به عبارت دیگر به واسطه ی query های با پارامتر کاری می کنیم که رشته های ارسال شده توسط صفحه ی وب به عنوان دستورات SQL تفسیر نشوند. در آموزش حاضر برای گنجاندن (یکپارچه سازی) ورودی کاربر در یک query، از query های باپارامتر استفاده می کنیم.

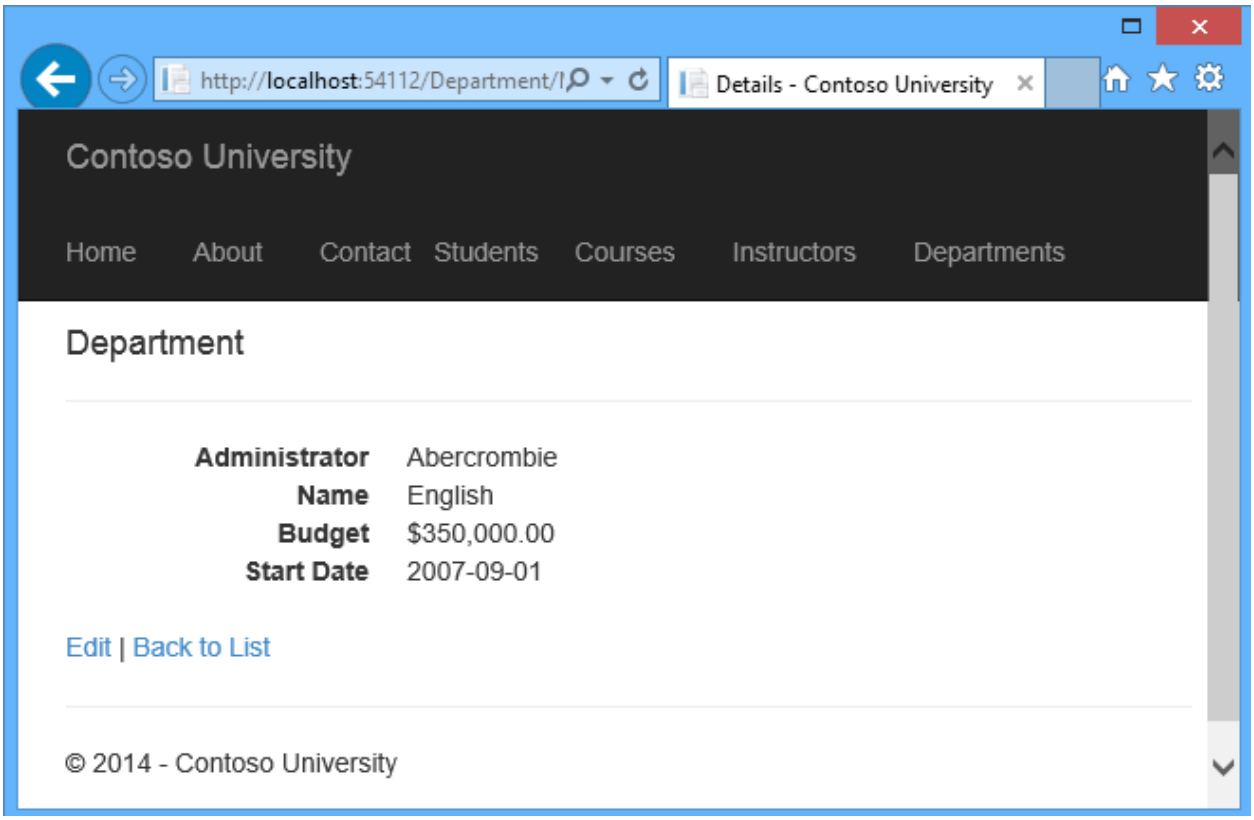
فراخوانی query ای که خروجی آن Entity می باشد

کلاس DbSet<TEntity> متدی ارائه می کند که با استفاده از آن می توانید query را اجرا کنید که خروجی آن موجودیتی از نوع TEntity می باشد. برای آشنایی با نحوه ی کارکرد آن، کد متد Details را در کنترلر Department تغییر می دهیم.

در فایل DepartmentController.cs، در متد Details، فراخوانی متد db.Departments.FindAsync را با فراخوانی متد db.Departments.SqlQuery جایگزین کنید، همان طور که در کد رنگی شده ی زیر نمایش داده شده است:

```
public async Task<ActionResult> Details(int? id)
    {
      if (id == null)
      {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
      }
      // Create and execute raw SQL query.
      string query = "SELECT * FROM Department WHERE DepartmentID = @p0";
      Department department = await db.Departments.SqlQuery(query, id).SingleOrDefaultAsync();

      if (department == null)
      {
        return HttpNotFound();
      }
      return View(department);
    }
```

جهت بررسی و کسب اطمینان از عملکرد صحیح کد، تب Departments و سپس صفحه ی Details مربوط به یکی از department ها را انتخاب کرده و مشاهده کنید.



فراخوانی query ای که اشیا از نوع دیگر را به عنوان خروجی برمی گرداند

پیشتر یک grid دربردارنده ی اطلاعات مربوط به آمار دانشجویان (student statistics) برای صفحه ی About ایجاد کردید. این صفحه تعداد دانشجویانی را که در هر تاریخ (enrollment date) ثبت نام کردند را نشان می دهد. کدی که این کار را در فایل HomeController.cs انجام می دهد، از زبان LINQ استفاده می کند:

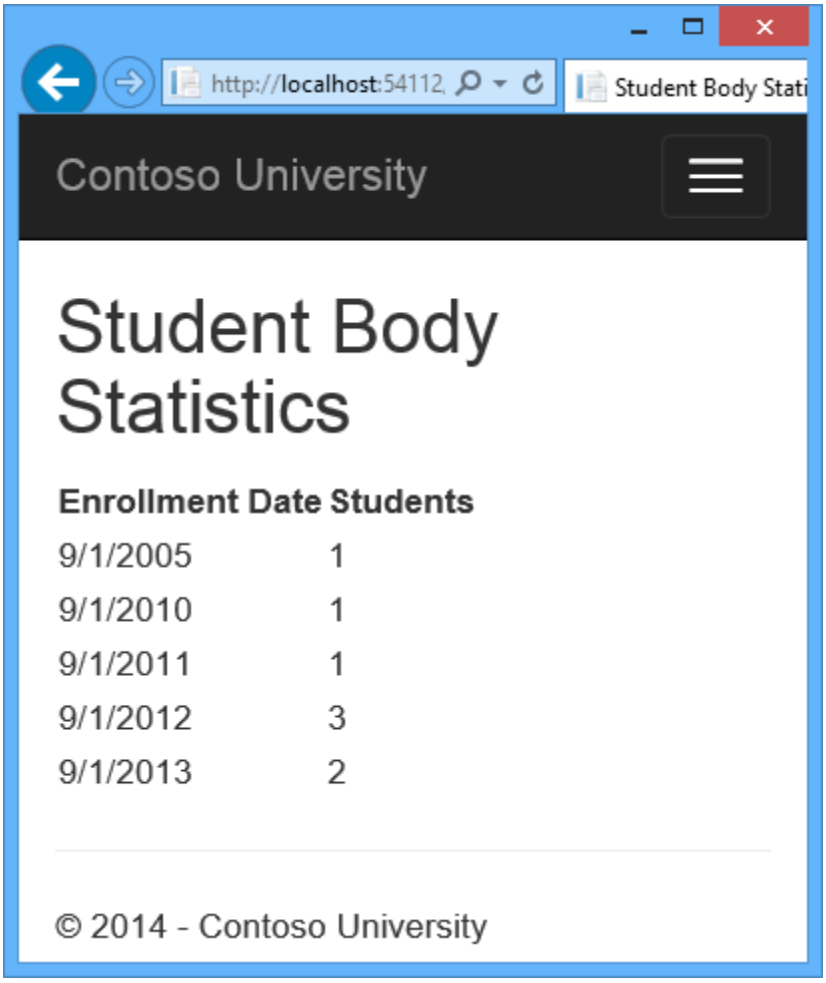
```
var data = from student in db.Students
           group student by student.EnrollmentDate into dateGroup
           select new EnrollmentDateGroup()
           {
             EnrollmentDate = dateGroup.Key,
             StudentCount = dateGroup.Count()
           };
```

فرض کنید کدی می خواهیم بنویسیم که داده ها را مستقیما از خود SQL و بدون استفاده از LINQ بازیابی می کند. برای این منظور می بایست کدی بنویسید که خروجی آن entity object نباشد، متد Database.SqlQuery نیز دقیقا همین کار را انجام می دهد.

در فایل HomeController.cs، دستور LINQ را در متد About با یک دستور SQL جایگزین کنید:

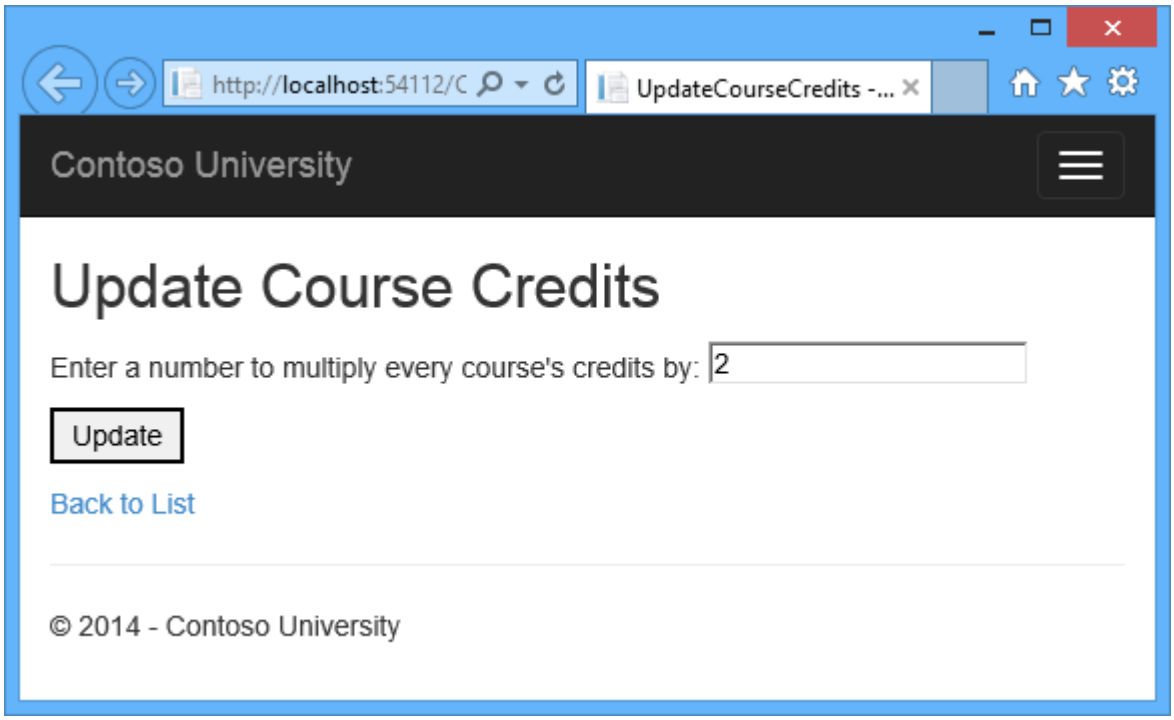
```
public ActionResult About()
 {
   // Commenting out LINQ to show how to do the same thing in SQL.
   //IQueryable<EnrollmentDateGroup> = from student in db.Students
   //        group student by student.EnrollmentDate into dateGroup
   //        select new EnrollmentDateGroup()
   //        {
   //          EnrollmentDate = dateGroup.Key,
   //          StudentCount = dateGroup.Count()
   //        };

   // SQL version of the above LINQ code.
   string query = "SELECT EnrollmentDate, COUNT(*) AS StudentCount "
     + "FROM Person "
     + "WHERE Discriminator = 'Student' "
     + "GROUP BY EnrollmentDate";
   IEnumerable<EnrollmentDateGroup> data = db.Database.SqlQuery<EnrollmentDateGroup>(query);

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

صفحه ی About را اجرا کنید. خواهید دید که همان اطلاعاتی که قبلا نمایش می داد را در حال حاضر نیز نمایش می دهد.



فراخوانی Update Query

فرض کنید administrator ها (مدیران پایگاه داده) Contoso University می خواهند تغییرات دسته جمعی (bulk) همچون تعداد credit ها یا واحدهای پاس شده برای هر دوره ی آموزشی یا course، را در دیتابیس مورد نظر اعمال کنند. چنانچه تعداد course ها در دانشگاه بیش از حد معمول بود، در آن صورت استخراج و بازیابی آن ها به عنوان entity و ویرایش هر یک به صورت جدا، ناکارامد خواهد بود. در این بخش یک صفحه ی وب پیاده سازی می کنیم که به کاربر اجازه می دهد یک ضریب مشخص کند که تعداد credit های تمامی course ها در آن ضرب شده و بر آن اساس تغییر یابد. این تغییر را با اجرای یک دستور UPDATE اعمال می کنیم. صفحه ی وب مورد نظر بدین شکل خواهد بود:

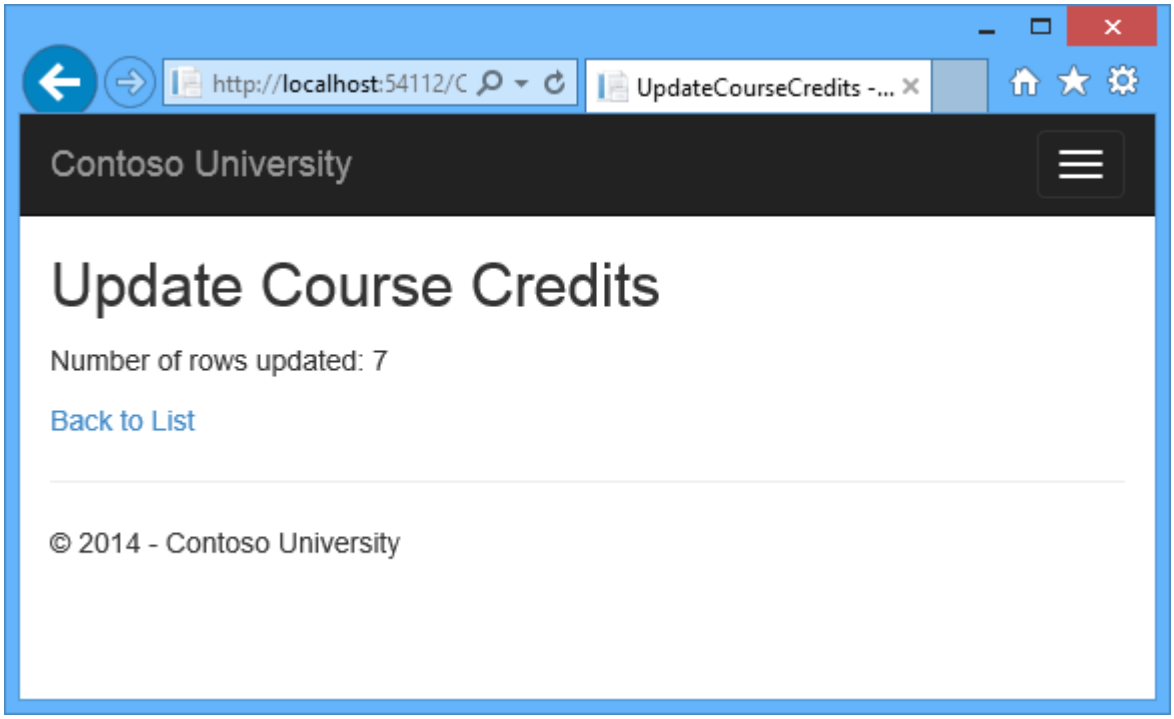


فایل CourseContoller.cs را باز کرده و دو متد UpdateCourseCredits را برای خصیصه های HttpGet و HttpPost اضافه می کنیم:

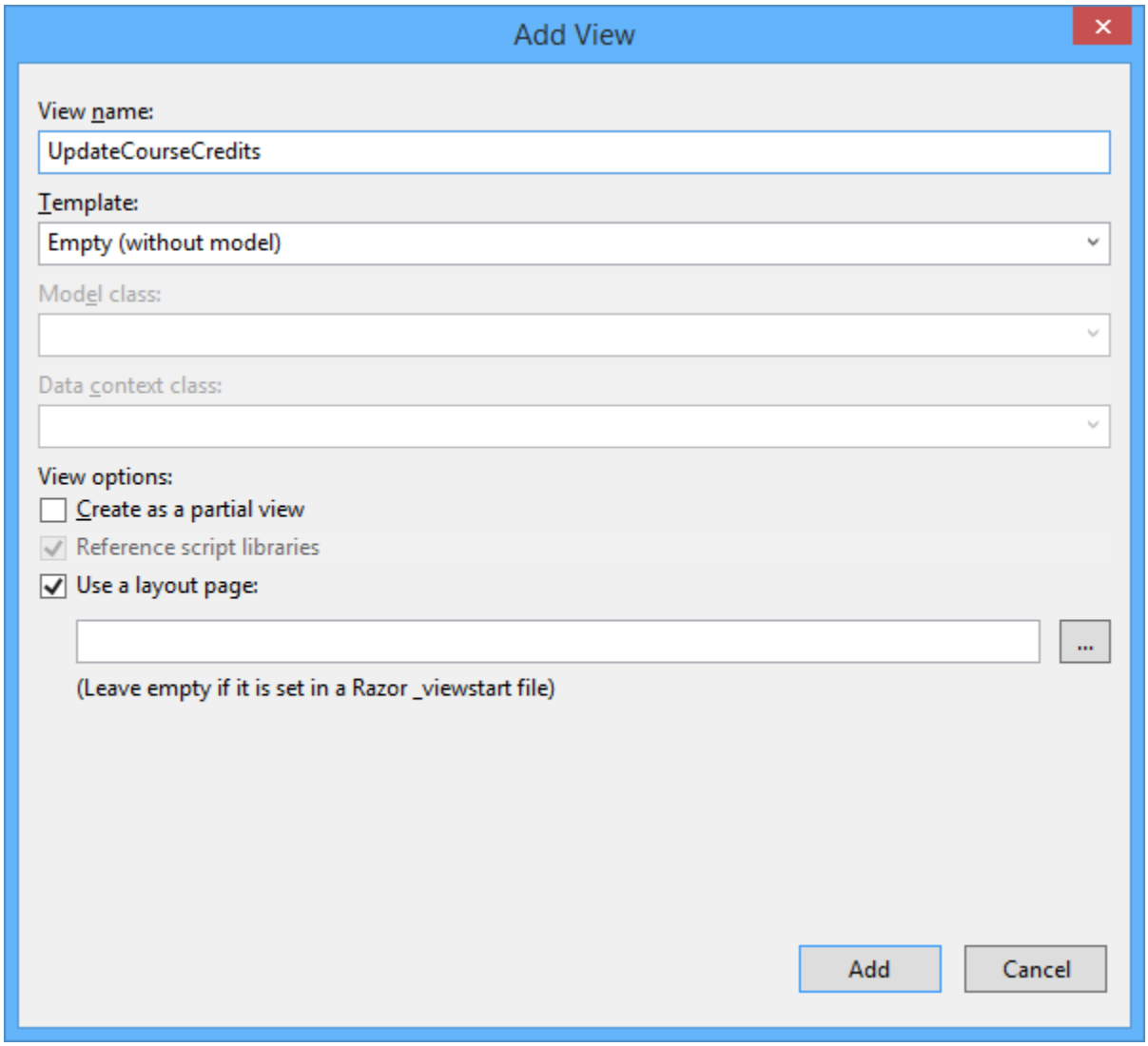
```
public ActionResult UpdateCourseCredits()
{
  return View();
}
[HttpPost]
public ActionResult UpdateCourseCredits(int? multiplier)
{
  if (multiplier != null)
  {
    ViewBag.RowsAffected = db.Database.ExecuteSqlCommand("UPDATE Course SET Credits = Credits * {0}",
multiplier);
  }
  return View();
}
```

هنگامی که controller درخواست HttpGet را پردازش می کند، هیچ مقداری در متغیر ViewBag.RowsAffected بازگردانده و ثبت نمی شود و همان طور که در تصویر قبلی مشاهده شد، view مورد نظر یک textbox تهی به ضمیمه ی دکمه ی ارسال (submit) نمایش می دهد.

پس از این دکمه ی Update کلیک می شود، متد HttpPost صدا زده شده و multiplier مقدار را در textbox درج می کند. کد مورد نظر سپس دستور SQL که course ها رابروز رسانی می کند، اجرا کرده و تعداد سطرهای اصلاح شده را به view در متغیر ViewBag.RowsAffected بازمی گرداند. هنگامی که view مقداری را از آن متغیر دریافت می کند، بجای آن textbox و دکمه ی submit، تعداد سطرهای بروز رسانی شده را نمایش می دهد:



در CourseController.cs، یکی از متدهای UpdateCourseCredits را راست کلیک کرده، سپس Add View را کلیک کنید:

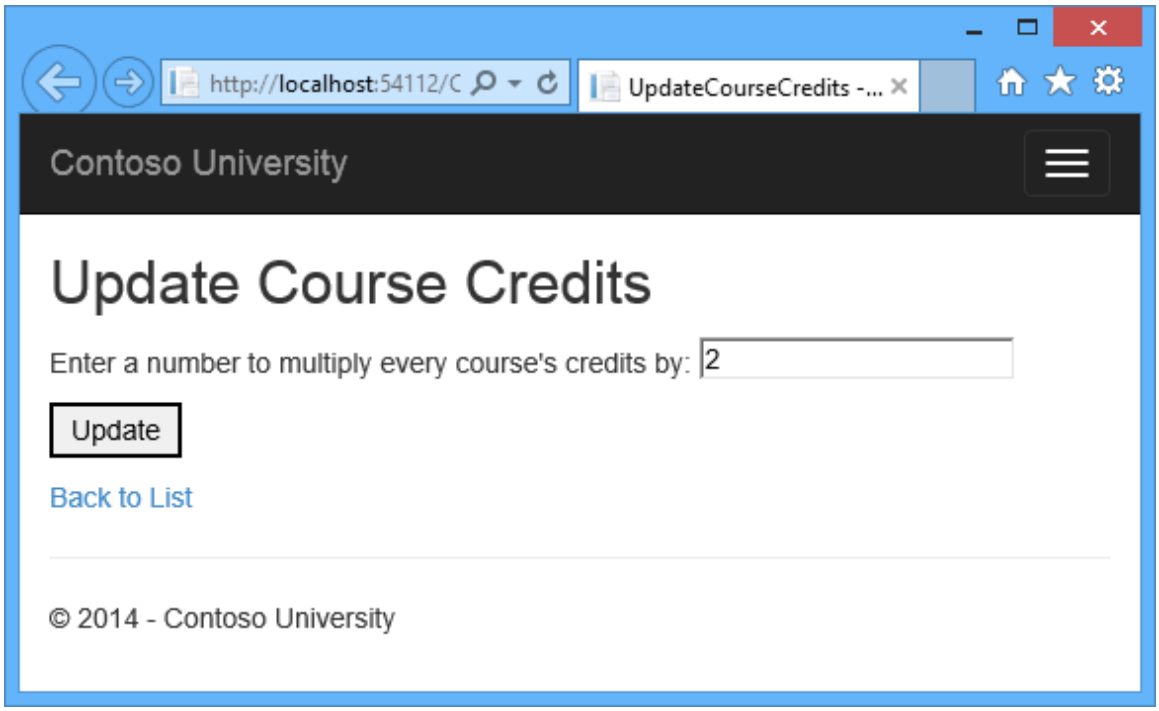


در فایل Views\Course\UpdateCourseCredits.cshtml، کد template را با کد زیر جایگزین کنید:

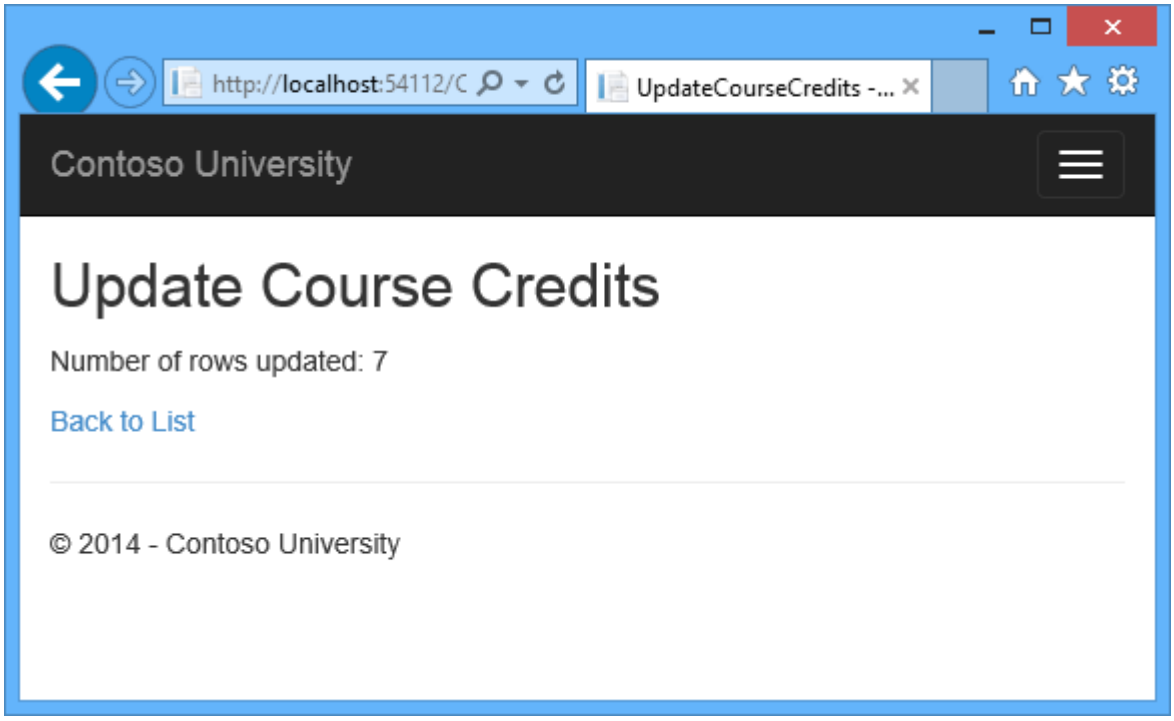
```
@model ContosoUniversity.Models.Course
@{
  ViewBag.Title = "UpdateCourseCredits";
}
<h2>Update Course Credits</h2>
@if (ViewBag.RowsAffected == null)
{
  using (Html.BeginForm())
  {
    <p>
      Enter a number to multiply every course's credits by: @Html.TextBox("multiplier")
    </p>
    <p>
      <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>
```

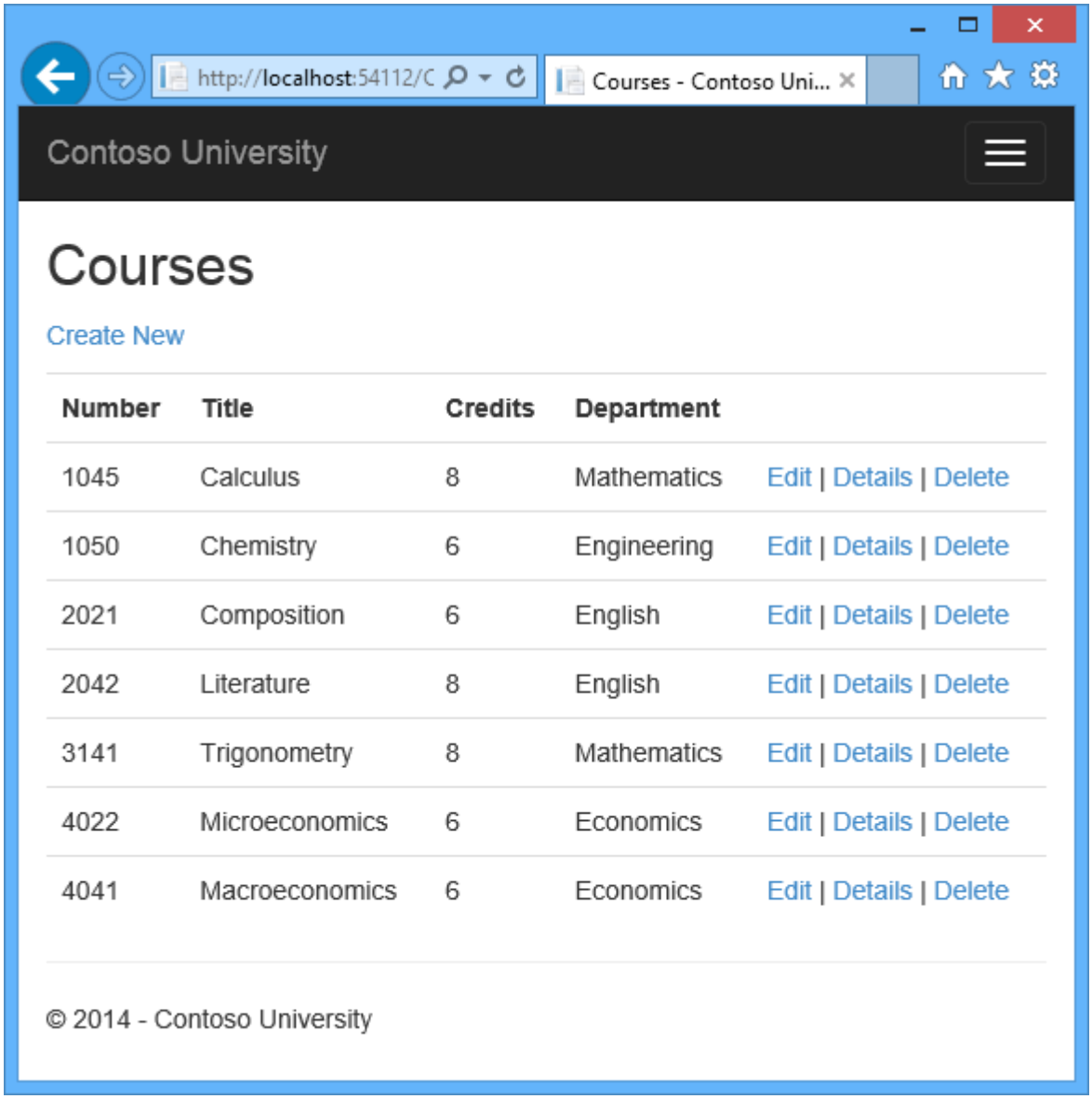
با انتخاب تب Courses و افزودن "/UpdateCourseCredits" به انتهای URL در نوار آدرس مرورگر، متد UpdateCourseCredits را اجرا کنید (مثال: http://localhost:50205/Course/UpdateCourseCredits). یک عدد داخل کادر وارد کنید:



دکمه ی Update را کلیک کنید. تعداد سطرهای بروز رسانی شده را مشاهده خواهید کرد:



روی لینک Back to List کلیک کرده تا فهرست course ها را به همراه تعداد credit های اصلاح شده هر course مشاهده کنید:



No-Tracking Queries

هنگامی که database context سطرهای جدول را بازیابی کرده و entity object هایی را برای نمایش آن ها ایجاد می کند، به صورت پیش فرض بررسی می کند آیا موجودیت های مقیم در حافظه با آنچه در پایگاه داده وجود دارد همگام هست یا خیر. داده های موجود در حافظه به عنوان یک cache عمل کرده و زمانی که یک موجودیت را بروز رسانی می کنید، مورد استفاده قرار می گیرد. به این خاطر که context instance ها معمولا موقتی هستند بدین معنا که به ازای هر request یکی ایجاد شده و دور انداخته می شود و همچنین context

ای که یک موجودیت را می خواند اغلب پیش از اینکه آن موجودیت بار دیگر استفاده شود، دور انداخته (dispose) می شود، استفاده از caching در یک برنامه ی تحت وب غیر ضروری می باشد.

می توانید tracking (ردیابی) entity object ها در حافظه را با استفاده از متد AsNoTracking غیر فعال کنید. سناریوهایی که اغلب در آن ها tracking را غیرفعال می کنید، در زیر شرح داده شده اند:

1. query چنان حجم سنگینی از داده ها را بازیابی می کند که غیر فعال کردن tracking، افزایش کارایی بسزایی را به دنبال دارد.

2. می خواهید یک موجودیت به متد AsNoTracking پیوست کنید که آن موجودیت را آپدیت کند، اما قبلا آن موجودیت را برای منظوری دیگری بازیابی کرده اید. چون که آن موجودیت از قبل توسط database context track (ردیابی) می شود، نمی توانید موجودیتی که می خواهید ویرایش کنید را پیوست نمایید. یکی از روش های مدیریت این سناریو استفاده از گزینه ی AsNoTracking با query مورد نظر می باشد.

در این آموزش به خاطر اینکه flag ویرایش شده بر روی موجودیت ایجاد شده توسط model-binder در متد Edit تنظیم نشده، به متد AsNoTracking نیز نیازی نیست.

بررسی query های ارسالی به پایگاه داده

گاهی اوقات دیدن خود query ارسالی به پایگاه داده به شما کمک شایانی می کند. در آموزش های قبلی نحوه ی انجام آن را در کد interceptor آموختید، حال روشی را برای انجام این کار بدون نوشتن کد interceptor خواهید دید. برای تست این روش، ابتدا یک query ساده را مورد بررسی قرار می دهیم، سپس گزینه ها و قابلیت های مختلفی همچون eager loading، filtering و sorting به آن اضافه می کنیم و رفتار آن را بار دیگر آزمایش می کنیم.

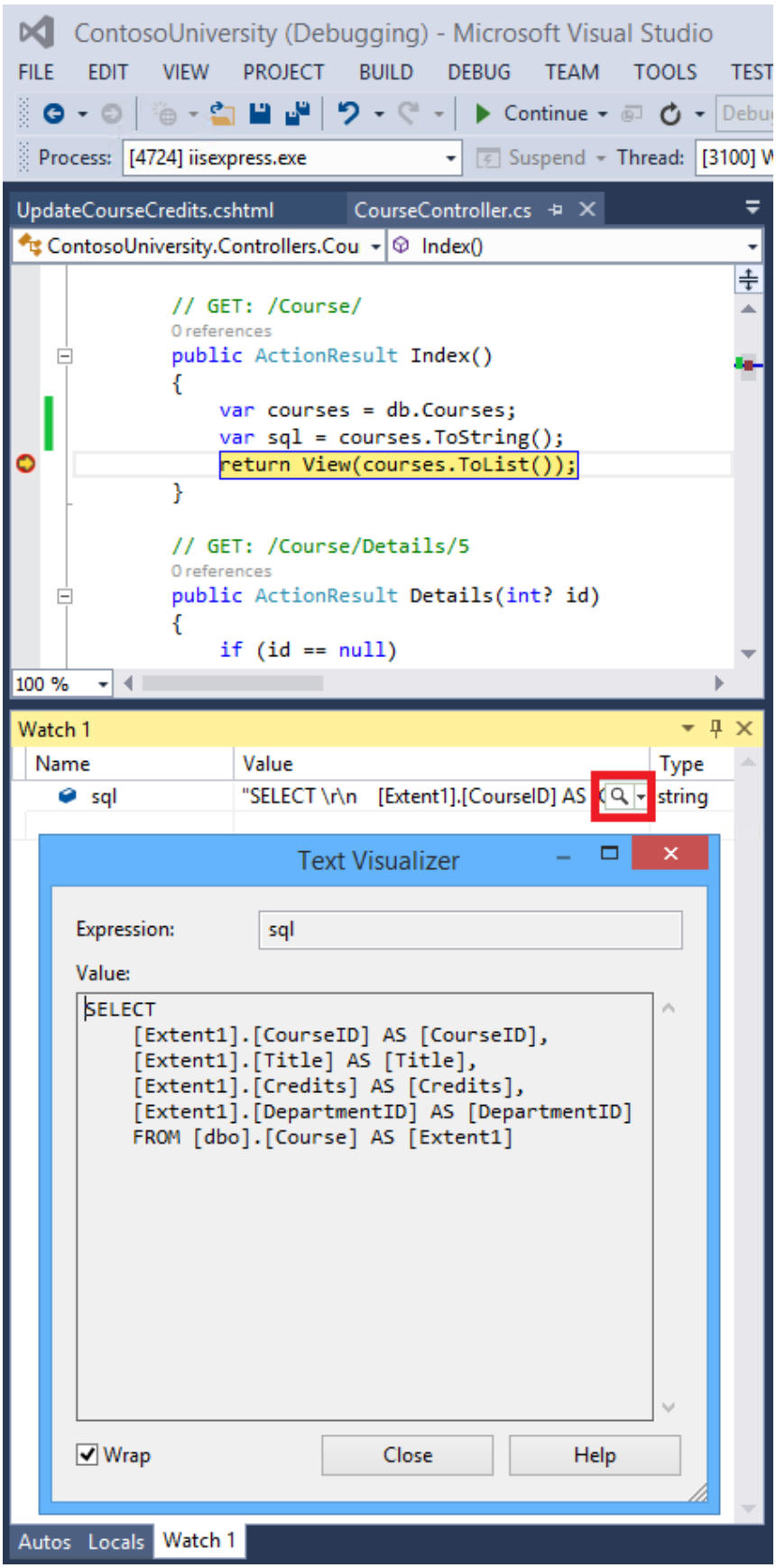
در فایل Controllers/CourseController، متد Index را با کد زیر جایگزین نمایید. این کار باعث می شود eager loading به طور موقت غیرفعال شود:

```
public ActionResult Index()
{
  var courses = db.Courses;
  var sql = courses.ToString();
  return View(courses.ToList());
}
```

حال یک breakpoint (نقطه ی انفصال) بر روی خط دربردارنده ی دستور return اعمال کنید (برای این منظور اشاره گر موس را روی خطر مورد نظر قرار داده و دکمه ی F9 را فشار دهید). کلید F5 را زده تا پروژه در حالت خطایابی (Debug mode) اجرا شود، سپس صفحه ی Course Index را انتخاب کنید. با رسیدن کد به breakpoint، متغیر sql را مورد بررسی قرار دهید. در اینجا query ارسالی به SQL Server را خواهید دید؛ یک دستور ساده ی Select پرس و جوی ارسالی به پایگاه داده می باشد.

```
{SELECT
[Extent1].[CourseID] AS [CourseID],
[Extent1].[Title] AS [Title],
[Extent1].[Credits] AS [Credits],
[Extent1].[DepartmentID] AS [DepartmentID]
FROM [Course] AS [Extent1]}
```

بر روی کلاس magnifier (آیکون ذره بین) کلیک کرده تا بتوانید query مورد نظر را در پنجره ی Text Visualizer مشاهده کنید.


ContosoUniversity (Debugging) - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST
Continue
Process: [4724] iisexpress.exe
Suspend
Thread:
UpdateCourseCredits.cshtml
CourseController.cs
ContosoUniversity.Controllers.Cou
Index()
// GET: /Course/
0 references
public ActionResult Index()
{
var courses = db.Courses;
var sql = courses.ToString();
return View(courses.ToList());
}
// GET: /Course/Details/5
0 references
public ActionResult Details(int? id)
{
if (id == null)
100 %
Watch 1
Name
Value
Type
sql
"SELECT \r\n [Extent1].[CourseID] AS
string
Text Visualizer
Expression:
sql
Value:
SELECT
[Extent1].[CourseID] AS [CourseID],
[Extent1].[Title] AS [Title],
[Extent1].[Credits] AS [Credits],
[Extent1].[DepartmentID] AS [DepartmentID]
FROM [dbo].[Course] AS [Extent1]
Wrap
Close
Help
Autos
Locals
Watch 1

اکنون یک drop-down list (فهرست کشویی) به صفحه ی Courses Index اضافه خواهیم کرد تا کاربران بتوانند با استفاده از قابلیت فیلترینگ، department مورد نظر را جستجو کنند. Course ها را بر اساس عنوان (title) آن ها مرتب سازی خواهیم کرد، همچنین eager loading را برای navigation property (خاصیت پیمایشی) Department مشخص می کنیم:

در فایل CourseController.cs، کد زیر را با متد Index جایگزین نمایید:

```
    public ActionResult Index(int? SelectedDepartment)
{
  var departments = db.Departments.OrderBy(q => q.Name).ToList();
  ViewBag.SelectedDepartment = new SelectList(departments, "DepartmentID", "Name", SelectedDepartment);
  int departmentID = SelectedDepartment.GetValueOrDefault();

  IQueryable<Course> courses = db.Courses
    .Where(c => !SelectedDepartment.HasValue || c.DepartmentID == departmentID)
    .OrderBy(d => d.CourseID)
    .Include(d => d.Department);
  var sql = courses.ToString();
  return View(courses.ToList());
}
```

Breakpoint را بار دیگر روی خط حاوی دستور return اعمال کنید.

متد مورد نظر مقدار انتخابی drop-down list را در پارامتر SelectedDepartment دریافت می کند. اگر هیچ مقداری انتخاب نشده باشد، این پارامتر null خواهد بود.

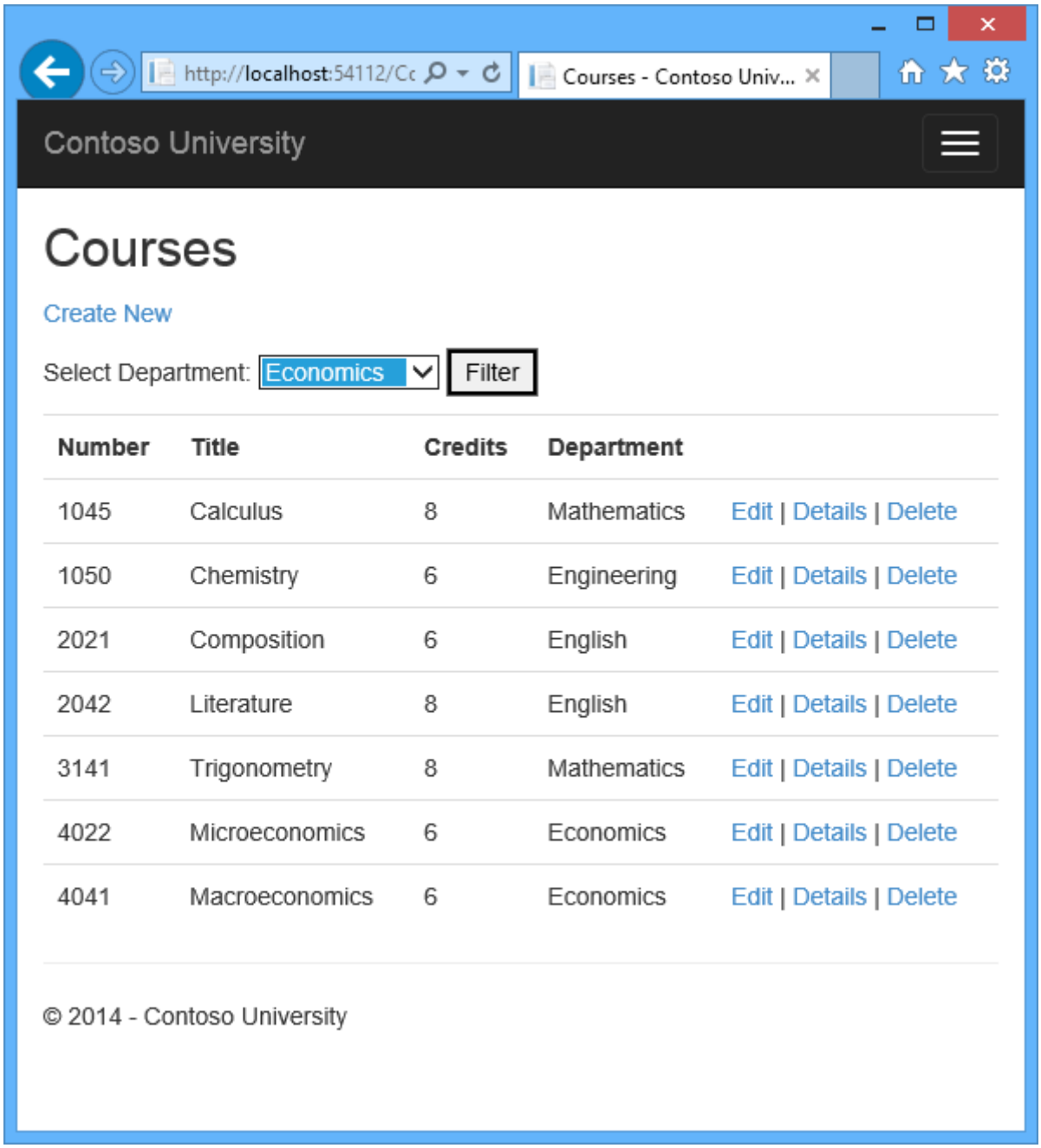
مجموعه (collection) SelectList حاوی تمامی department ها برای نمایش در لیست کشویی به view ارسال می شود. پارامترهای ارسالی به constructor (سازنده) SelectList اسم فیلد مقدار، اسم فیلد متن و آیتم انتخاب شده را مشخص می کند.

کد حاضر برای متد Get مجموعه (repository) Course، یک عبارت فیلتر (filter expression)، یک ترتیب مرتب سازی (sort order) تعریف کرده، همچنین برای خاصیت پیمایشی (navigation property) Department روش بارگذاری eager loading را مشخص می کند.

در فایل Views\Course\Index.cshtml، درست پیش از تگ باز table، کد زیر را به منظور ایجاد drop-down list و دکمه ی ارسال (submit) درج نمایید:

```
@using (Html.BeginForm())
{
  <p>Select Department: @Html.DropDownList("SelectedDepartment","All")
  <input type="submit" value="Filter" /></p>
}
```

توجه داشته باشید که breakpoint را بایستی از قبل بر روی خط مورد نظر اعمال کرده باشید. حال صفحه ی Course Index را اجرا کنید. اجرا را برای اولین بار ادامه دهید تا کد به breakpoint برسد و صفحه در مرورگر نمایش داده شود. یک department از drop-down list انتخاب کرده و دکمه ی Filter را کلیک نمایید:



این بار اولین breakpoint برای کوئری department ای که drop-down list را پر می کند، می باشد. این breakpoint را رد کرده و دفعه ی بعدی که کد مورد نظر به breakpoint می رسد، متغیر query را در نظر داشته تا ببینید، کوئری Course اکنون چگونه بنظر می رسد. کد زیر را مشاهده خواهید کرد:

```
SELECT
  [Project1].[CourseID] AS [CourseID],
  [Project1].[Title] AS [Title],
  [Project1].[Credits] AS [Credits],
```

```
[Project1].[DepartmentID] AS [DepartmentID],
[Project1].[DepartmentID1] AS [DepartmentID1],
[Project1].[Name] AS [Name],
[Project1].[Budget] AS [Budget],
[Project1].[StartDate] AS [StartDate],
[Project1].[InstructorID] AS [InstructorID],
[Project1].[RowVersion] AS [RowVersion]
FROM ( 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].[InstructorID] AS [InstructorID],
  [Extent2].[RowVersion] AS [RowVersion]
  FROM  [dbo].[Course] AS [Extent1]
  INNER JOIN [dbo].[Department] AS [Extent2] ON [Extent1].[DepartmentID] = [Extent2].[DepartmentID]
  WHERE @p__linq__0 IS NULL OR [Extent1].[DepartmentID] = @p__linq__1
) AS [Project1]
ORDER BY [Project1].[CourseID] ASC
```

همان طور که مشاهده می کنید، query مورد نظر اکنون از نوع JOIN می باشد که داده های Department را به همراه داده های Course بارگذاری کرده و حاوی یک عبارت WHERE می باشد.

خط var sql = courses.ToString() را حذف کنید.

الگوهایی برای تعریف واحد کاری (unit of work) و مجموعه ها (repository)

بسیاری از برنامه نویسان الگوهای واحد کاری و repository هایی را به عنوان دربرگیرنده هایی (wrapper) برای کدهایی که با EF کار می کنند، پیاده سازی می کنند. این الگوها در راستا یا به هدف ایجاد یک لایه ی انتزاعی بین لایه دسترسی داده (data access layer) و لایه ی business logic برنامه ی مورد نظر، پیاده سازی می شوند. پیاده سازی الگوهای ذکر شده، برنامه ی شما را از تغییراتی که در انبار داده (data store) اعمال می شود جدا می سازد و علاوه بر آن تست واحد (unit-testing) یا توسعه ی تست محور (test-driven development) خودکار را به مراتب آسان می سازد. با این حال، نوشتن کدهای اضافی بر سازمان جهت پیاده سازی این الگوها برای برنامه هایی که از EF بهره می گیرند، بنا به دلایل زیر همیشه توصیه نمی شود:

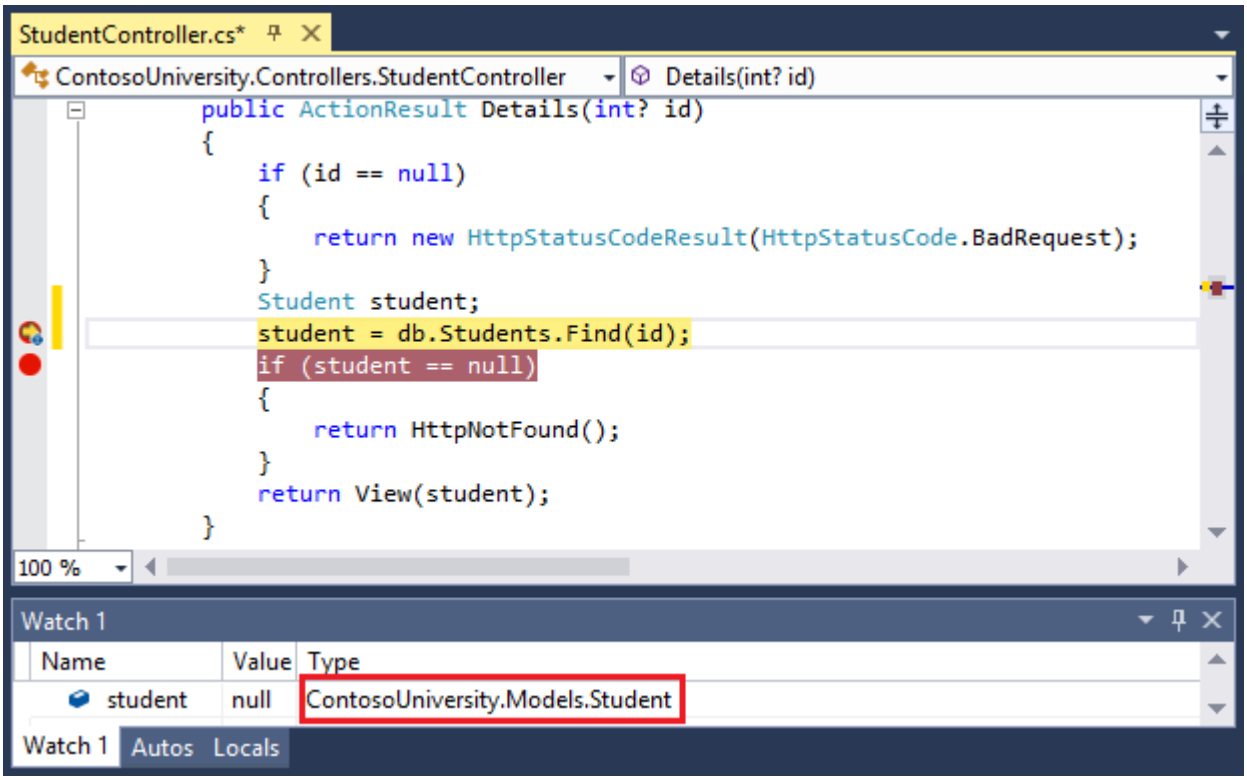
1. کلاس context در EF خودش کدهای شما را از کدهای مختص data store جدا می سازد.

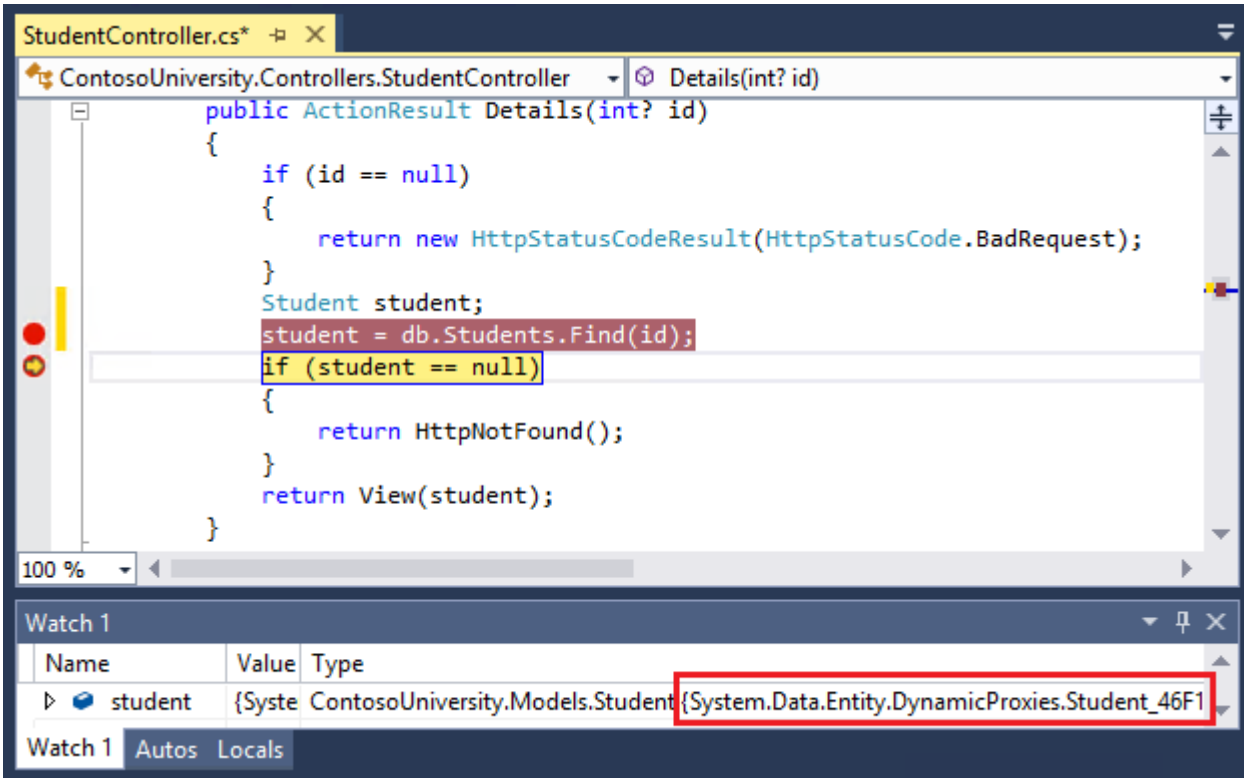
2. کلاس context در EF به مثابه ی یک کلاس واحد کاری (unit-of-work class) برای بروز رسانی هایی که با استفاده از EF انجام می دهید، عمل می کند.

3. امکاناتی که در ویرایش 6 تکنولوژیEF ارائه شده، پیاده سازی TDD را بدون نوشتن کد لازم برای repository آسان می سازد.

Proxy classes (کلاس های پیشکار)

هنگامی که EF نمونه موجودیت هایی را ایجاد می کند (به عنوان مثال در زمان اجرای یک query)، آن ها را به صورت یک نوع مشتق شده که به صورت پویا ساخته شده ایجاد می کند. حال این موجودیت ها به عنوان یک proxy یا پیشکار (یک کلاس که به عنوان یک رابط برای چیز دیگری ایفای نقش می کند) برای موجودیت مورد نظر عمل می کنند. به عنوان مثال می توانید به دو تصویر debugger زیر نگاه کنید. در تصویر اول، می بینید که متغیر Student، بلافاصله پس از اینکه entity را نمونه سازی می کنید، آن نوع Student مورد انتظار می باشد. در تصویر دوم، پس از اینکه EF برای خواندن یک موجودیت student از پایگاه داده بکار رفته، کلاس proxy برای شما قابل مشاهده می باشد.





این کلاس proxy با بازنویسی برخی از virtual property های موجودیت مورد نظر، hook هایی را برای اجرای عملیات به صورت خودکار به هنگام دسترسی به property مورد نظر وارد می کند. یکی از کاربردهای این مکانیزم، lazy loading (بارگذاری با تاخیر) می باشد.

در بیشتر مواقع لازم نیست که از این کاربرد (استفاده از proxy ها) آگاهی داشته باشید، اما در این میان موارد استثنایی نیز وجود دارد:

1. در برخی مواقع، بایستی مانع از این شوید که EF نمونه های proxy ایجاد کند. به عنوان مثال می توان به زمانی اشاره داشت که در آن موجودیت هایی را serialize می کنید. در این شرایط کلاس های POCO مورد نیاز می باشد و استفاده از proxy ها عملا جایز نیست. یک روش برای اجتناب از مشکلات ناشی از serialization، این است که بجای serialize کردن entity object ها ، این کار را برای DTO ها انجام دهید (اشیا انتقال داده را بجای شی موجودیت سریاله کنید). روش دیگر غیرفعال کردن proxy creation می باشد.

2. زمانی که شما با استفاده از عملگر new یک کلاس entity نمونه سازی می کنید، در واقع هیچ نمونه proxy در اختیار شما قرار داده نمی شود، بدین معنی که از قابلیت هایی همچون بارگذاری با تاخیر (lazy loading) و

ردیابی خودکار تغییرات (automatic change tracking) بهره مند نخواهید شد. اغلب این امر برای شما مشکلی را پیش نخواهد آورد؛ بدین معنا که در بیشتر موارد نیازی به بار گذاری با تاخیر نیست زیرا موجودیتی را ایجاد می کنید که در پایگاه داده موجود نمی باشد، همچنین اگر موجودیت مورد نظر را به عنوان Added علامت گذاری می کنید، در آن صورت change tracking بکار شما نخواهد آمد. با این حال اگر به دو امکان ذکر شده نیاز پیدا کردید، می توانید با استفاده از متد Create از کلاس DbSet، نمونه های entity جدیدی را با proxy ها ایجاد کنید.

3. چنانچه لازم بود نوع موجودیت را از نوع proxy بگیرید، در آن صورت می توانید متد GetObjectType را از کلاس ObjectContext مورد استفاده قرار دهید تا نوع موجودیت نمونه ی proxy type را دریافت کنید.

تشخیص و شناسایی خودکار تغییر

EF به واسطه ی مقایسه ی مقادیر جاری یک موجودیت با مقادیر اصلی و اولیه آن، پی می برد entity چگونه تغییر یافته (و به دنبال آن اینکه کدام updateها می بایست به پایگاه داده فرستاده شود). مقادیر اصلی به هنگام پیوست موجودیت یا زمانی که از آن موجودیت query گرفته می شود، ذخیره می گردند. برخی از توابع که تشخیص خودکار تغییر را فعال می سازند، به شرح زیر می باشند:

DbSet.Find
DbSet.Local
DbSet.Remove
DbSet.Add
DbSet.Attach
DbContext.SaveChanges
DbContext.GetValidationErrors
DbContext.Entry
DbChangeTracker.Entries

اگر تعداد entity های مورد ردیابی و در حال track قابل توجه است و در این میان یکی از توابع فهرست شده در بالا را بارها در یک حلقه صدا می زنید، در آن صورت با غیرفعال کردن قابلیت automatic change

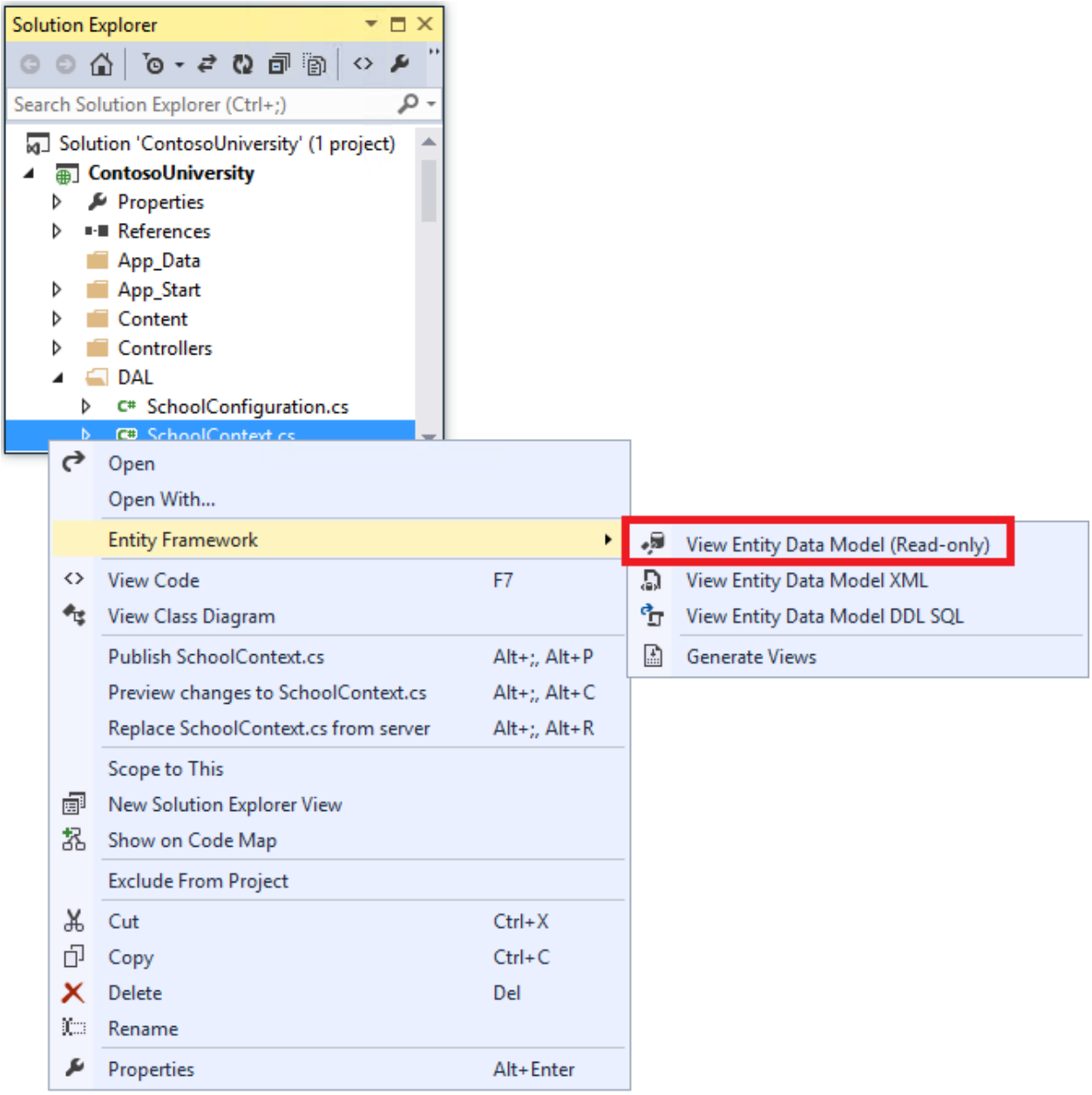
detection بهبود کارایی چشم گیری را شاهده خواهید بود. این کار را به واسطه ی استفاده از خاصیت AutoDetectChangesEnabled ترتیب می دهیم.

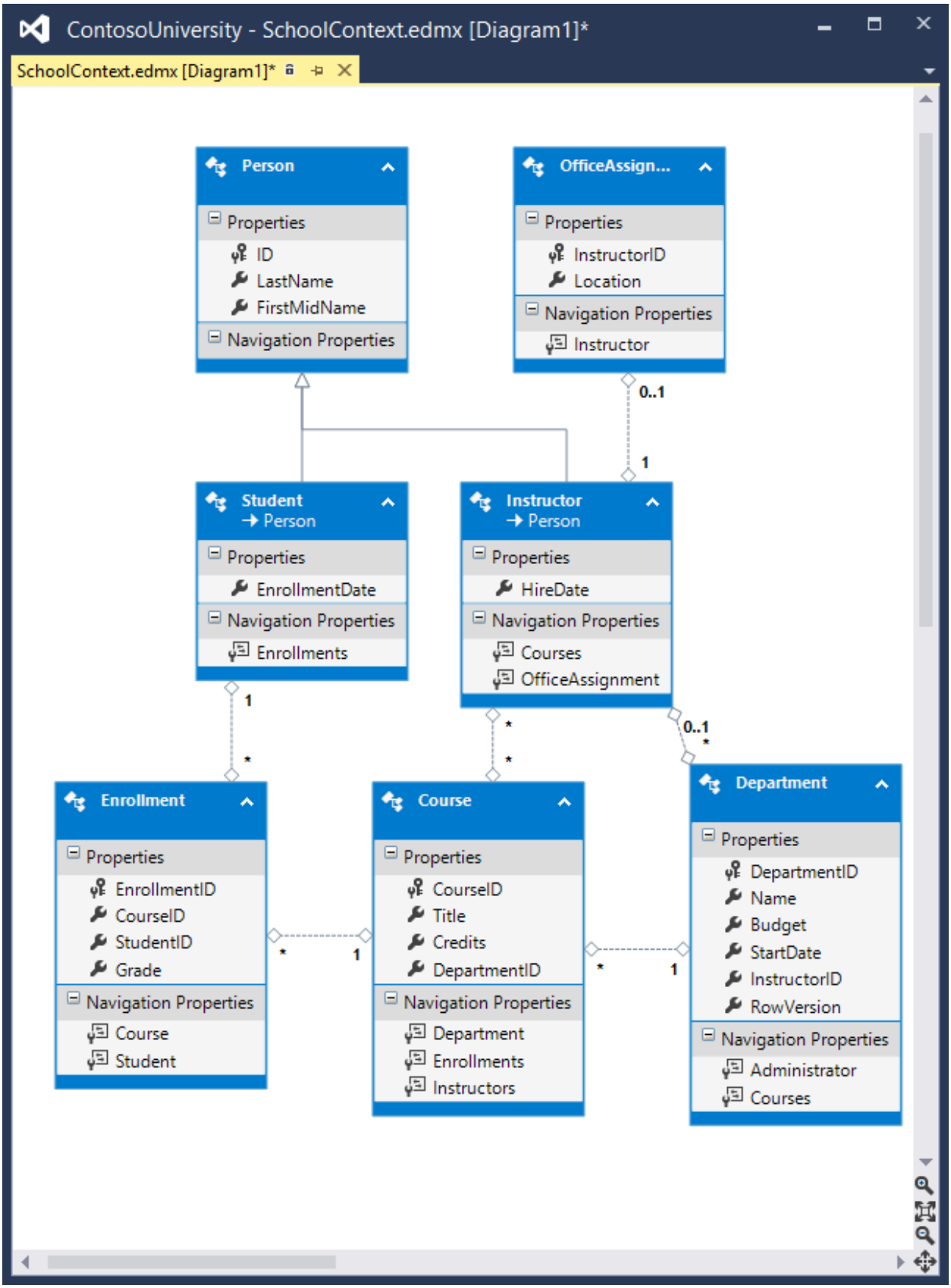
اعتبارسنجی خودکار

هنگامی که متد SaveChanges را صدا می زنید، EF به صورت پیش فرض داده های (موجود در) تمامی property های کلیه ی موجودیت های تغییر داده شده را پیش از بروز آوری پایگاه داده، اعتبار سنجی می کند. چنانچه تعداد زیادی موجودیت را بروز رسانی کرده و داده های مورد نظر را از قبل اعتبار سنجی نموده اید، در آن صورت اعتبار سنجی خودکار غیر ضروری بوده و می توانید طول فرایند ذخیره سازی تغییرات را با غیر فعال ساختن موقتی validation، به طور قابل توجهی کاهش دهید.

افزونه ی Entity Framework Power Tools

Entity Framework Power Tools یک افزونه یا add-in به محیط Visual Studio است که با استفاده از آن نمودارهای data model، که در این آموزش به نمایش گذاشته شد، را ایجاد کردیم. این افزونه کاربردهای دیگری هم دارد که ایجاد کلاس هایی بر اساس جداول موجود در پایگاه داده (برای اینکه بتوان از Code First با پایگاه داده ی خود استفاده کرد) یکی از آن ها است. پس از نصب افزونه ی مزبور، تعدادی گزینه های اضافه بر سازمان در context menu ها ظاهر می شود. به عنوان مثال، زمانی که روی کلاس context خود در پنجره ی Solution Explorer راست کلیک می کنید، گزینه ای برای ایجاد نمودار مشاهده خواهید کرد. زمانی که از Code First استفاده می کنید، اجازه ی تغییر data model در نمودار را نخواهید داشت، اما این امکان برای شما وجود دارد برخی آیتم ها را جابجا کرده و فهم بهتری پیدا کرد.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ContosoUniversity' (1 project)
ContosoUniversity
Properties
References
App_Data
App_Start
Content
Controllers
DAL
SchoolConfiguration.cs
Open
Open With...
Entity Framework
View Code F7
View Class Diagram
Publish SchoolContext.cs Alt+;, Alt+P
Preview changes to SchoolContext.cs Alt+;, Alt+C
Replace SchoolContext.cs from server Alt+;, Alt+R
Scope to This
New Solution Explorer View
Show on Code Map
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Delete Del
Rename
Properties Alt+Enter
View Entity Data Model (Read-only)
View Entity Data Model XML
View Entity Data Model DDL SQL
Generate Views


ContosoUniversity - SchoolContext.edmx [Diagram1]*
SchoolContext.edmx [Diagram1]*
Person
Properties
ID
LastName
FirstMidName
Navigation Properties
OfficeAssign...
Properties
InstructorID
Location
Navigation Properties
Instructor
0..1
1
Student
→ Person
Properties
EnrollmentDate
Navigation Properties
Enrollments
Instructor
→ Person
Properties
HireDate
Navigation Properties
Courses
OfficeAssignment
1
*
*
*
0..1
*
Enrollment
Properties
EnrollmentID
CourseID
StudentID
Grade
Navigation Properties
Course
Student
Course
Properties
CourseID
Title
Credits
DepartmentID
Navigation Properties
Department
Enrollments
Instructors
Department
Properties
DepartmentID
Name
Budget
StartDate
InstructorID
RowVersion
Navigation Properties
Administrator
Courses
*
1
*
1