بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

مدیریت همروندی یا برنامه نویسی موازی در MVC

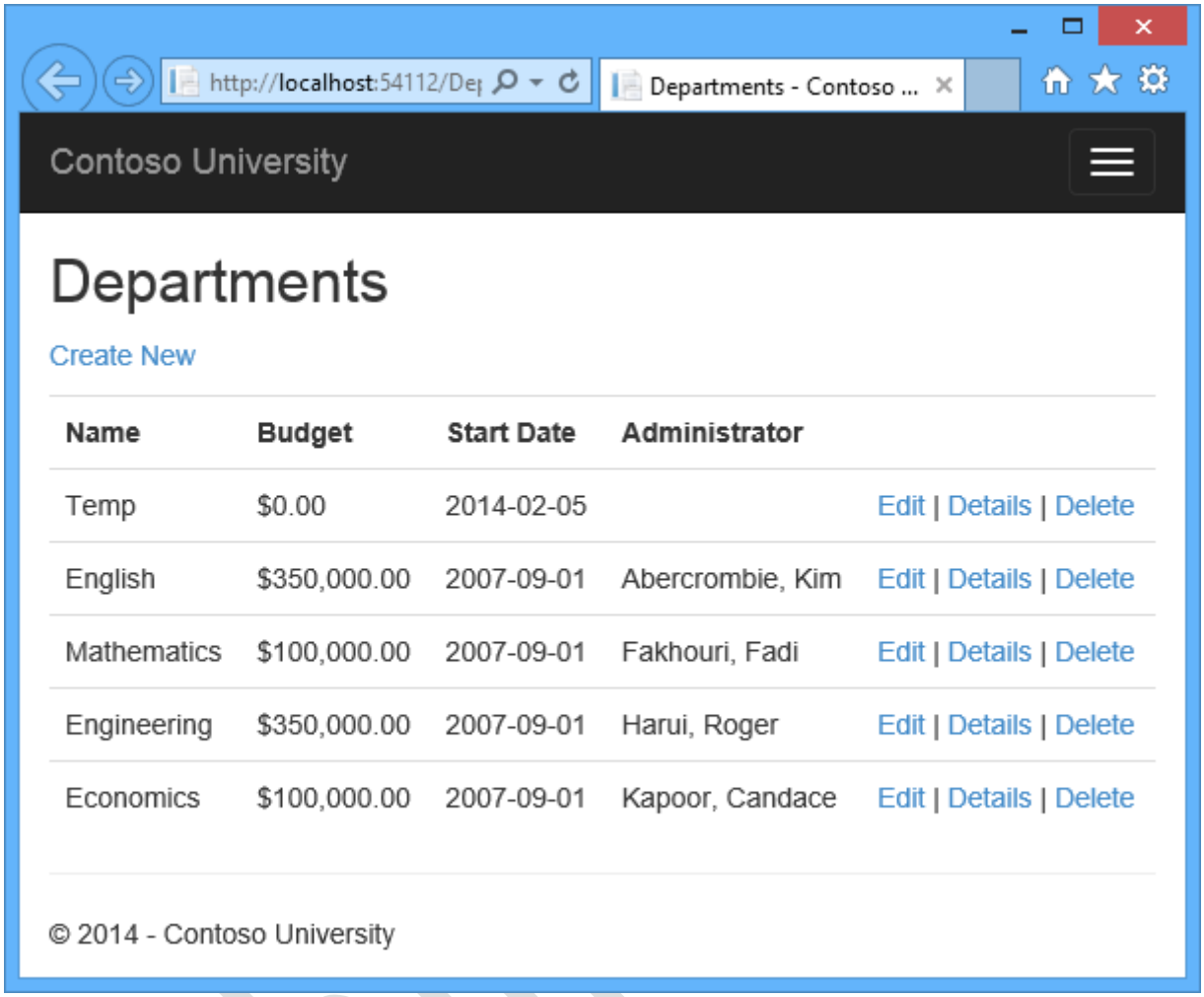
مدرس : مهندس افشین رفوآ

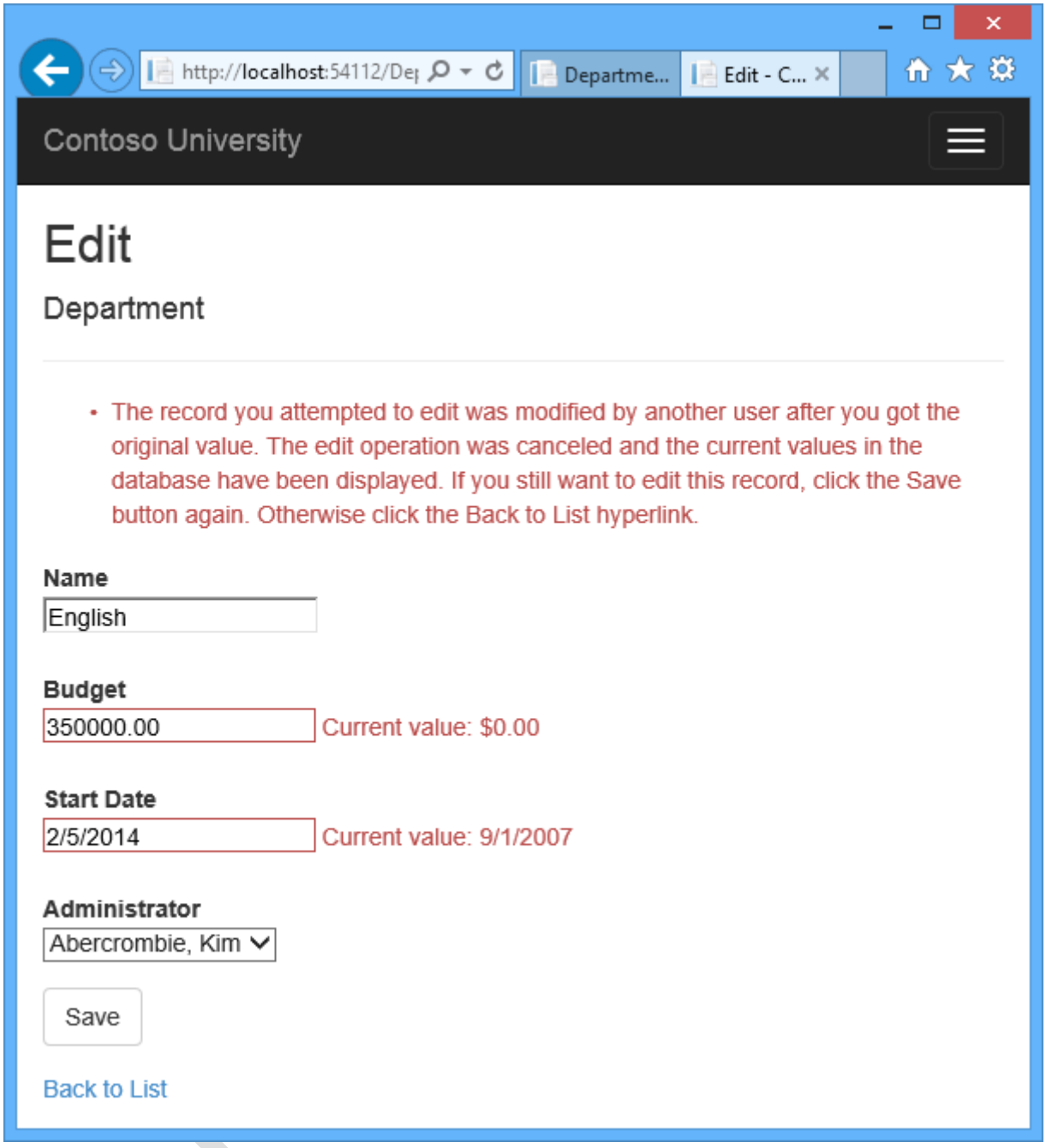
دوره آموزش MVC

مدیریت همروندی یا برنامه نویسی موازی (concurrency) با EF6 در یک برنامه ی MVC 5

در آموزش های قبلی با نحوه ی بروز رسانی داده ها آشنا شدید. آموزش حاضر به شما می آموزد، زمانی که چند کاربر سعی بر بروز آوری entity یکسان می کنند، چگونه تداخلات (conflict) ناشی از آن را مدیریت کنید.

آن صفحاتی که با موجودیت Department کار می کنند را گونه ای ویرایش کنید که بتوانند خطاهای همروندی (concurrency error) را در صورت رخداد آن ها، مدیریت کنند. تصاویر زیر صفحات Index و Delete را به ضمیمه ی خطاهایی که با وقوع تداخل همروندی نمایش داده می شود را نشان می دهد.


http://localhost:54112/Dep
Departments - Contoso ...
Contoso University
Departments
Create New
Name
Budget
Start Date
Administrator
Temp
$0.00
2014-02-05
Edit | Details | Delete
English
$350,000.00
2007-09-01
Abercrombie, Kim
Edit | Details | Delete
Mathematics
$100,000.00
2007-09-01
Fakhouri, Fadi
Edit | Details | Delete
Engineering
$350,000.00
2007-09-01
Harui, Roger
Edit | Details | Delete
Economics
$100,000.00
2007-09-01
Kapoor, Candace
Edit | Details | Delete
© 2014 - Contoso University



تداخلات همزمانی (concurrency conflict)

زمانی رخ می دهد که کاربری داده های یک entity را برای ویرایش نمایش می دهد و در این میان کاربری دیگر سعی می کند همان داده ها را، قبل از اینکه تغییرات اعمال شده توسط کاربر اول در پایگاه داده نوشته شود، بروز رسانی کند. چنانچه شما شناسایی چنین تداخلاتی را فعال سازی نکنید، آن تغییراتی که توسط آخرین کاربر اعمال می شود، تمامی بروز رسانی دیگر کاربران را بازنویسی می کند. در بسیاری از برنامه ها، این ریسک

پذیرفتنی است، یعنی برنامه هایی که کاربران آن محدود است و تعداد اندکی بروز رسانی در آن صورت می گیرد یا بازنویسی برخی تغییرات در آن مشکل بزرگی را ایجاد نمی کند، هزینه ی برنامه نویسی برای همروندی نسبت به مزایایی که به دنبال دارد بیشتر است. در این صورت، نیازی به تنظیم و پیکربندی برنامه برای مدیریت تداخلات همزمانی نیست.

کنترل همروندی بدبینانه – قفل گذاری (pessimistic concurrency)

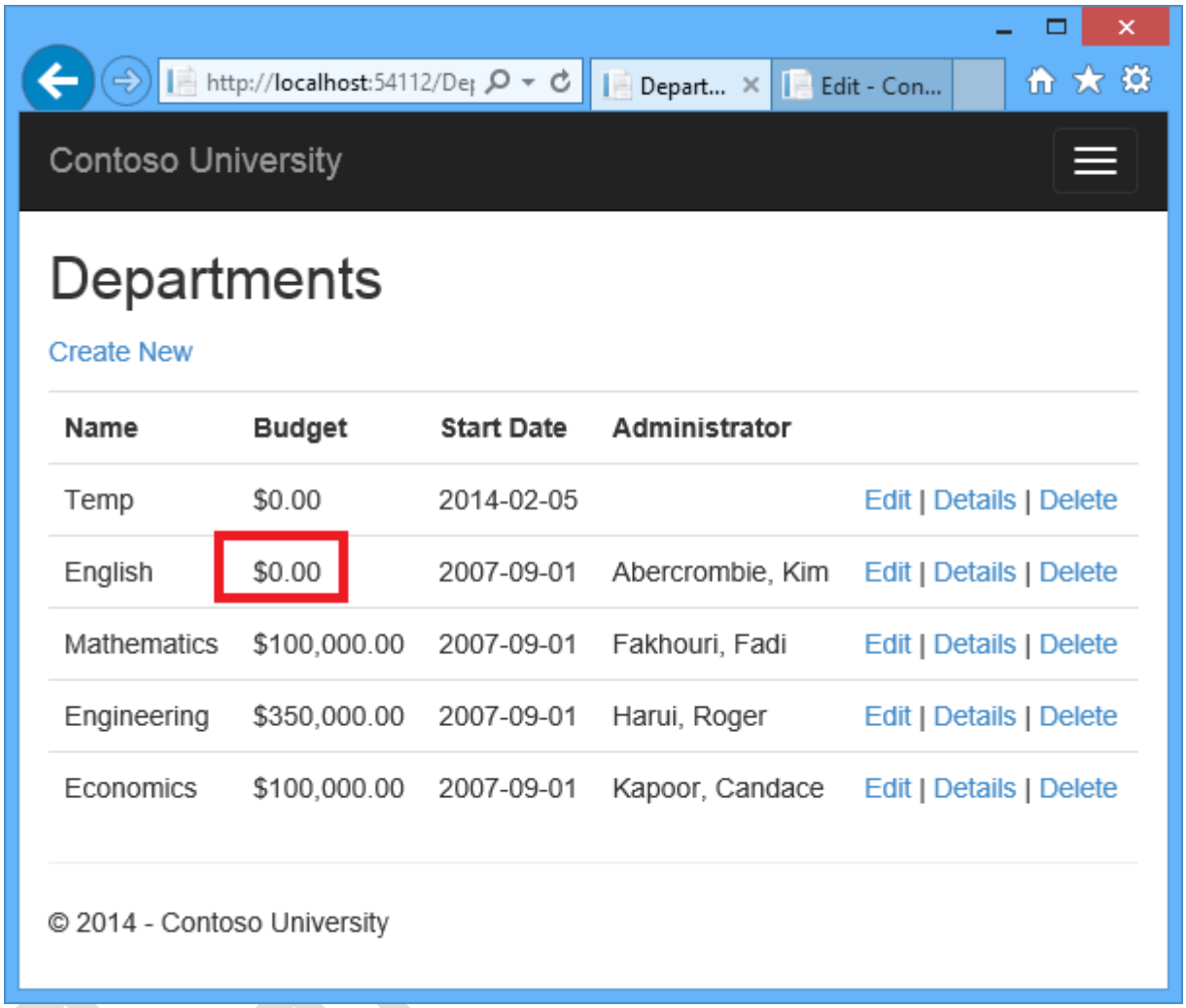
یکی از روش های نوشتن برنامه ای که قادر باشد از از دست رفت داده در سناریوهای همروندی جلوگیری کند، اعمال قفل های پایگاه داده است. به این امر کنترل همروندی بدبینانه (pessimistic concurrency) می گویند. در این روش به وسیله قفل گذاری جلوی هرگونه همروندی گرفته می‌شود. تا زمانی که یک دستور درحال اجراست جلوی اجرای دستورهایی که ممکن است مانع اجرای صحیح دستور نخست شوند گرفته می‌شود. برای مثال، پیش از اینکه سطری را از پایگاه داده بخوانیم، یک قفل دسترسی فقط خواندنی (read-only) یا بروز رسانی درخواست می کنیم. (قفل امتیاز دستیابی به یک واحد داده است که توسط سیستم قفل گذاری به یک تراکنش داده می‌شود و یا از او پس گرفته می‌شود.) اگر یک سطر را برای بروز رسانی قفل کنید، هیچ کاربر دیگری قادر به قفل گذاری بر روی آن سطر برای بروز رسانی یا فقط ـ خواندن نخواهد بود زیرا در آن صورت تنها یک کپی از داده ها در حال ویرایش دریافت خواهد کرد. اگر یک سطر را برای سطح دسترسی فقط خواندن قفل کنید، در آن صورت دیگر کاربران می توانند آن را برای دسترسی فقط ـ خواندن قفل کنند اما این اجازه را در مورد بروز رسانی نخواهد داشت.

اعمال قفل ها و مدیریت آن ها معایبی را نیز به همراه دارد که از جمله می توان به پیچیدگی برنامه نویسی آن اشاره کرد. این کار همچنین لازمه ی منابع مدیریتی بسیار سنگین است. بعلاوه در صورت بالا رفتن تعداد کاربران برنامه ممکن است مشکلاتی در زمینه کارایی به وجود آورده و افت آن را به همراه داشته باشد. بنا به دلایل ذکر شده، تمامی سیستم های مدیریتی پایگاه داده از همروندی بدبینانه پشتیبانی نمی کنند.

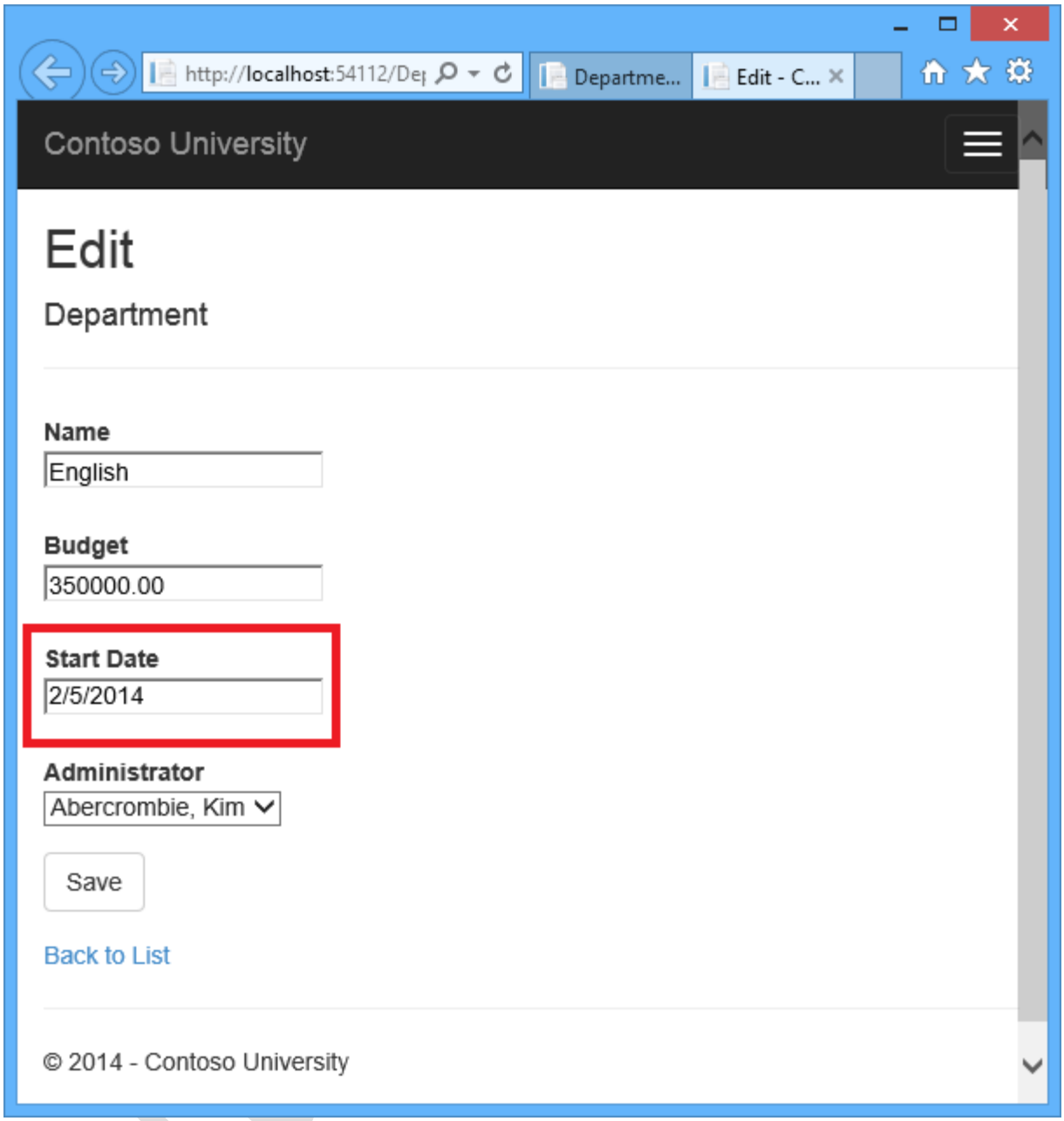
کنترل همروندی خوش‌بینانه (Optimistic Concurrency)

گزینه ی مقابل همروندی بدبینانه، همروندی خوش بینانه (optimistic concurrency) می باشد. در همروندی خوش بینانه به تداخلات همزمانی اجازه ی رخ دادن داده می شود و در صورت روی دادن آن ها،

واکنش مناسب صورت می گیرد. به عنوان مثال، کاربری به نام john صفحه ی Departments Edit را اجرا کرده و مقدار Budget را برای فیلد English از $350,000.00 به $0.00 تغییر می دهد.



قبل از اینکه john دکمه ی Save را کلیک کند، کاربر دیگری به نام jane همان صفحه را اجرا کرده و فیلد Start Date را از 9/1/2007 به 8/8/2013 تغییر می دهد.



ابتدا John دکمه ی Save را کلیک کرده، تغییرات خود را با بازگشت مرورگر به صفحه ی Index مشاهده می کند، سپس jane دکمه ی Save را کلیک می کند. اینکه بعد چه اتفاقی رخ می دهد، بسته به نحوه ی مدیریت تداخلات همروندی توسط شما دارد.

1. می توانید حساب اینکه کاربر کدام property را اصلاح کرده نگه دارید و بر اساس آن فقط ستون های مربوطه را در پایگاه داده بروز رسانی نمایید. در مثالی که ذکر شد، هیچ داده ای از دست نمی رود زیرا

property های مختلف توسط دو کاربر متفاوت بروز رسانی شده. دفعه ی بعدی که کاربری English department را پیمایش می کند، تغییرات اعمال شده توسط هر دو کاربر را مشاهده خواهد کرد: start date (تاریخ شروع) با مقدار 8/8/2013 و budget ای (بودجه) با مقدار $0.00.

این روش بروز رسانی می تواند تعداد رخدادهای تداخل همروندی را که ممکن است منجر به از دست رفت داده شود، کاهش دهد. اما در صورت اعمال تغییرات متقابل به یک property از یک موجودیت، دیگر قادر به جلوگیری از از دست رفت اطلاعات نخواهد بود. اینکه EF به این روش عمل کند، بستگی به نحوه ی پیاده سازی شما از Update code دارد. این کار در یک برنامه ی تحت وب معمولا امکان پذیر و کاربردی نیست، زیرا در آن صورت لازم است برای اینکه علاوه بر مقادیر جدید حساب تمامی مقادیر property های اولیه یک موجودیت را داشته باشیم، مقادیر زیادی از state ها را حفظ و نگه داری کنیم. حفظ و نگهداری مقدار زیادی از state ها می تواند اثر سوء بر کارایی برنامه داشته باشد، زیرا این کار لازمه ی اشغال منابع سرور بوده یا اطلاعات مربوط به آن را می بایست در خود صفحه ی وب (برای مثال در فیلدهای پنهان) و یا یک cookie گنجاند.

2. می توانید اجازه دهید تغییرات jane تغییرات اعمال شده توسط john را بازنویسی کند. حال دفعه ی بعدی که کاربری English department را پیمایش می کند، تاریخ 8/8/2013 و مقدار بازگردانده شده ی $350,000.00 را مشاهده می کند. این سناریو، client wins یا last in wins خوانده می شود (بدین معنا که مقادیر ارائه شده توسط client بر مقادیر موجود در انبار داده یا data store اولویت دارد). همان طور که بخش مقدمه ی این آموزش تشریح شد، اگر هیچ کدنویسی برای مدیریت همروندی انجام ندهید، این اتفاق خود به صورت پیش فرض رخ می دهد.

3. می توان کاری کرد تغییرات اعمال شده توسط jane در پایگاه داده بروز رسانی نشود. به طور معمول یک پیغام خطا برای کاربر (jane) نمایش می دهیم، وضعیت جاری داده ها را به اطلاع وی می رسانیم، سپس به کاربر مذکور اجازه می دهیم در صورت تمایل (اگر می خواهد همواره اصلاحاتی را ایجاد کند) تغییرات خود را مجددا اعمال نماید. این سناریو تحت عنوان store wins شناخته می شود (بدین معنا که مقادیر انبار داده یا data store بر مقادیر ارائه شده توسط client اولویت دارد). در آموزش حاضر، این روش را پیاده خواهیم کرد. در این روش هیچ تغییری بازنویسی نمی شود، مگر اینکه کاربر قبل آن مطلع شده باشد.

تشخیص تداخلات همزمانی

می توان تداخلات همزمانی را با مدیریت خطاهای OptimisticConcurrencyException که توسط EF صادر می شود، برطرف ساخت. برای این که EF تشخیص دهد چه زمانی بایستی خطاهای مربوطه را صادر کند، ابتدا لازم است آن تداخلات را شناسایی کند. بنابراین، می بایست پایگاه داده و data model را به درستی پیکربندی نمود. گزینه هایی که برای فعال سازی conflict detection (تشخیص تداخل) در دست دارید به شرح زیر می باشند:

1. در جدول پایگاه داده، یک tracking column (ستون ردیابی) ایجاد می کنیم. این ستون را برای رهگیری و تعیین زمان اصلاح سطر مورد نظر بکار می بریم. سپس می توانیم EF را طوری تنظیم کنیم که آن ستون را در عبارت Where دستورهای Update و Delete اس کیو ال قرار دهد.

نوع داده ی ستون مزبور معمولا rowversion می باشد. مقدار rowversion یک sequential number (عدد ترتیبی) است که با هر بروز رسانی سطر مورد نظر، آن عدد افزایش می یابد. در دستور Update یا Delete، عبارت Where مقدار اصلی tracking column (نسخه ی اصلی و اولیه ی سطر مورد نظر) را نگه می دارد. چنانچه سطری که در دست بروزرسانی است توسط کاربر دیگر تغییر داده شود، در آن صورت مقدار موجود در ستون rowversion با مقدار اصلی یا اولیه ی آن متفاوت خواهد بود و از این رو دستور Update یا Delete نمی تواند (بخاطر عبارت Where) سطر مورد نظر را برای آپدیت پیدا کند. هنگامی که EF پی ببرد که هیچ سطری یا رکوردی توسط دستور Update یا Delete بروز آوری نشده (بدین معنا که تعداد سطرهای ویرایش شده برابر با صفر باشد)، در آن صورت EF آن را یک تداخل همروندی درنظر گرفته و رفتار خود را بر اساس آن تنظیم می کند.

2. EF را گونه ای تنظیم کنید که مقادیر اولیه ی تمامی ستون های موجود در جدول را داخل عبارت Where دستورات Update و Delete قرار دهد.

همان طور که در روش اول تشریح شد، چنانچه از زمانی که سطر برای اولین بار خوانده شد، چیزی در آن تغییر داده شده باشد، در آن صورت عبارت Where هیچ سطری برای بروز رسانی باز نمی گرداند. EF این رخداد را به

عنوان یک تداخل همروندی (concurrency conflict) تفسیر می کند. برای آن دسته از جداول پایگاه داده که دربردارنده ی ستون های متعددی هستند، این روش ممکن است باعث ایجاد عبارت های بسیار طولانی Where شود و همچنین شما را مجاب کند مقادیر زیادی از state ها را حفظ و نگداری کنید. پیش تر نیز ذکر شد که حفظ مقادیر زیادی از state ها می تواند کارایی برنامه را تحت تاثیر قرار دهد. بنا به دلایل مذکور، استفاده از این روش توصیه نمی شود.

اگر می خواهید این روش را برای کنترل همروندی پیاده کنید، در آن صورت بایستی تمامی خاصیت های غیر کلید اصلی (non-primary-key) را در موجودیتی که می خواهید concurrency آن را ردیابی کنید، با افزودن خصیصه ی ConcurrencyCheck به آن ها علامت گذاری کنید. این تغییر به EF امکان می دهد تمامی ستون ها را در عبارت WHERE دستورهای UPDATE اضافه (include) کند.

در ادامه ی این آموزش، یک rowversion tracking property به موجودیت Department اضافه خواهیم کرد، یک controller به همره view هایی ایجاد کرده و درنهایت همه چیز را بررسی کرده و از عملکرد صحیح آن ها اطمینان حاصل می کنیم.

افزودن یک property همروندی خوشبینانه به موجودیت Department

فایل Models\Department.cs را باز کرده و یک tracking property به نام RowVersion اضافه کنید:

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

  [StringLength(50, MinimumLength = 3)]
  public string Name { get; set; }

  [DataType(DataType.Currency)]
  [Column(TypeName = "money")]
  public decimal Budget { get; set; }

  [DataType(DataType.Date)]
  [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
  [Display(Name = "Start Date")]
  public DateTime StartDate { get; set; }
```

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

        [Timestamp]
        public byte[] RowVersion { get; set; }

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

خصیصه ی Timestamp مشخص می کند که این ستون باید در عبارت Where دستورهای Update و Delete ارسالی به پایگاه داده گنجانده شود. خصیصه ی ذکر شده از آنجایی Timestamp خوانده می شود که نسخه های قبلی SQL Server از نوع داده ی timestamp، قبل از اینکه rowversion جایگزین آن شود، استفاده می کردند. معادل rowversion در .NET. یک byte array (آرایه ای از نوع byte) می باشد.

در صورت تمایل به استفاده از fluent API، می توانید از متد IsConcurrencyToken برای مشخص کردن tracking property مورد نظر استفاده کنید:

```
        modelBuilder.Entity<Department>()
    .Property(p => p.RowVersion).IsConcurrencyToken();
```

افزودن یک خاصیت باعث تغییر database model شد، از این باید یک migration دیگر اجرا کنید. در پنجره ی PMC، دستورات زیر را وارد نمایید:

```
Add-Migration RowVersion
Update-Database
```

ویرایش کنترلر Department

در فایل Controllers\DepartmentController.cs، یک دستور using اضافه کنید:

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

داخل فایل DepartmentController.cs، هر چهار نمونه ی "LastName" را به "FullName" تغییر دهید تا لیست های کشویی (drop-down list) department administrator علاوه بر last name آن instructor، full name آن را نیز شامل شود.

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

کد جاری متد HttpPost Edit را با کد زیر جایگزین کنید:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Edit(int? id, byte[] rowVersion)
{
    string[] fieldsToBind = new string[] { "Name", "Budget", "StartDate", "InstructorID", "RowVersion" };

    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    var departmentToUpdate = await db.Departments.FindAsync(id);
    if (departmentToUpdate == null)
    {
        Department deletedDepartment = new Department();
        TryUpdateModel(deletedDepartment, fieldsToBind);
        ModelState.AddModelError(string.Empty,
            "Unable to save changes. The department was deleted by another user.");
        ViewBag.InstructorID = new SelectList(db.Instructors, "ID", "FullName", deletedDepartment.InstructorID);
        return View(deletedDepartment);
    }

    if (TryUpdateModel(departmentToUpdate, fieldsToBind))
    {
        try
        {
            db.Entry(departmentToUpdate).OriginalValues["RowVersion"] = rowVersion;
            await db.SaveChangesAsync();

            return RedirectToAction("Index");
        }
        catch (DbUpdateConcurrencyException ex)
        {
            var entry = ex.Entries.Single();
            var clientValues = (Department)entry.Entity;
            var databaseEntry = entry.GetDatabaseValues();
            if (databaseEntry == null)
            {
                ModelState.AddModelError(string.Empty,
                    "Unable to save changes. The department was deleted by another user.");
            }
            else
```

```
            {
                var databaseValues = (Department)databaseEntry.ToObject();

                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.");
                departmentToUpdate.RowVersion = databaseValues.RowVersion;
            }
        }
        catch (RetryLimitExceededException /* dex */)
        {
            //Log the error (uncomment dex variable name and add a line here to write a log.
            ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists, see your
system administrator.");
        }
    }
    ViewBag.InstructorID = new SelectList(db.Instructors, "ID", "FullName", departmentToUpdate.InstructorID);
    return View(departmentToUpdate);
}
```

اگر متد FindAsync مقدار null را بازگرداند، بدین معنا است که department توسط کاربر دیگر حذف شده است. کدی که نمایش داده شده مقادیر ارسالی فرم را برای ایجاد موجودیت department بکار می برد تا بدین وسیله صفحه ی Edit با پیغام خطای مربوطه مجددا نمایش داده شود. اگر صرفا یک پیام خطا را بدون اینکه لازم باشد فیلدهای department را بار دیگر نشان دهیم، به نمایش بگذاریم در آن صورت دیگر نیازی هم به ایجاد مجدد موجودیت department نیست. این روش می تواند جایگزینی برای روش قبلی باشد.

View مقدار اصلی و اولیه ی RowVersion را در یک فیلد پنهان ذخیره می کند و متد مورد نظر آن مقدار را در قالب پارامتر rowVersion دریافت می کند. پیش از اینکه شما متد SaveChanges را فراخوانی کنید، باید مقدار اولیه ی خاصیت RowVersion را در مجموعه (collection) OriginalValues آن موجودیت قرار

دهید. سپس هنگامی که EF دستور UPDATE اس کیو ال را ایجاد می کند، آن دستور دربردارنده ی یک عبارت WHERE خواهد بود. حال عبارت WHERE آن سطری را جستجو می کند که دارای مقدار اصلی RowVersion باشد.

چنانچه هیچ یک از سطرها توسط دستور UPDATE مورد تغییر قرار نگیرد (هیچ یک از سطرها حاوی مقدار اصلی RowVersion نباشد)، در آن صورت EF یک خطای DbUpdateConcurrencyException صادر می کند و کد موجود در قطعه ی catch موجودیت Department ویرایش شده (متاثر) را از شی exception می گیرد.

```
var entry = ex.Entries.Single();
```

این شی دربردارنده ی مقادیر جدید است که توسط کاربر در خاصیت (property) Entity آن وارد شده و شما می توانید مقادیر خوانده شده از پایگاه داده را با فراخوانی متد GetDatabaseValues بازیابی کنید.

```
var clientValues = (Department)entry.Entity;
      var databaseEntry = entry.GetDatabaseValues();
```

متد GetDatabaseValues، در صورتی که سطر مورد نظر توسط کاربری از پایگاه داده حذف شده باشد، مقدار null را برمی گرداند؛ در غیر این صورت می بایست برای دسترسی به propertyهای Department، شی بازگشتی را به کلاس Department تبدیل کنید. (از آنجایی که قبلا deletion را بررسی کردیم، databaseEntry تنها در صورتی null می شود که department پس از اجرای FindAsync و قبل از اجرای SaveChanges حذف شود).

```
if (databaseEntry == null)
{
  ModelState.AddModelError(string.Empty,
    "Unable to save changes. The department was deleted by another user.");
}
else
{
  var databaseValues = (Department)databaseEntry.ToObject();
```

سپس، کد مورد نظر یک پیام خطای سفارشی به ازای هر ستون که دارای مقادیر پایگاه داده ی متفاوت از آنچه کاربر در صفحه ی Edit وارد کرده، اضافه می کند:

```
if (databaseValues.Name != currentValues.Name)
ModelState.AddModelError("Name", "Current value: " + databaseValues.Name);
// ...
```

یک پیام خطای طولانی تر شرح می دهد که چه رخ داده و چه کاری می توان برای حل آن انجام داد:

```
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.");
```

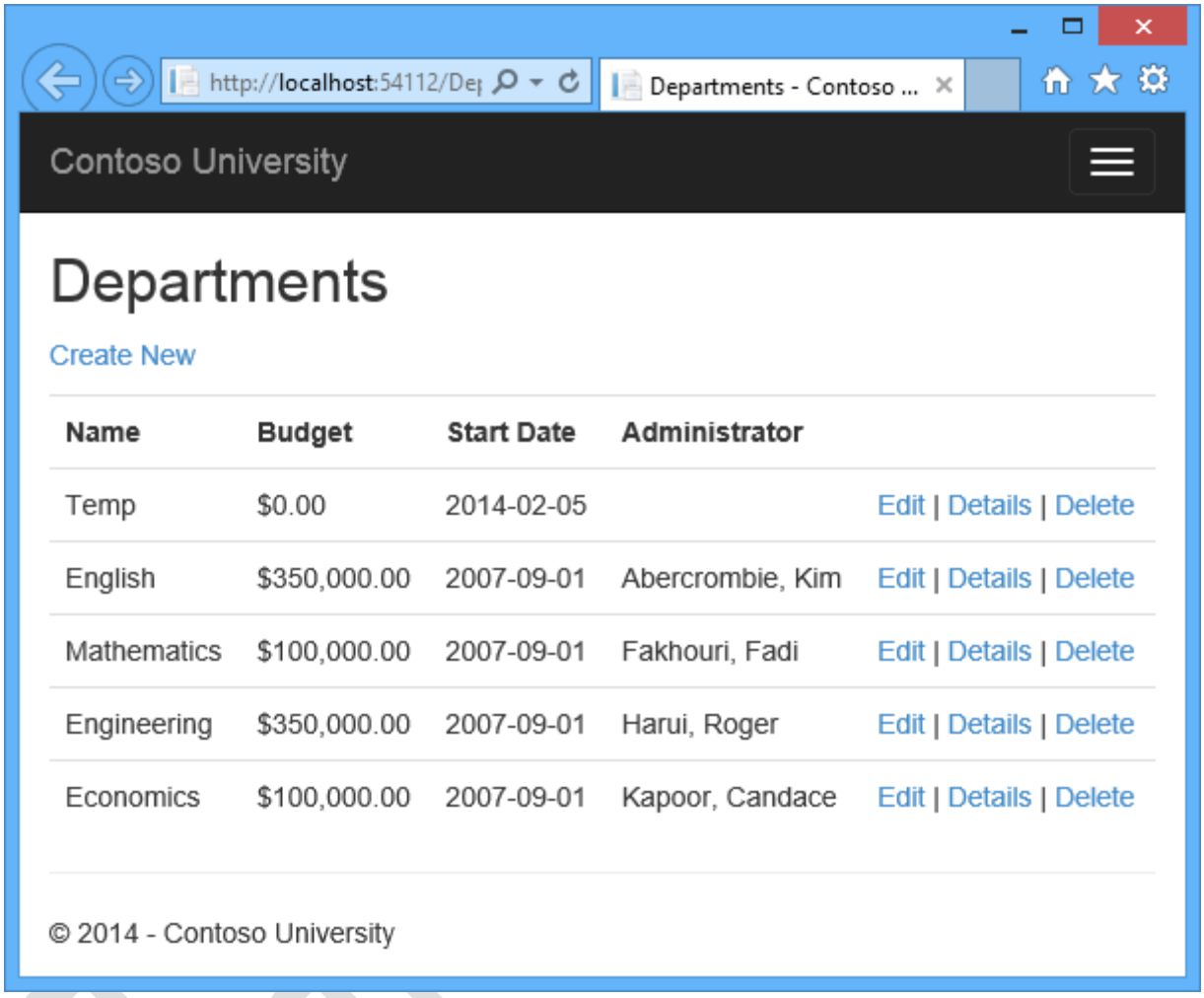
در نهایت، کد مقدار RowVersion شی Department را به مقدار جدید بازیابی شده از پایگاه داده تنظیم می کند. این مقدار جدید RowVersion هنگامی که صفحه ی Edit مجددا نمایش داده می شود، در فیلد پنهان ذخیره می گردد، و دفعه ی بعدی که کاربر دکمه ی Saves را کلیک می کند، فقط خطاهای همروندی (concurrency errors) که از زمان نمایش مجدد صفحه ی Edit رخ داده، گرفته (catch) می شوند.

در فایل Views\Department\Edit.cshtml، یک فیلد پنهانی که بتوان مقدار خاصیت RowVersion را در آن ذخیره کرد، بلافاصله پس از فیلد پنهانی مربوط به خاصیت DepartmentID اضافه کنید:

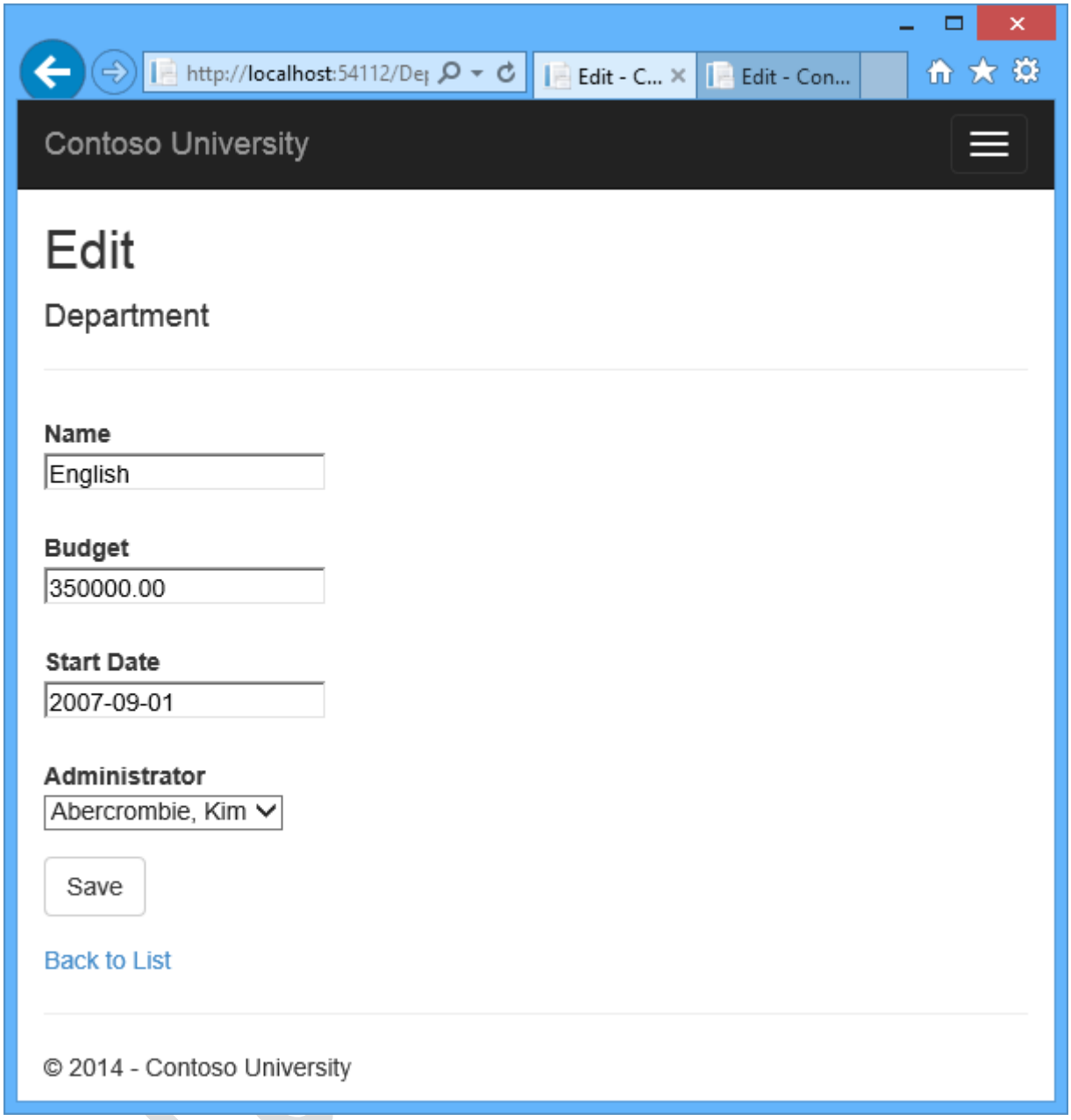
```
@model ContosoUniversity.Models.Department

@{
  ViewBag.Title = "Edit";
}

<h2>Edit</h2>


@using (Html.BeginForm())
{
  @Html.AntiForgeryToken()

  <div class="form-horizontal">
    <h4>Department</h4>
    <hr />
    @Html.ValidationSummary(true)
    @Html.HiddenFor(model => model.DepartmentID)
    @Html.HiddenFor(model => model.RowVersion)
```

تست مدیریت همروندی خوشبینانه (Optimistic Concurrency)

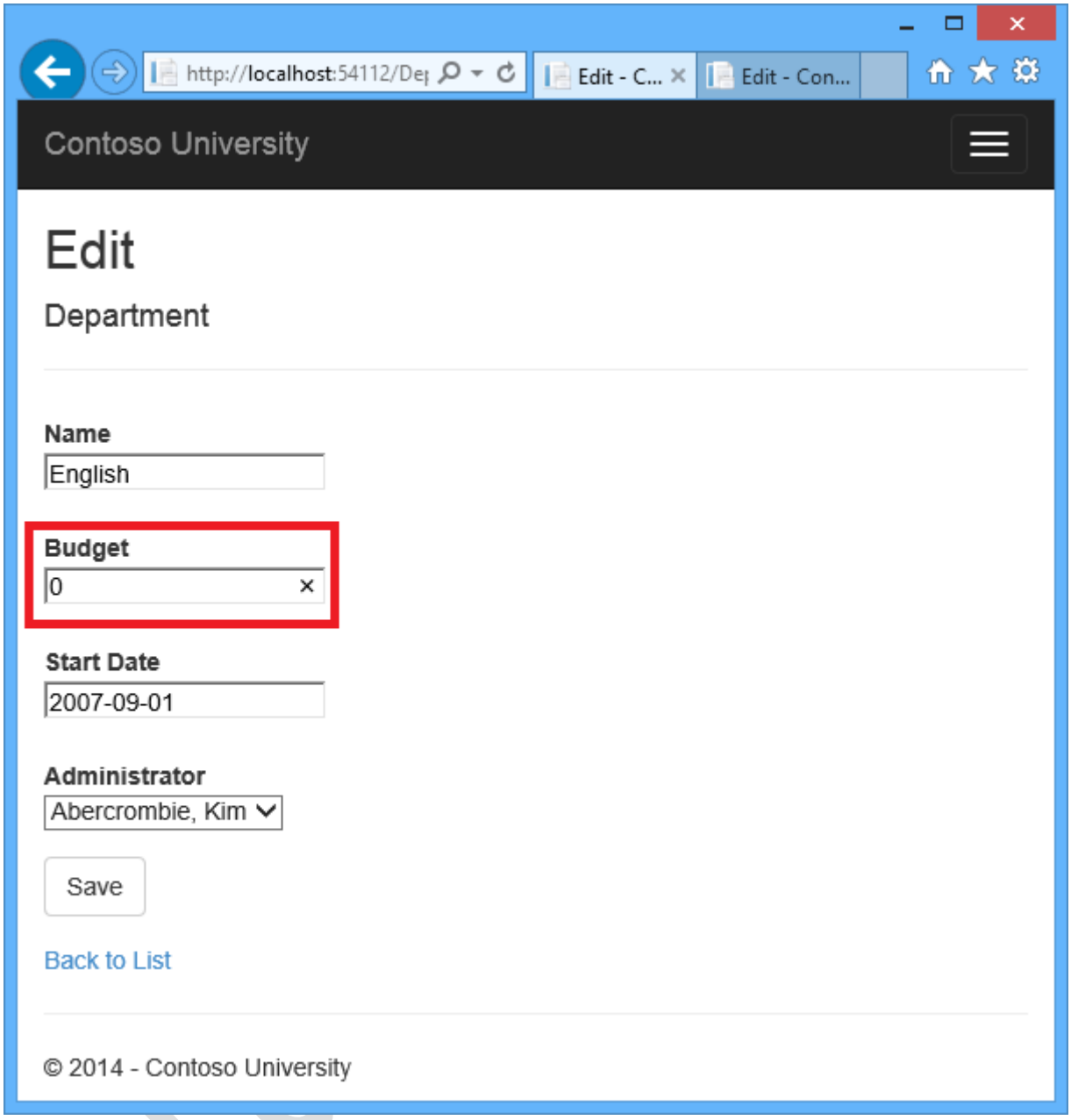
سایت را راه اندازی کرده و Departments را کلیک کنید:



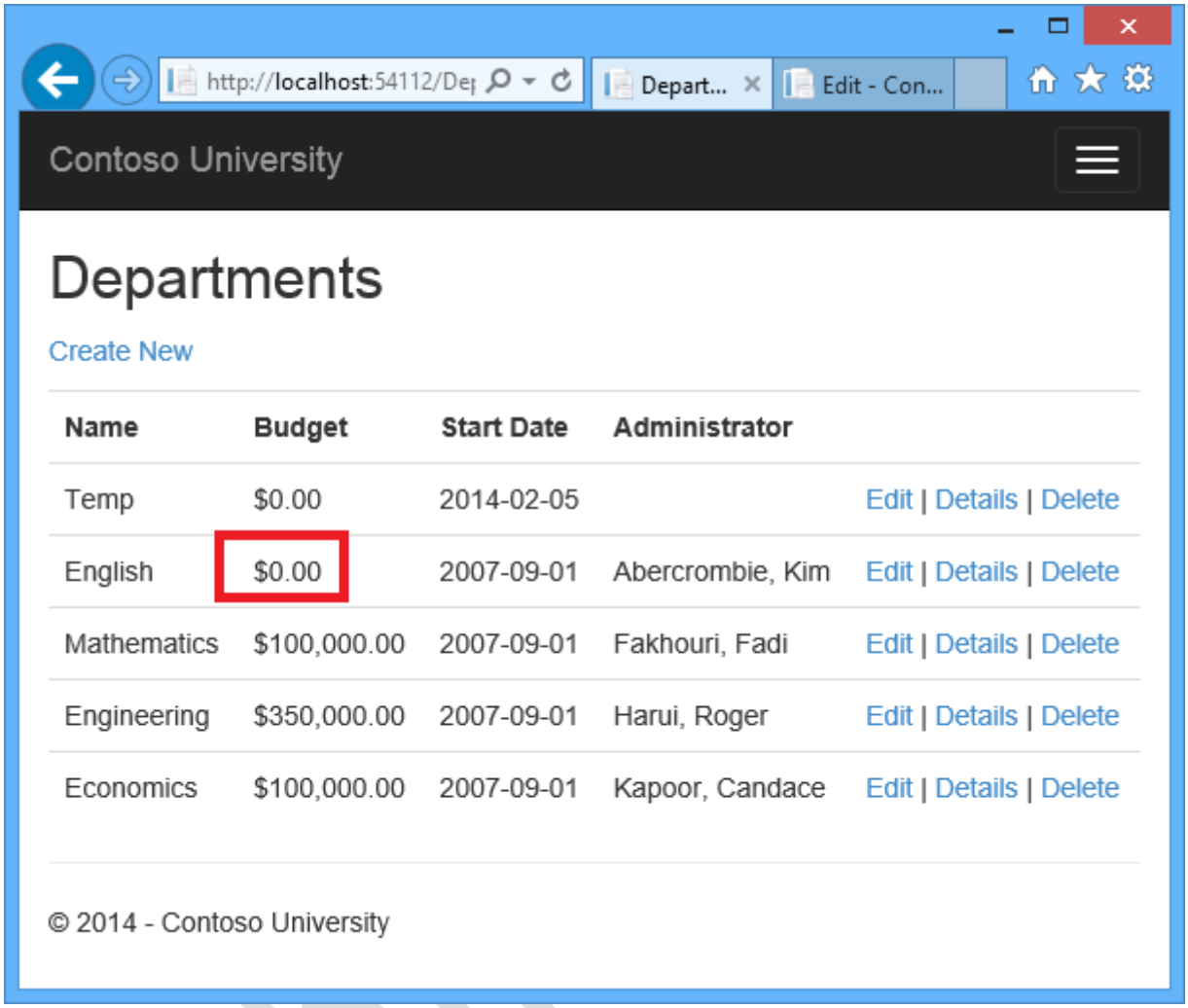
بر روی لینک Edit مربوط به فیلد English راست کلیک کرده و Open in new tab را انتخاب کنید، سپس لینک Edit فیلد English را کلیک کنید. هر دو تب اطلاعات یکسان را به نمایش می گذارند:



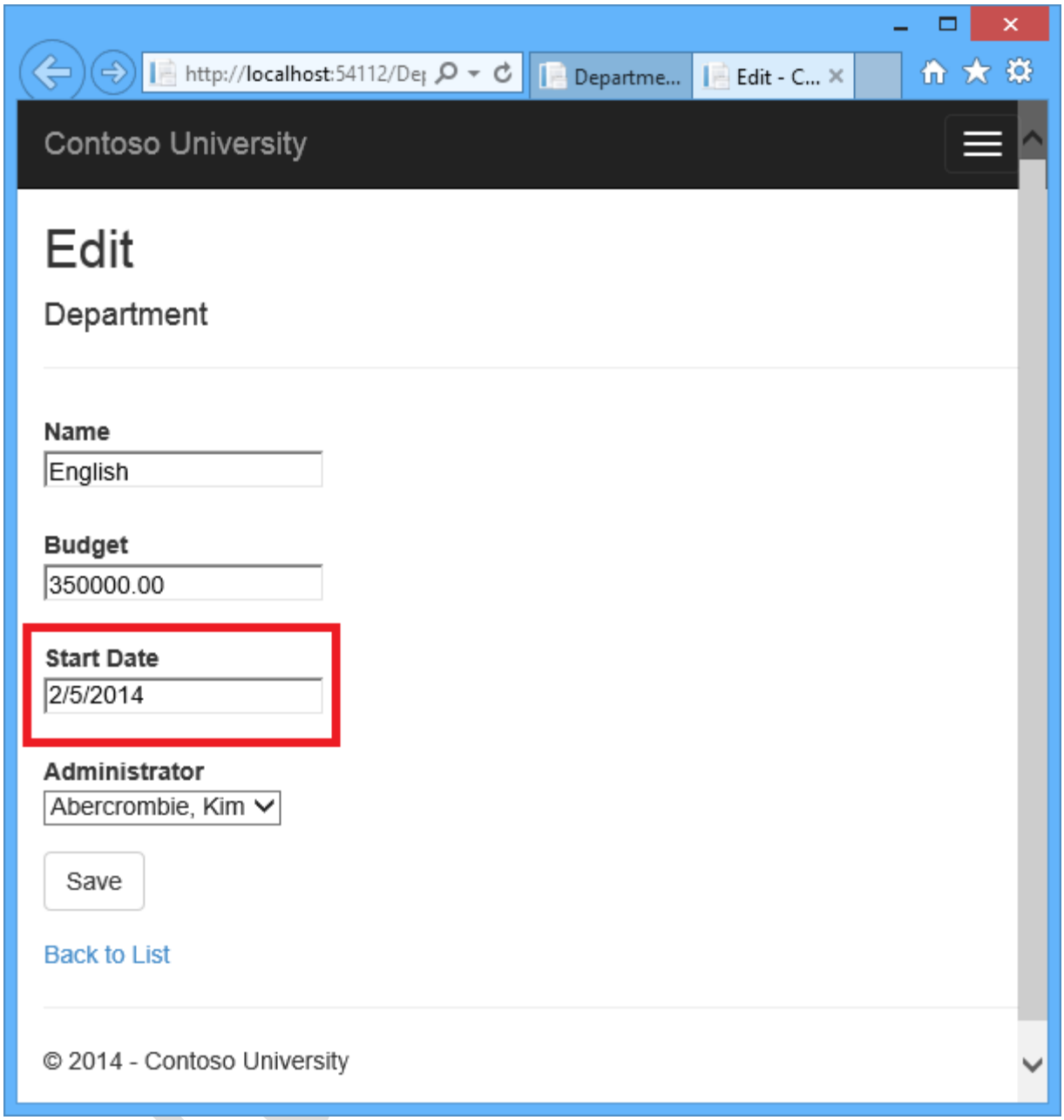
مقدار فیلد مربوطه را در تب اول مرورگر ویرایش کرده و دکمه ی Save را کلیک کنید.



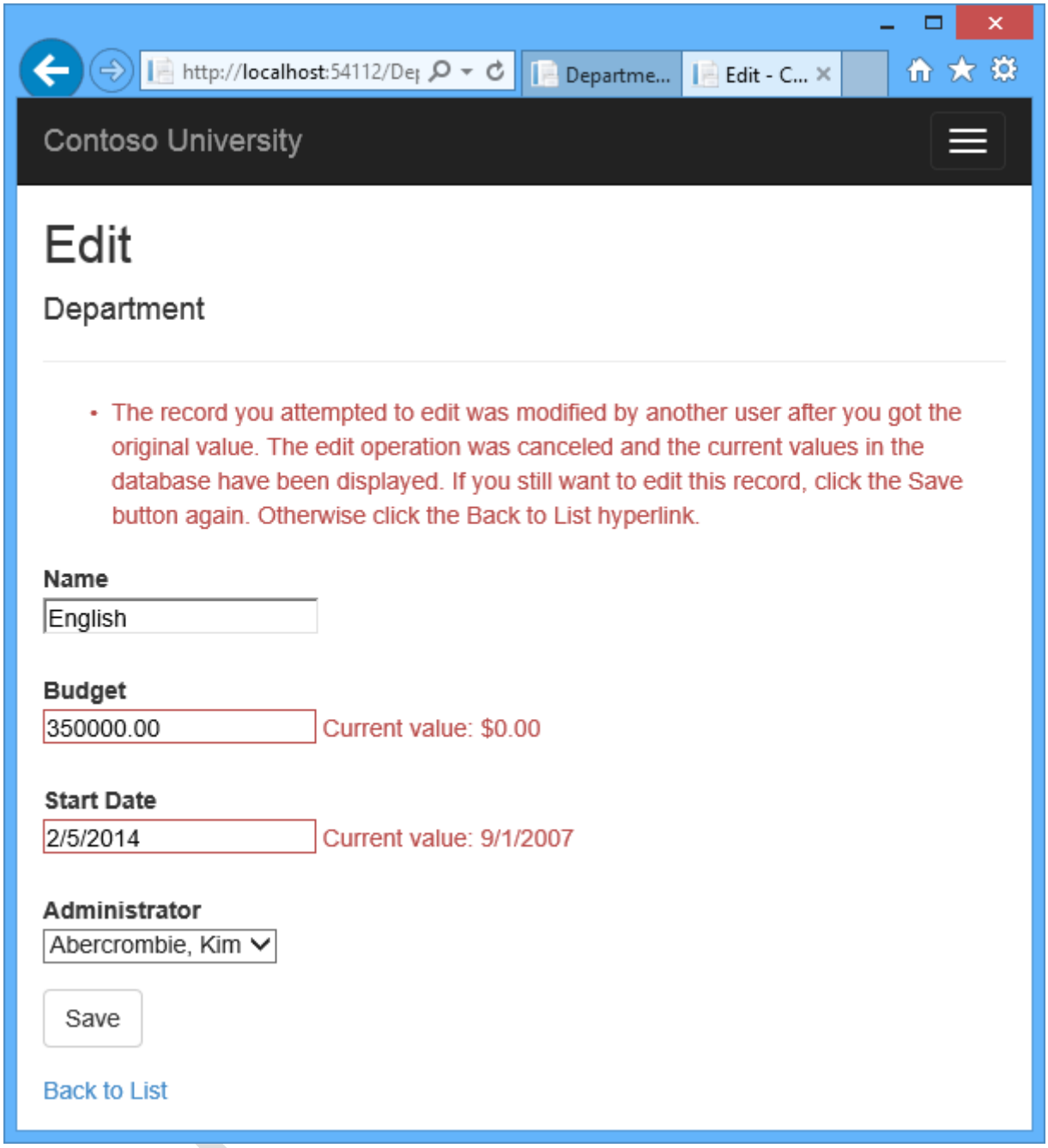
مرورگر صفحه ی Index را با مقدار اصلاح شده نمایش می دهد.



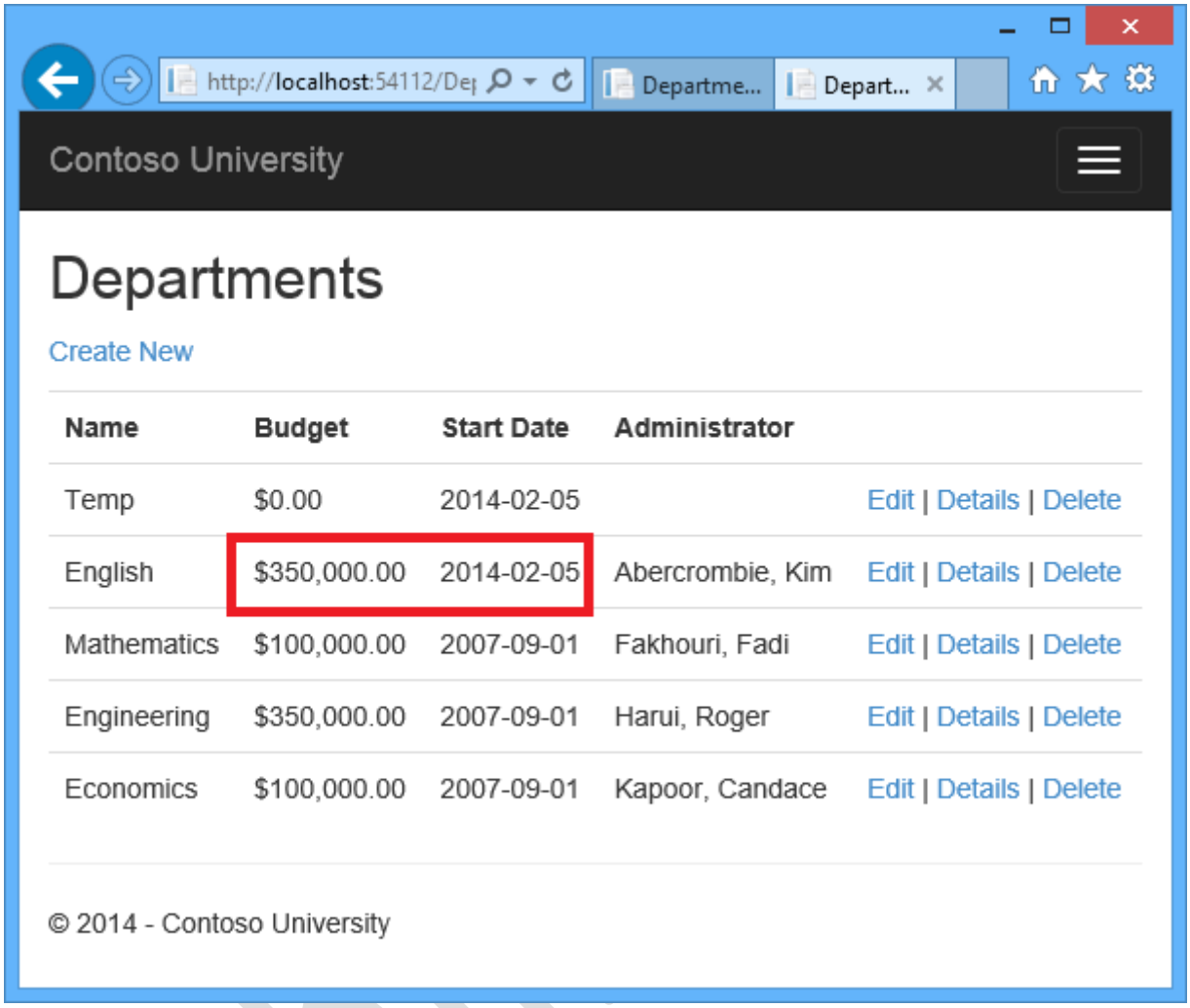
مقدار فیلد را در تب دوم مرورگر تغییر داده و دکمه ی Save را کلیک نمایید.



حال دکمه ی Save را در تب دوم پنجره ی مرورگر کلیک کنید. با یک پیغام خطا مواجه خواهید شد:



بار دیگر دکمه ی Save را کلیک کنید. مقداری که در تب دوم پنجره ی مرورگر وارد کردید همراه با مقدار اصلی داده ای که در مرورگر اول تغییر دادید ذخیره می گردد. مقادیر ذخیره شده را زمانی که صفحه ی Index ظاهر می شود، مشاهده خواهید کرد.



بروز آوری صفحه ی Delete

در مورد صفحه ی Delete، EF تداخلات همروندی ناشی از ویرایش department مورد نظر که توسط کاربر دیگر انجام می شود را به شیوه ای مشابه تشخیص می دهد. هنگامی که متد HttpGet Delete، confirmation view را نمایش می دهد، view مورد نظر مقدار اولیه ی RowVersion را در یک فیلد پنهانی قرار خواهد داد (include خواهد کرد)، سپس آن مقدار در دسترس متد HttpPost Delete قرار خواهد گرفت. این متد زمانی صدا زده می شود که کاربر حذف را قطعی (confirm) می کند. هنگامی که EF دستور DELETE را ایجاد می کند، همراه با مقدار اولیه یا اصلی RowVersion یک عبارت WHERE اضافه می نماید. اگر دستور صادر شده هیچ سطری را تحت تاثیر قرار ندهد (بدین معنا که سطر مورد نظر پس از نمایش صفحه ی تایید Delete تغییر داده شد)، در آن صورت خطای همروندی (concurrency exception) رخ می

دهد و در پی آن HttpGet Delete به همراه یک error flag که بر روی true تنظیم شده، فراخوانده می شود تا از این طریق صفحه ی تایید (confirmation page) با یک پیغام خطا نمایش داده شود. ممکن است به این خاطر هیچ سطری تغییر یافته باشد که سطر مورد نظر قبلا توسط کاربر دیگر حذف گردیده. در این صورت یک پیغام خطای متفاوت ظاهر می شود.

در فایل DepartmentController.cs، کد زیر را جایگزین متد HttpGet Delete نمایید:

```
public async Task<ActionResult> Delete(int? id, bool? concurrencyError)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  Department department = await db.Departments.FindAsync(id);
  if (department == null)
  {
    if (concurrencyError.GetValueOrDefault())
    {
      return RedirectToAction("Index");
    }
    return HttpNotFound();
  }

  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.";
  }

  return View(department);
}
```

این متد یک پارامتر اختیاری می پذیرد. پارامتر ذکر شده نشانگر این است که آیا صفحه ی مورد نظر پس از رخداد خطای همروندی نمایش داده می شود یا خیر. اگر این flag روی true تنظیم شده، در آن صورت یک پیام خطا از طریق خاصیت ViewBag به view فرستاده می شود.

کد موجود در متد HttpPost Delete را که DeleteConfirmed نام گذاری شده با کد زیر جایگزین کنید:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Delete(Department department)
{
  try
  {
    db.Entry(department).State = EntityState.Deleted;
    await db.SaveChangesAsync();
    return RedirectToAction("Index");
  }
  catch (DbUpdateConcurrencyException)
  {
    return RedirectToAction("Delete", new { concurrencyError = true, id=department.DepartmentID });
  }
  catch (DataException /* dex */)
  {
    //Log the error (uncomment dex variable name after DataException and add a line here to write a log.
    ModelState.AddModelError(string.Empty, "Unable to delete. Try again, and if the problem persists contact your
system administrator.");
    return View(department);
  }
}
```

در کدی که توسط scaffolding ایجاد شده و شما هم اکنون جایگزین کردید، این متد فقط یک record ID به عنوان پارامتر پذیرفته:

```
public async Task<ActionResult> DeleteConfirmed(int id)
```

این پارامتر را به یک نمونه موجودیت (entity instance) Department که توسط model binder ایجاد شده، تغییر دادید. کدی که توسط scaffolding ایجاد شده متد HttpPost Delete را DeleteConfirmed نام گذاری کرده تا بتواند به متد HttpPost یک signature (ورودی) منحصر بفرد بدهد. (همان طور که می دانید CLR ایجاب می کند که امضا یا پارامترهای ورودی متدهای overload شده از یکدیگر متفاوت باشد.) حال که ورودی های متدها منحصر بفرد هستند، می توانید با قوانین MVC پیش رفته و از اسمی یکسان برای متدهای delete HttpGet و HttpPost استفاده کنید.

اگر یک خطای همروندی توسط قطعه کد catch گرفته شود، کد مورد نظر صفحه ی تایید Delete را مجدد نمایش داده و یک flag ارائه می دهد. این flag بیانگر این است که بایستی یک پیام خطا همروندی نمایش داده شود.

در فایل Views\Department\Delete.cshtml، کد ارائه شده توسط scaffolding را با کد زیر جایگزین کنید. این کد یک فیلد پیام خطا به همراه فیلدهای پنهان برای خاصیت های RowVersion و DepartmentID اضافه می کند. این تغییرات هایلایت شده اند:

```
@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>
<div>
  <h4>Department</h4>
  <hr />
  <dl class="dl-horizontal">
    <dt>
      Administrator
    </dt>

    <dd>
      @Html.DisplayFor(model => model.Administrator.FullName)
    </dd>

    <dt>
      @Html.DisplayNameFor(model => model.Name)
    </dt>

    <dd>
      @Html.DisplayFor(model => model.Name)
    </dd>

    <dt>
      @Html.DisplayNameFor(model => model.Budget)
    </dt>

    <dd>
      @Html.DisplayFor(model => model.Budget)
    </dd>

    <dt>
      @Html.DisplayNameFor(model => model.StartDate)
    </dt>

    <dd>
```

```
        @Html.DisplayFor(model => model.StartDate)
    </dd>

</dl>

@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.HiddenFor(model => model.DepartmentID)
    @Html.HiddenFor(model => model.RowVersion)

    <div class="form-actions no-color">
        <input type="submit" value="Delete" class="btn btn-default" /> |
        @Html.ActionLink("Back to List", "Index")
    </div>
}
</div>
```

کد حاضر یک پیام خطا بین heading های h2 و h3 نمایش می دهد:

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

FullName را در فیلد Administrator جایگزین LastName می کند:

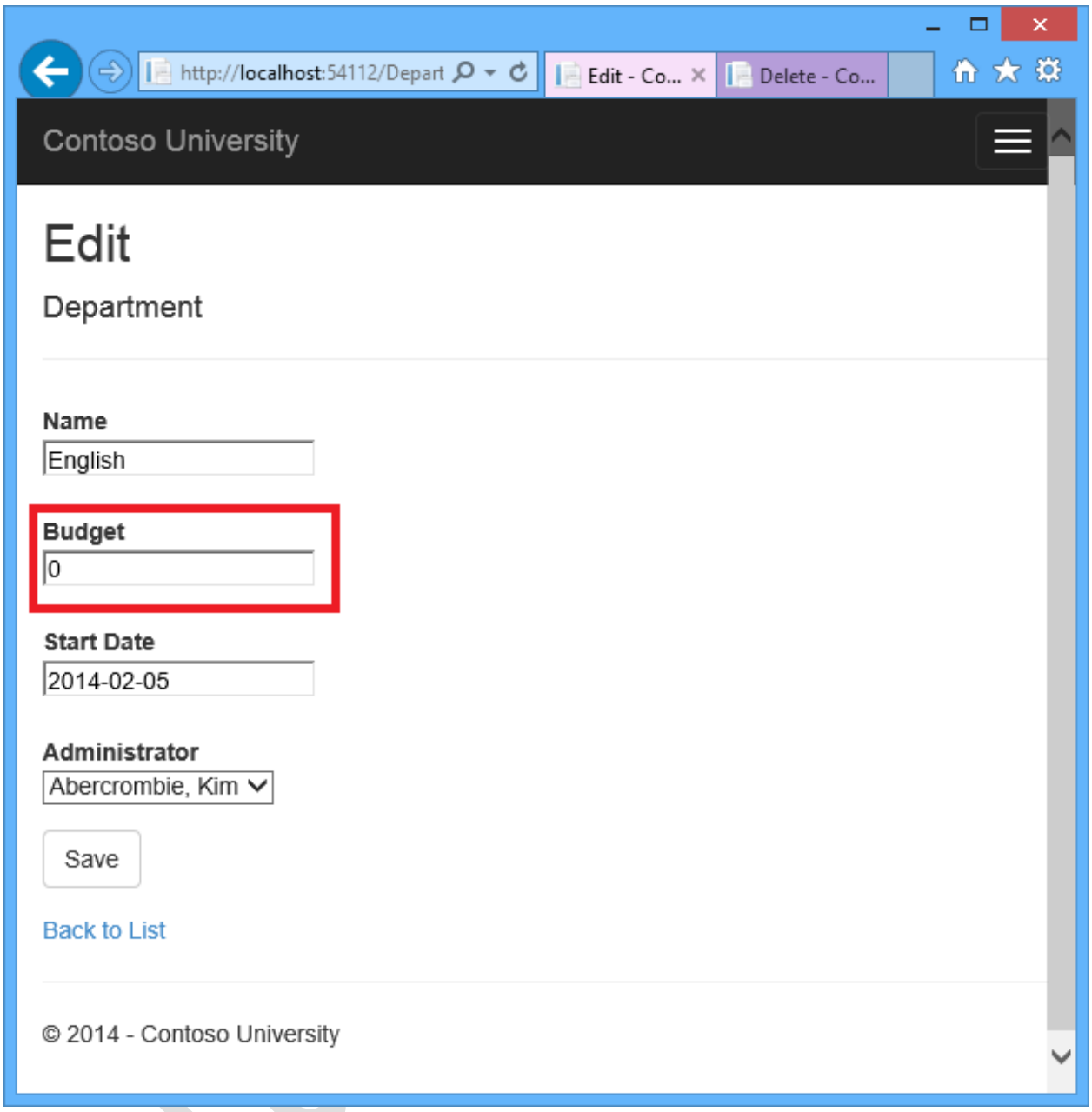
```
<dt>
 Administrator
</dt>
<dd>
 @Html.DisplayFor(model => model.Administrator.FullName)
</dd>
```

سرانجام فیلدهای پنهان را برای خواص DepartmentID و RowVersion پس از دستور Html.BeginForm اضافه می کند:

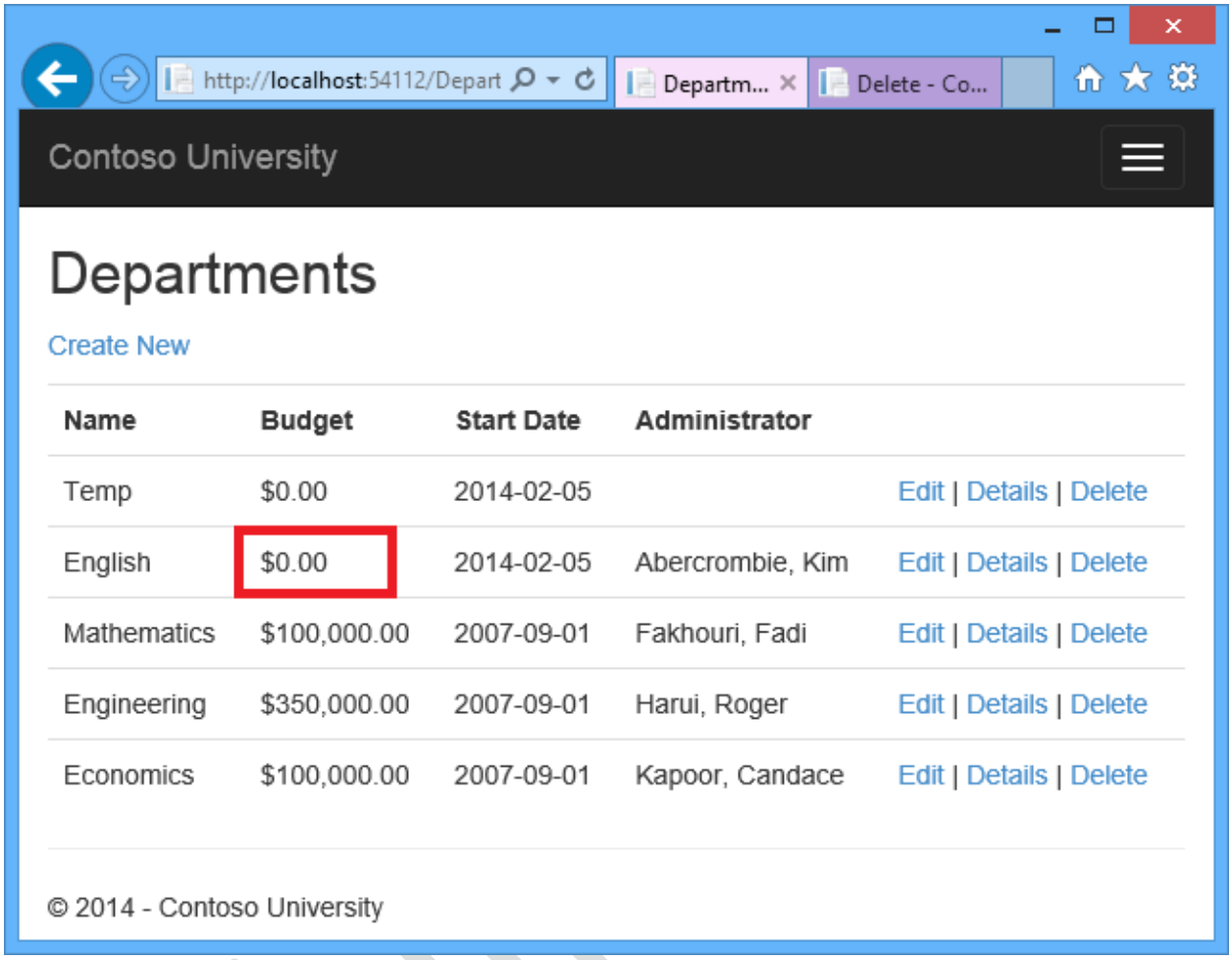
```
@Html.HiddenFor(model => model.DepartmentID)
@Html.HiddenFor(model => model.RowVersion)
```

صفحه ی Departments Index را اجرا کنید. بر روی لینک Delete فیلد English راست کلیک کرده و Open in new tab را انتخاب نمایید.

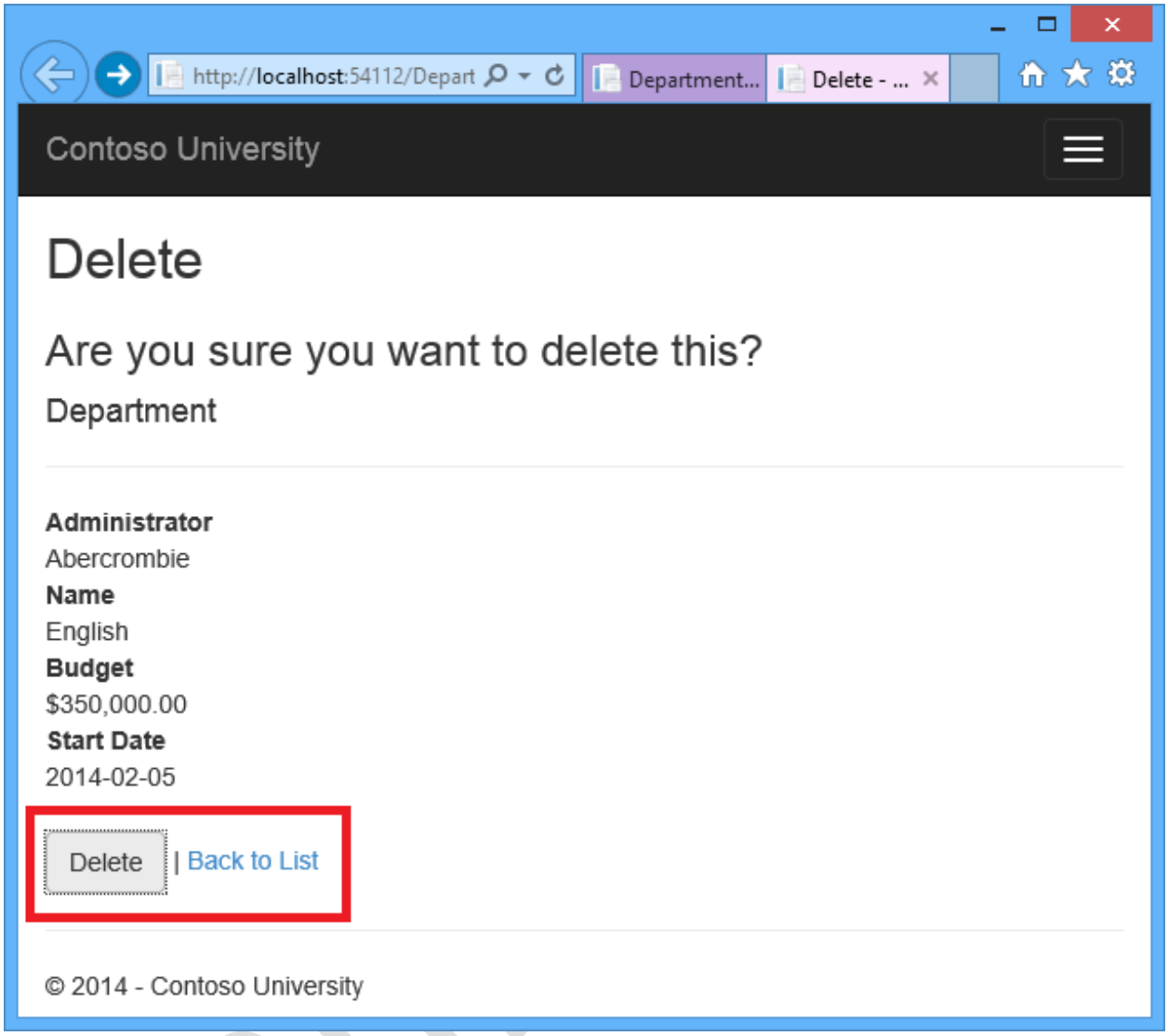
در پنجره ی اول، یکی از مقادیر را ویرایش کرده و Save را کلیک کنید:



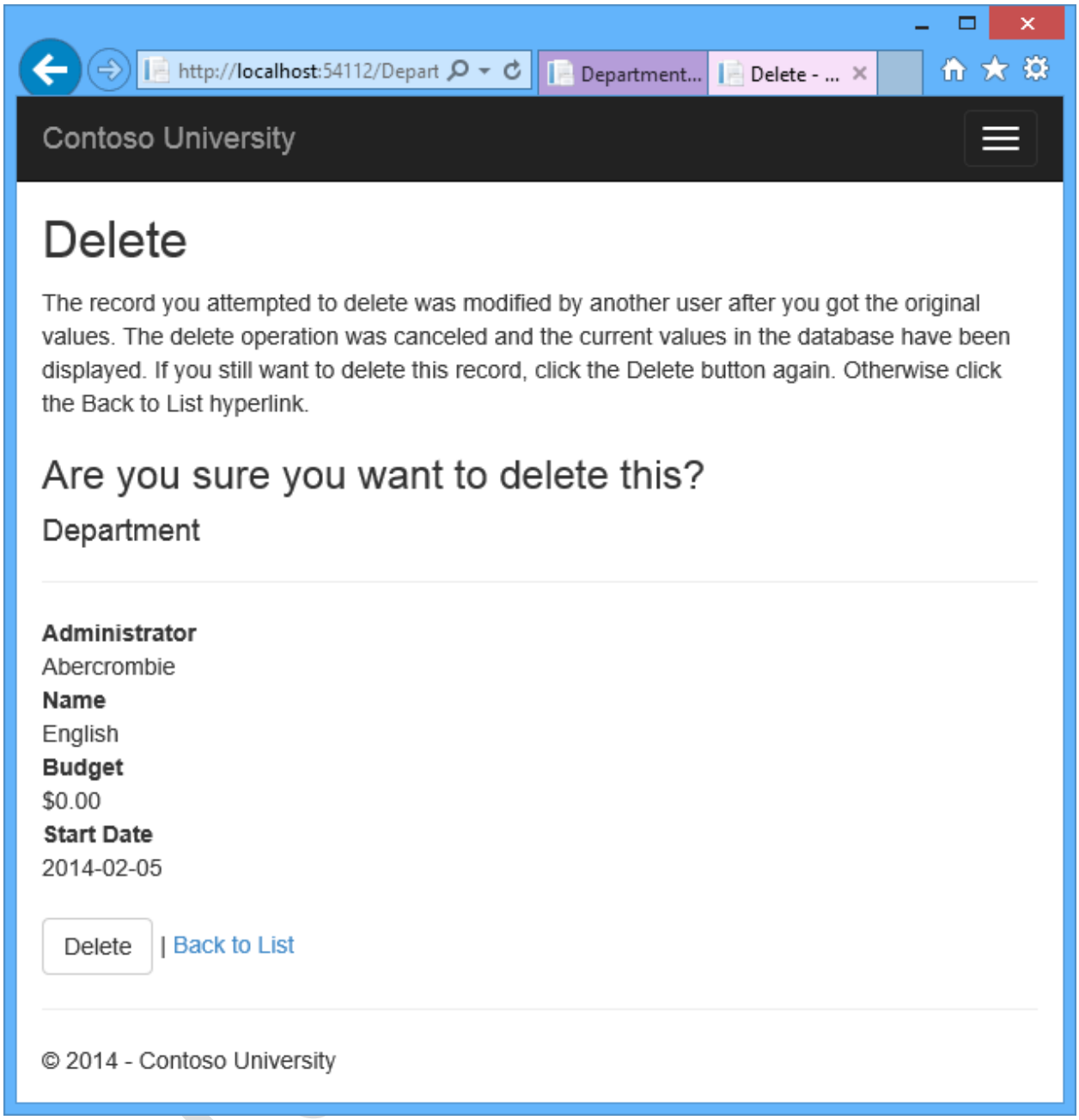
صفحه ی Index تغییر را confirm (تایید قطعی) می کند.



در تب دوم، دکمه ی Delete را کلیک کنید.



پیام خطای همروندی و مقادیر Departments را مشاهده می کنید که با مقادیر موجود در پایگاه داده بروز رسانی شده است.



اکنون اگر بر روی Delete کلیک کنید، به صفحه ی Index هدایت می شوید. در این صفحه خواهید دید که department مورد نظر پاک شده است.