بسم الله الرحمن الرحيم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

ساخت data model پیچیده تر برای برنامه ی تحت وب ASP.NET MVC با EF6

مدرس : مهندس افشین رفوآ

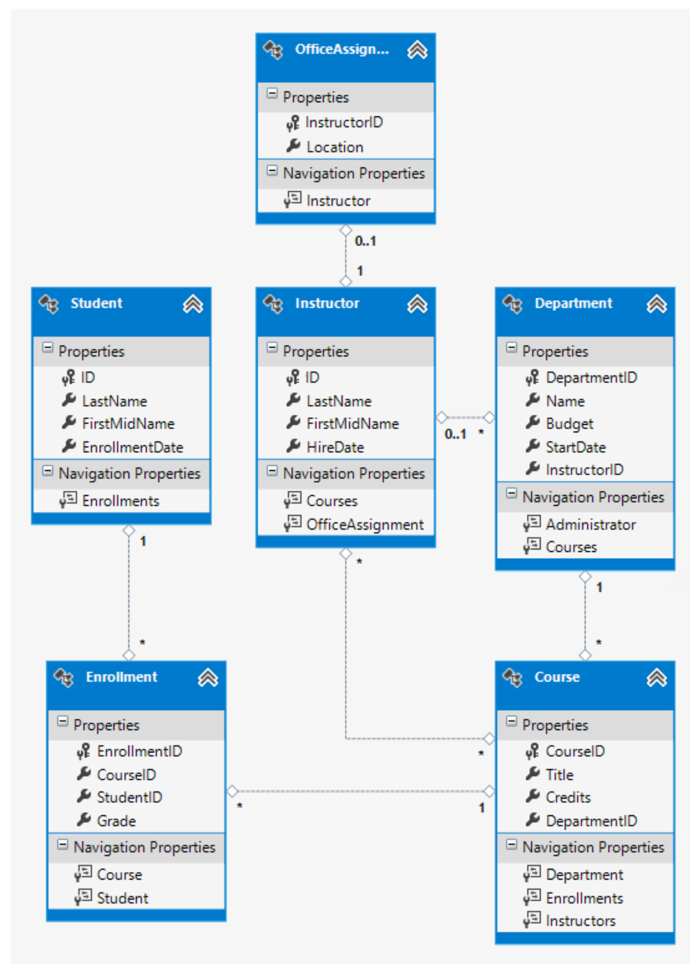
دوره آموزش MVC

ساخت data model پیچیده تر برای برنامه ی تحت وب ASP.NET MVC با EF6

در مباحث پیشین، با data model ساده کار می کردیم که در کل از سه بخش تشکیل می شد. در این درس موجودیت ها (entity) و روابط (relationship) پیچیده تری اضافه کرده و با تعریف اصول و قوانین قالب بندی – formatting، اعتبارسنجی و نگاشت پایگاه داده (database mapping)، data model خود را اختصاصی تنظیم می کنیم. در درس حاضر با دو روش data model را تنظیم می کنیم: 1. با افزودن خصیصه هایی (attribute) به entity class ها 2. با اضافه کردن کدهایی به کلاس database context.

پس از اجرای تمامی مراحل، کلاس های entity، data model کامل زیر را تشکیل خواهد داد:

Tahlildadeh



روش اول (تنظیم data model با استفاده از خصیصه ها)

در این بخش به وسیله ی خصیصه هایی که قوانین قالب بندی، اعتبارسنجی و نگاشت پایگاه داده را تعریف می کند، data model خود را تنظیم می کنیم. سپس با افزودن خصیصه هایی به کلاس هایی که قبلا ایجاد کردیم و همچنین ایجاد کلاس های جدید برای دیگر entity type ها در model، School data model را به طور کامل می سازیم.

خصیصه ی DataType

تمامی صفحات وب برنامه ی ما برای تاریخ ثبت نام دانشجویان (student enrollment dates)، در حال حاضر همراه با تاریخ زمان را نیز نمایش می دهد، اما آنچه در این فیلد برای ما اهمیت دارد، فقط تاریخ است. با استفاده از خصیصه های فضای نام data annotation، می توانید کاری کنید که با تغییر یک کد، فرمت نمایش (display format) در هر view ای که آن داده را به نمایش می گذارد، تنظیم شود. برای این که یاد بگیرید چگونه این کار را انجام دهید، کافی است یک خصیصه (attribute) به EnrollmentDate property در کلاس Student اضافه کنید.

در فایل Models\Student.cs، یک دستور using در فضای نام System.ComponentModel.DataAnnotations و به دنبال آن خصیصه های DataType و DisplayFormat را به EnrollmentDate property اضافه کنید، مانند نمونه ی زیر:

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

namespace ContosoUniversity.Models
{
  public class Student
  {
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
    public DateTime EnrollmentDate { get; set; }

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

خصیصه ی DataType به منظور تعریف یک نوع داده ای بکار می رود که از نوع ذاتی (intrinsic type) پایگاه داده خاص تر یا دقیق تر باشد. برای این پروژه در حال حاضر فقط فیلد date را در نظر می گیریم. نوع داده ی شمارشی DataType Enumeration مقادیر مختلفی همچون Date، Time، PhoneNumber، Currency، EmailAdress و غیره .. را ارائه می دهد. خصیصه ی DataType همچنین برنامه را قادر می سازد تا امکاناتی ویژه ی هر نوع خاص فراهم نماید. به عنوان مثال، با بهره گیری از خصیصه ی ذکر شده می توان لینک :mailto را برای DataType.EmailAddress ایجاد کرد و یک date selector (انتخاب گر تاریخ) ویژه ی

DataType.Date در مرورگرهایی که از زبان نشانه گذاری HTML5 پشتیبانی می کنند، فراهم نمود.

خصیصه های DataType، attribute های -data که توسط HTML5 پشتیبانی می شود، ارائه می دهد که این خصیصه ها برای مرورگر قابل فهم می باشند. گفتنی است این خصیصه ها هیچ گونه عملیات اعتبارسنجی بر روی ورودی ها اجرا نمی کنند.

DataType.Date فرمت نمایش تاریخ را مشخص نمی کند. به طور پیش فرض، فیلد داده ها با توجه به فرمت پیش فرض و بر اساس CultureInfo سرویس دهنده نمایش داده می شود.

جهت تعیین فرمت نمایش تاریخ می توان از خصیصه ی DisplayFormat بهره جست:

```
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

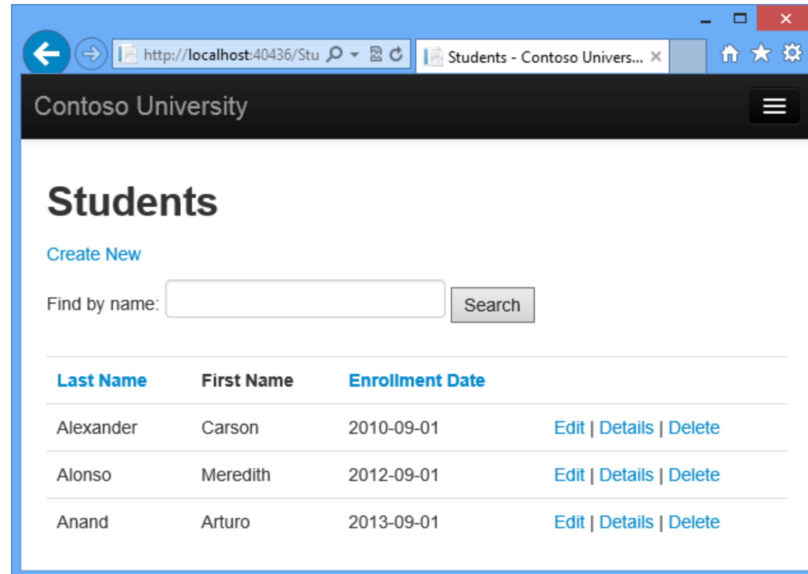
تنظیم خصیصه ی ApplyFormatInEditMode تعیین می کند که قالب بندی یا formatting مشخص شده بایستی زمانی که مقدار در textbox برای ویرایش نمایش داده می شود نیز اعمال شود. البته این کار را برای تمامی فیلدها نباید انجام داد، به عنوان مثال می توان از مقادیر مربوط به ارز یا currency نام برد. مطمئنا شما نمی خواهید که علامت ارز در کادر متن ویرایش شود.

می توانید خصیصه ی DisplayFormat را به تنهایی بکار برد، اما بد نیست آن را همراه با خصیصه ی DataType نیز استفاده کرد. خصیصه ی DataType بجای اینکه نحوه ی نمایش و render داده بر روی صفحه نمایش را تعیین کند، منطق (semantics) داده را می رساند و علاوه بر آن مزایای زیر را که در صورت استفاده از DisplayFormat از آن ها محروم خواهید بود، فراهم می نماید:

1. مرورگر قادر خواهد بود امکانات HTML5 را فعال ساخته و از آن ها استفاده کند (برای مثال می توان به کنترل تقویم، علامت ارز مربوط به آن کشور، لینک های ایمیل و اعتبارسنجی ورودی ها در سمت سرویس گیرنده اشاره نمود.)
2. به صورت پیش فرض، مرورگر با توجه به کشور محل زندگی شما فرمت مناسب را انتخاب کرده و داده ها را بر اساس آن نمایش می دهد (render).
3. خصیصه ی DataType به MVC این قابلیت را می دهد که field template مناسب را برای نمایش (render) داده ها گزینش کند (خصیصه ی DisplayFormat الگوهای رشته ای یا string template را انتخاب می نماید).

در صورت استفاده از خصیصه ی DataType برای یک فیلد داده، می بایست خصیصه ی DisplayFormat را همراه با آن بکار برد تا مطمئن شویم فیلد مد نظر در مرورگر Chrome نیز به درستی نمایش داده شود.

صفحه ی Student Index را بار دیگر اجرا کنید. خواهید دید که دیگر زمان برای تاریخ ثبت نام (enrollment dates) نمایش داده نمی شود. این اتفاق برای تمامی view هایی که از Student model استفاده می کنند، رخ می دهد.



خصیصه های StringLength

می توان به کمک attribute ها، قوانین اعتبارسنجی داده و نیز پیام های خطای اعتبارسنجی تعریف کرد. خصیصه ی StringLength حداکثر تعداد مجاز کاراکترها در پایگاه داده را تنظیم می کند، همچنین اعتبارسنجی در سمت سرویس گیرنده و سرویس دهنده را برای برنامه های ASP.NET MVC انجام می دهد. شما می توانید حداقل تعداد کاراکترهای مجاز را در این خصیصه مشخص کنید، اما لازم است بدانید که حداقل مقدار هیچ تاثیری بر روی database schema ندارد.

فرض بگیرید می خواهیم یک محدودیت تعیین کنیم که در آن کاربر نتواند بیش از 50 کاراکتر در فیلد name وارد کند. به منظور اعمال این قید، کافی است خصیصه ی StringLength را به property های LastName و FirstMidName الحاق کنید:

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

namespace ContosoUniversity.Models
{
  public class Student
  {
    public int ID { get; set; }
    [StringLength(50)]
    public string LastName { get; set; }
    [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
    public string FirstMidName { get; set; }
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
    public DateTime EnrollmentDate { get; set; }

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

خصیصه ی StringLength مانع از آن نمی شود که کاربر فضای خالی وارد کند. می توانید با استفاده از خصیصه ی RegularExpression، محدودیت هایی را به ورودی کاربر اعمال کنید. به عنوان مثال، کد زیر ایجاب می کند که کاراکتر ورودی اول با حرف بزرگ نوشته شود و باقی کاراکترها از نوع الفبا باشد:

```
[RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]
```

خصیصه ی MaxLength نیز تقریبا کاربردی مشابه خصیصه ی StringLength دارد با این تفاوت که در سمت سرویس گیرنده اعتبارسنجی انجام نمی دهد.

برنامه را اجرا کرده، سپس بر روی تب Students کلیک نمایید. پیام خطای زیر نمایش داده می شود:

The model backing the 'SchoolContext' context has changed since the database was " created. Consider using Code First Migrations to update the database "(http://go.microsoft.com/fwlink/?LinkId=238269).

"model ای که 'SchoolContext' را پشتیبانی می کرد از زمانی که پایگاه داده برای اولین بار ایجاد شد، تغییر یافته. توصیه می کنیم با استفاده از Code First Migration پایگاه داده ی مورد نظر را بروزآوری کنید. "

Model پایگاه داده به گونه ای تغییر یافته که لازمه ی ایجاد یک تغییر در database schema می باشد. EF بلافاصله این نیاز را شناسایی می کند. با استفاده از Migrations شمای پایگاه داده (database schema) را گونه ای بروز رسانی می کنیم که هیچ یک از داده هایی که از طریق UI به پایگاه داده ی مربوطه افزوده شده، از دست نرود. اگر داده هایی را که توسط متد Seed ایجاد شده، تغییر دهید، خواهید دید که به حالت اولیه باز می گردد زیرا که از متد AddOrUpdate در متد Seed استفاده شده است. (متد AddOrUpdate معادل همان عملیات upsert در پایگاه داده می باشد.)

در پنجره ی Package Manager Console، دستورات زیر درج کنید:

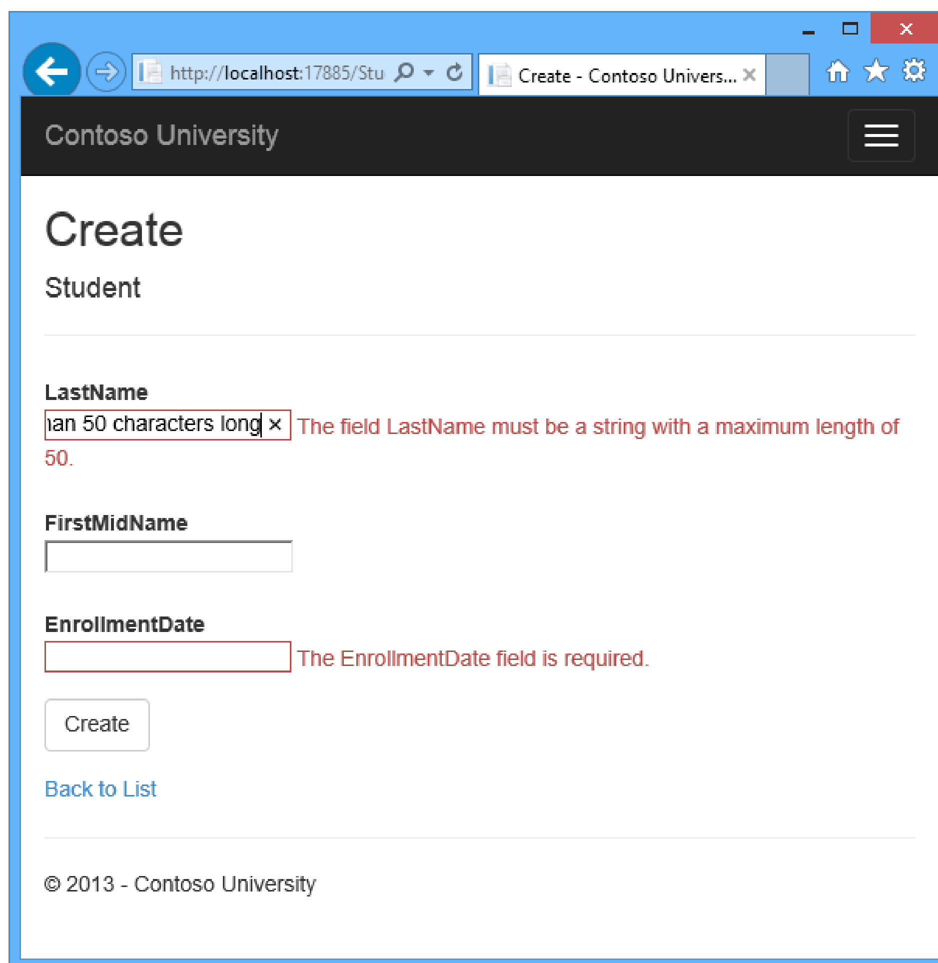
```
add-migration MaxLengthOnNames
update-database
```

دستور add-migration یک فایل به نام <timeStamp>_MaxLengthOnNames.cs ایجاد می کند. این فایل حاوی کدی در متد Up است که پایگاه داده را برای منطبق شدن و همخوانی با data model جاری بروز آوری می کند. دستور update-database آن کد را اجرا می کند.

Timestamp ای که به پیش از اسم فایل migrations الصاق شده، توسط EF برای مرتب سازی migrations (انتقال تغییرات به پایگاه داده) بکار گرفته می شود. می توان چندین migration، قبل از اجرای دستور update-database ایجاد کرد، سپس تمامی migration ها دقیقا به همان ترتیبی که ایجاد شدند، اعمال می گردند.

صفحه ی Create را اجرا کرده، سپس هر دو اسم را بیش از 50 کاراکتر در فیلد مربوط وارد نمایید. پس از کلیک بر روی Create، خواهید دید که اعتبارسنجی که در سمت سرویس گیرنده اجرا می شود، یک پیغام خطا به نمایش می گذارد.

Tahlildadeh



خصیصه ی Column

همچنین می توانید خصیصه هایی را مورد استفاده قرار دهید که نحوه ی نگاشت (map) کلاس ها و خاصیت ها (property) به پایگاه داده را مدیریت کند. فرض بگیرید اسم FirstMidName را برای فیلد first-name بکار برده بودید، زیرا احتمال دارد کاربر در فیلد اسم دوم خود را وارد کند. اما شما می خواهید که ستون پایگاه داده FirstName نامیده شود، زیرا کاربرانی که query های ویژه ای را برای اجرا بر روی پایگاه داده ی مورد نظر می نویسند، به آن اسم عادت دارند. جهت اجرای این mapping (نگاشت خاصیت ها و کلاس ها به پایگاه داده)، کافی است از خصیصه ی Column استفاده کنید.

خصیصه ی Column مشخص می کند که هنگامی که پایگاه داده ایجاد شد، آن ستونی که از جدول Student به خاصیت FirstMidName نگاشت می گردد، FirstName نام گذاری شود. به عبارتی دیگر، هنگامی که کد شما به Student.FirstMidName اشاره می کند، معنیش این است که داده ها از

ستون FirstName گرفته می شوند یا در ستون FirstName متعلق به جدول Student بروز رسانی می گردند. در صورت مشخص نکردن اسم ستون، همان اسم خاصیت به عنوان اسم ستون بکار می رود.

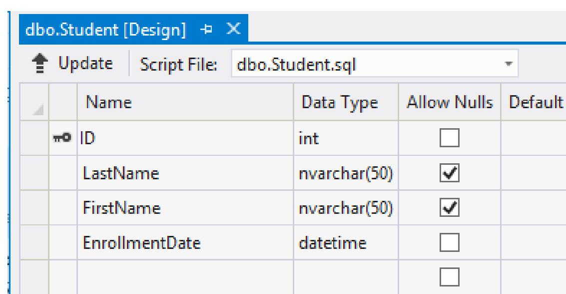
فایل Student.cs را باز کرده، یک دستور using به فضای نام System.ComponentModel.DataAnnotations.Schema اضافه کنید، سپس خصیصه ی column name را به خاصیت FirstMidName اضافه کنید، به صورتی که در کدهای هایلایت شده ی زیر نمایش داده شده است:

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

namespace ContosoUniversity.Models
{
  public class Student
  {
    public int ID { get; set; }
    [StringLength(50)]
    public string LastName { get; set; }
    [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
    [Column("FirstName")]
    public string FirstMidName { get; set; }
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

    public DateTime EnrollmentDate { get; set; }

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

افزودن خصیصه ی Column، سبب می شود آن Model ای که پشتیبان SchoolContext است تغییر کند، پس از این رویداد، model دیگر با پایگاه داده منطبق و همخوان نخواهد بود. حال دستورات زیر را در پنجره ی PMC وارد کرده تا migration دیگری ایجاد شود:

```
add-migration ColumnFirstName
update-database
```

در پنجره ی Server Explorer، با دوبار کلیک بر روی جدول Student، Student table designer را باز کنید:

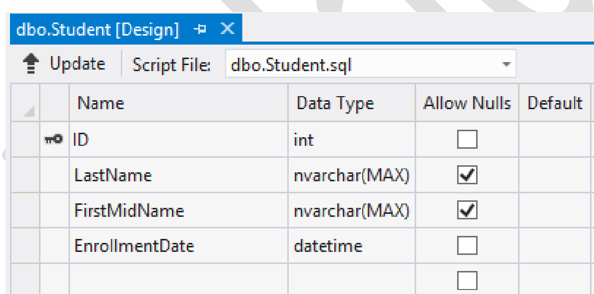


dbo.Student [Design]

Update | Script File: dbo.Student.sql

Name	Data Type	Allow Nulls	Default
ID	int	☐	
LastName	nvarchar(50)	☑	
FirstName	nvarchar(50)	☑	
EnrollmentDate	datetime	☐	
		☐	

تصویر زیر، اسم ستون را همان گونه ای که قبل از اعمال دو migration اول بود، نمایش می دهد. همان طور که می بینید، علاوه بر اسم ستون که از FirstMidName به FirstName تغییر یافته، دو ستون های Name نیز از حداکثر تعداد کاراکتر مجاز MAX به 50 تغییر کرده.



dbo.Student [Design]

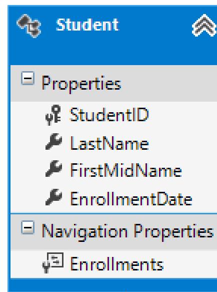
Update | Script File: dbo.Student.sql

Name	Data Type	Allow Nulls	Default
ID	int	☐	
LastName	nvarchar(MAX)	☑	
FirstMidName	nvarchar(MAX)	☑	
EnrollmentDate	datetime	☐	
		☐	

می توان با استفاده از رابط برنامه سازی کاربردی (API) Fluent، تغییراتی را به database mapping اعمال کنید.

نکته: چنانچه قبل از ایجاد کردن تمامی کلاس های entity سعی کنید کامپایل را انجام دهید، ممکن است با خطاهای compiler مواجه شوید.

تکمیل اعمال تغییرات به موجودیت Student



در فایل Models\Student.cs ، کدهای هایلایت شده ی زیر را جایگزین کدی که قبلا اضافه کرده بودید، کنید:

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

namespace ContosoUniversity.Models
{
  public class Student
  {
    public int ID { get; set; }
    [Required]
    [StringLength(50)]
    [Display(Name = "Last Name")]
    public string LastName { get; set; }
    [Required]
    [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]
    [Column("FirstName")]
    [Display(Name = "First Name")]
    public string FirstMidName { get; set; }
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
    [Display(Name = "Enrollment Date")]
    public DateTime EnrollmentDate { get; set; }

    [Display(Name = "Full Name")]
    public string FullName
    {
      get
      {
        return LastName + ", " + FirstMidName;
      }
    }

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

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 165 واحد7
88146323 - 88446780 - 88146330

خصیصه ی Required

Required یک خصیصه یا attribute تلقی می‌شود که name property ها را به فیلدهای الزامی (فیلدهایی که باید مقداری داخل آن ها وارد شود) تبدیل می کند. استفاده از این خصیصه برای value type هایی نظیر DateTime، int، double و float ضرورتی ندارد. نمی توان به value type ها مقدار null تخصیص داد، از این رو (این فیلدها به صورت ذاتی الزامی محسوب می‌شوند) با آن ها به عنوان فیلدهای الزامی برخورد می شود. می توان خصیصه ی Required را حذف کرده و آن را با یک پارامتر minimum length (مشخص کننده حداقل تعداد کاراکتر مجاز) در خصیصه ی StringLength جایگزین کرد:

```
[Display(Name = "Last Name")]
    [StringLength(50, MinimumLength = 1)]
    public string LastName { get; set; }
```

خصیصه ی Display

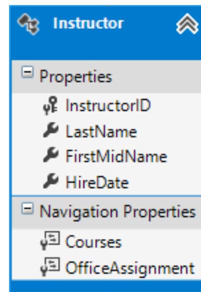
خصیصه ی Display مشخص می کند که عنوان کادر متن (textbox caption) در هر نمونه بجای اسم property (که هیچ فضایی که دو واژه را از هم جدا کند، در آن وجود ندارد)، به ترتیب "First Name"، "Last Name"، "Full Name" و "Enrollment Date" باشد.

Property محاسباتی FullName

یک خاصیت یا property است که مقداری را بازیابی می کند که از اتصال یا ادغام دو خاصیت دیگر حاصل می شود. از این رو خاصیت مزبور تنها از یک get accessor برخوردار می باشد. همچنین هیچ ستون FullName ای در پایگاه داده ایجاد نمی شود.

ساخت موجودیت Instructor

Tahlildadeh



فایل Models\Instructor.cs را ایجاد کرده، کد زیر را جایگزین کد template کنید:

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

namespace ContosoUniversity.Models
{
  public class Instructor
  {
    public int ID { get; set; }

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

    [Required]
    [Column("FirstName")]
    [Display(Name = "First Name")]
    [StringLength(50)]
    public string FirstMidName { get; set; }

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

    [Display(Name = "Full Name")]
    public string FullName
    {
      get { return LastName + ", " + FirstMidName; }
    }

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

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 165 واحد7
88146323 - 88446780 - 88146330

همان طور که مشاهده می کنید، تعداد زیادی از property های موجود در entity های Student و Instructor، یکی هستند.

می توان چندین attribute را در یک خط واحد جای گذاری کرد. از این رو می توان کلاس Instructor را به صورت زیر نیز نوشت:

```
public class Instructor
{
  public int ID { get; set; }

  [Display(Name = "Last Name"),StringLength(50, MinimumLength=1)]
  public string LastName { get; set; }

  [Column("FirstName"),Display(Name = "First Name"),StringLength(50, MinimumLength=1)]
  public string FirstMidName { get; set; }

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

  [Display(Name = "Full Name")]
  public string FullName
  {
    get { return LastName + ", " + FirstMidName; }
  }

  public virtual ICollection&lt;Course> Courses { get; set; }
  public virtual OfficeAssignment OfficeAssignment { get; set; }
}



}
```

Navigation Properties های (خاصیت های پیمایشی) Courses و OfficeAssignment

خاصیت های Courses و OfficeAssignment، property هایی هستند که برای پیمایش مورد استفاده قرار می گیرند. همان طور که پیش تر شرح داده شد، خاصیت های نام برده اغلب به صورت virtual تعریف می شوند تا بدین وسیله از یک امکان ویژه ی EF به نام Lazy Loading یا بارگذاری با تاخیر بهره ببرند.

علاوه بر آن، اگر یک navigation property قادر است همزمان چندین entity داشته باشد، در آن صورت type آن بایستی رابط (interface) ICollection<T> را پیاده سازی کند. برای مثال، IList<T> شرایط

Tahlildadeh

لازم را دارد در حالی که رابط IEnumerable<T> برای این منظور به هیچ وجه مناسب نمی باشد، زیرا رابط IEnumerable<T> متد Add را پیاده سازی نمی کند.

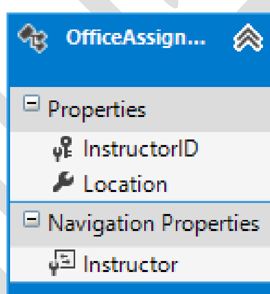
یک استاد (instructor) قادر است تعداد زیادی دوره ی آموزشی (courses) را تدریس کند، بنابراین Courses به عنوان یک مجموعه ای از موجودیت های Course تعریف می شود.

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

قوانین کاری حاکم بر دانشگاه ایجاب می کند که بیش از یک دفتر به یک استاد، تخصیص نیابد، از این رو OfficeAssignment به عنوان یک موجودیت واحد OfficeAssignment تعریف می شود (که در صورت تخصیص نیافتن دفتر به استاد، ممکن است مقدار null باشد).

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

ایجاد یک موجودیت به نام OfficeAssignment



فایل Models\OfficeAssignment.cs را ایجاد کرده و کد زیر را در آن بنویسید:

```
    using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
  public class OfficeAssignment
  {
    [Key]
    [ForeignKey("Instructor")]
    public int InstructorID { get; set; }
    [StringLength(50)]
    [Display(Name = "Office Location")]
    public string Location { get; set; }

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

}

پروژه را بازسازی (Build) کنید، این کار باعث می شود تغییرات شما ذخیره گردیده و نیز بررسی می کند که شما هیچ خطایی در خصوص copy و paste نکرده باشید که کامپایلر بتواند ضبط (catch) کند.

خصیصه ی Key

یک رابطه ی یک به صفر یا یک به یک بین موجودیت های Instructor و OfficeAssignment وجود دارد. یک office assignment تنها زمانی می تواند وجود داشته باشد که instructor ای که به آن تخصیص یابد، از این رو primary key (کلید اصلی) آن foreign key (کلید خارجی) موجودیت Instructor نیز محسوب می شود.

اما EF این توانایی را ندارد که InstructorID را به صورت خودکار به عنوان primary key این موجودیت بشناسد، زیرا اسم آن از قانون نام گذاری ID یا classnameID پیروی نمی کند. از اینرو تنها به واسطه ی خصیصه ی Key می توان InstructorID را به عنوان کلید معرفی کرد:

```
[Key]
    [ForeignKey("Instructor")]
    public int InstructorID { get; set; }
```

البته چنانچه entity مورد نظر خود دارای primary key هست اما شما می خواهید نام property را متفاوت از ID یا classnameID انتخاب کنید، در آن صورت می توان خصیصه ی Key را مورد استفاده قرار داد. به صورت پیش فرض، EF با کلید به گونه ای برخورد می کند که گویی توسط پایگاه داده ایجاد نشده، زیرا آن ستون صرفا به منظور شناسایی رابط ها بکار می رود.

خصیصه ی ForeignKey

زمانی که یک رابطه ی یک به صفر یا یک به یک و یا رابطه ی یک به یک بین دو موجودیت حاکم است (همانند رابطه ی بین دو موجودیت OfficeAssignment و Instructor)، EF نمی تواند دریابد کدام طرف رابطه اصلی (principal) و کدام وابسته (dependent) می باشد. رابطه های یک به یک، دارای یک navigation property مرجع در هر کلاس به کلاس دیگر است. ForeignKey Attribute می تواند به کلاس وابسته اعمال

شده تا رابطه ی لازم را برقرار کند. در صورت حذف ForeignKey Attribute، به هنگام تلاش برای ایجاد یک migration، با خطای زیر روبرو خواهید شد:

Unable to determine the principal end of an association between the types 'ContosoUniversity.Models.OfficeAssignment' and 'ContosoUniversity.Models.Instructor'. The principal end of this association must be explicitly configured using either the relationship fluent API or data annotations.

" قادر به شناسایی طرف اصلی (principal end) ارتباط بین type های 'ContosoUniversity.Models.OfficeAssignment' و 'ContosoUniversity.Models.Instructor' نیست. طرف اصلی این ارتباط بایستی با استفاده از رابطه ی Fluent API یا Data Annotation به صورت صریح پیکربندی شود. "

Instructor Navigation Property

موجودیت Instructor یک navigation property به نام office assignment که nullable است (یک خاصیت برای پیمایش که می تواند خالی نیز باشد) دارد (زیرا که ممکن است به یک استاد دفتر تخصیص داده نشود یا به عبارتی یک instructor، office assignment نداشته باشد) و موجودیت OfficeAssignment دارای یک non-nullable Instructor navigation property است (زیرا یک office assignment نمی تواند بدون یک instructor وجود داشته باشد – InstructorID قابلیت خالی یا nullable بودن را ندارد). یک instructor می تواند بدون office assignment باشد، اما یک office assignment نمی تواند بدون instructor باشد.

اگر که موجودیت Instructor دارای یک موجودیت مرتبط OfficeAssignment بود، در آن صورت هر موجودیت از طریق navigation property خود به دیگری یک reference می دهد.

می توانستید یک خصیصه ی [Required] به navigation property به نام Instructor الصاق کرده و مشخص کرد که instructor مربوطه باید ایجاد شود، اما از آنجایی که کلید خارجی (foreign key) InstructorID نمی تواند خالی باشد (non-nullable است) لزومی به انجام این کار نیست.

اصلاح موجودیت Course

در فایل Models\Course.cs، کد زیر را جایگزین کدی که قبلا به این فایل اضافه کرده بودید، کنید:

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

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

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

    [Range(0, 5)]
    public int Credits { get; set; }

    public int DepartmentID { get; set; }

    public virtual Department Department { get; set; }
    public virtual ICollection&lt;Enrollment> Enrollments { get; set; }
    public virtual ICollection&lt;Instructor> Instructors { get; set; }
  }
}
```

موجودیت course دارای یک foreign key property به نام DepartmentID می باشد که به موجودیت مربوطه ی Department اشاره می کند. این موجودیت همچنین دارای یک navigation property به نام Department می باشد. اگر شما از قبل یک navigation property ویژه ی موجودیت مربوطه دارید، در آن صورت EF از شما نمی خواهد یک foreign key property به data model خود اضافه کنید. EF به صورت خوکار و در هرجایی از پایگاه داده که لازم باشد، foreign key ایجاد می کند. با این حال داشتن foreign key در data model می تواند بروز رسانی را سهل ساخته و آن را به مراتب کارامدتر سازد. به عنوان مثال، زمانی را درنظر بگیرید که یک موجودیت course را برای ویرایش واکشی می کنید. اگر شما موجودیت department را بارگذاری نکنید، آن موجودیت تهی (null) خواهد بود. از این رو هنگامی که موجودیت course را بروز آوری می کنید، لازم است ابتدا موجودیت Department را واکشی کنید. حال اگر foreign key property به نام

Tahlildadeh

DepartmentID در data model گنجانده شود، دیگر لزومی ندارد موجودیت Department را پیش از بروز رسانی، واکشی کنید.

خصیصه ی DatabaseGenerated

خصیصه ی DatabaseGenerated که دارای پارامتر ورودی None بوده و بر روی خاصیت CourseID اعمال شده، در واقع تعیین می کند که مقادیر primary key می بایست توسط کاربر تعیین شود و نه پایگاه داده.

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
    [Display(Name = "Number")]
    public int CourseID { get; set; }
```

به صورت پیش فرض، EF فرض را بر این می گذارد که پایگاه داده خود مقادیر primary key را ایجاد می کند و در بیشتر موارد این دقیقا همان چیزی است که به آن نیاز دارید. اما برای موجودیت های Course، course number ای را استفاده می کنید که کاربر ارائه می دهد. برای مثال، برای یک دپارتمان سریال 1000 و برای دپارتمان دیگر سریال 2000 را مورد استفاده قرار می دهیم.

Foreign key (کلید خارجی) و navigation property ها

خاصیت های foreign key و navigation موجود در Course entity، روابط زیر را منعکس می کند:

1. یک دوره ی آموزشی (course) به یک دپارتمان (department) تخصیص می یابد، به همین خاطر یک کلید خارجی به نام DepartmentID و یک navigation property به نام Department، بنا به دلایلی که بالا ذکر شد، وجود دارد.

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

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

2. در یک course آموزشی می تواند تعداد زیادی دانشجو ثبت نام کرده باشد، بنابراین خاصیت Enrollments یک مجموعه (collection) محسوب می شود:

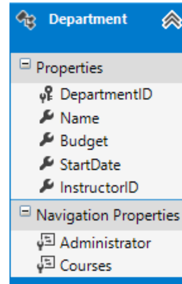
```
public virtual ICollection&lt;Enrollment> Enrollments { get; set; }
```

3. یک course ممکن است توسط چند instructor تدریس شود، از این رو خاصیت Instructors نیز یک مجموعه محسوب می شود:

Tahlildadeh

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

ایجاد موجودیت Department



فایل Models\Department.cs را با کد زیر ایجاد کنید:

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

namespace ContosoUniversity.Models
{
  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; }

    public int? InstructorID { get; set; }

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

خصیصه ی Column

پیش تر با استفاده از attribute یا خصیصه ی Column، نگاشت (mapping) را بر اساس اسم ستون تغییر دادیم. در کد مربوط به موجودیت Department، می بینید که خصیصه ی Column برای تغییر دادن mapping نوع داده ی SQL بکار می رود تا بدین وسیله ستون مورد نظر با نوع داده ی Money اس کیو ال سرور در پایگاه داده تعریف شود:

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

از آن جایی که EF نوع داده ی SQL Server مناسب را بر اساس نوع CLR ای که برای آن property تعریف کرده اید، انتخاب می کند، استفاده از column mapping در بیشتر مواقع ضرورتی ندارد. نوع decimal دات نت (CLR) به نوع decimal اس کیو ال نگاشت می شود. اما در این مورد خاص چون شما می دانید که ستون دربردارنده ی مقادیر ارزی خواهد بود، باید نوع داده ی Money را به عنوان گزینه مناسب تر درنظر گرفت.

Navigation property و Foreign Key

کلید خارجی و navigation property ها بیان گر روابط زیر می باشد:

1. یک دپارتمان ممکن است یک administrator داشته/نداشته باشد همچنین یک administrator همیشه یک instructor نیز محسوب می شود. به این خاطر خاصیت InstructorID به عنوان foreign key به موجودت Instructor اضافه می شود، سپس یک علامت سوال بعد از نوع int درج شده که آن property را nullable علامت گذاری می کند. در مثال زیر، با اینکه navigation property، Administrator نام گذاری شده اما موجودیت Instructor را نگه می دارد:

```
public int? InstructorID { get; set; }
public virtual Instructor Administrator { get; set; }
```

2. یک دپارتمان می تواند دوره های آموزشی (course) متعددی داشته باشد، خاصیت Courses نیز برای همین منظور است:

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

نکته: رسم بر این است که EF خود cascade delete را برای کلیدهای خارجی non-nullable و روابط چند به چند فعال می کند. این امر باعث می شود قوانین circular cascading deletion حاکم شده که هنگامی که

سعی می کنید migration اعمال کنید، باعث رخداد خطا می شود. برای مثال اگر خاصیت Department.InstructorID را nullable تعریف کنید، پیام خطای زیر را دریافت خواهید کرد:

"The referential relationship will result in a cyclical reference that's not allowed."

"رابطه ی ارجاعی باعث ایجاد یک circular reference می شود که اساسا غیر مجاز می باشد."

اگر قوانین مربوط به لایه ی business ایجاب کند که خاصیت InstructorID، non-nullable باشد، در آن صورت می بایست از دستور fluent API به منظور حذف این رابطه کمک گرفت:

```
modelBuilder.Entity().HasRequired(d=> d.Administrator).WithMany().WillCascadeOnDelete(false);
```

ویرایش موجودیت Enrollement

در فایل Models\Enrollment.cs، کد زیر را جایگزین کد قبلی کنید:

```
  using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
  public enum Grade
  {
    A, B, C, D, F
  }
  public class Enrollment
  {
    public int EnrollmentID { get; set; }
    public int CourseID { get; set; }
    public int StudentID { get; set; }
    [DisplayFormat(NullDisplayText = "No grade")]
    public Grade? Grade { get; set; }
    public virtual Course Course { get; set; }
    public virtual Student Student { get; set; }
  }
}
```

Property های foreign key و navigation

خاصیت های foreign key و navigation بیانگر روابط زیر هستند:

1. یک enrollment record برای تنها یک course است، از این رو یک خاصیت foreign key به نام CourseID و یک خاصیت navigation به نام Course وجود دارد:

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

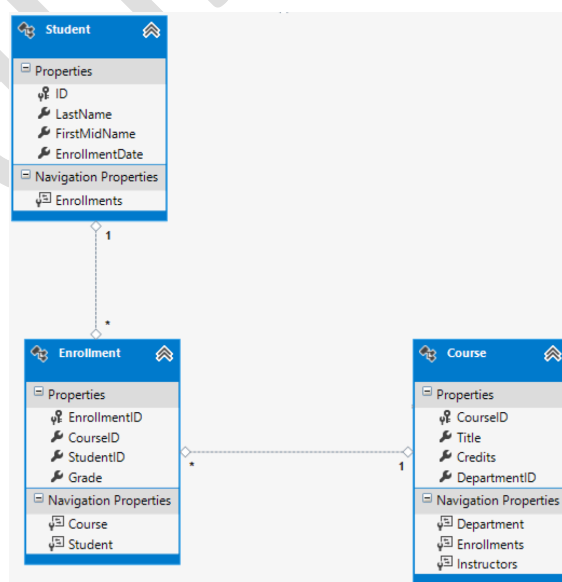
2. یک سطر enrollment تنها برای یک student است، بنابراین یک خاصیت foreign key به نام StudentID و یک navigation property به نام StudentID داریم:

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

روابط چند به چند

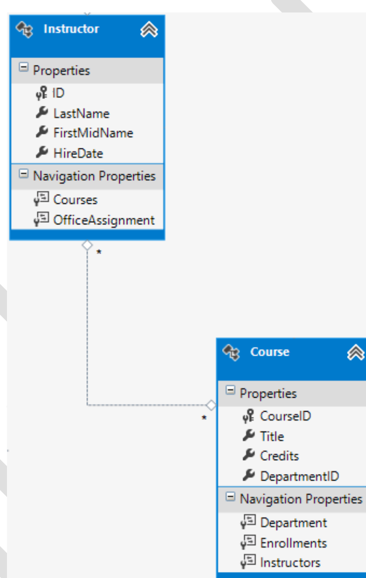
بین موجودیت های Student و Course یک نوع رابطه ی چند به چند وجود دارد، همچنین موجودیت Enrollment به عنوان یک جدول واسط بین (many-to-many join table) دو موجودیت مزبور با ستون اضافی (payload) پایگاه کار می کند، بدین معنا که جدول Enrollment، علاوه بر کلیدهای خارجی جداول ادغام شده، دارای داده های اضافه بر سازمان نیز می باشد. (در این مورد، یک کلید اصلی و یک خاصیت Grade).

تصویر زیر این رابطه را به نمایش می گذارد:

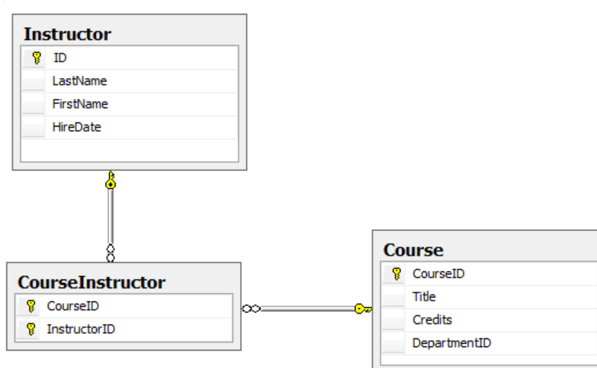


همان طور که در تصویر بالا مشاهده می کنید، هر خط ارتباط دارای یک 1 در یک طرف و یک * در طرف دیگر می باشد که نشانگر یک رابطه ی چند به چند است.

اگر جدول Enrollment شامل اطلاعات grade (نمره ی دانشجویان) نبود، در آن صورت بایستی فقط دو کلید خارجی CourseID و StudentID را دربرداشته باشد. در این مورد، آن جدول به مثابه ی یک جدول واسط چند به چند بدون ستون اضافی (یک ادغام خالص بدون ستون اضافی) در پایگاه داده خواهد بود و همچنین لازم نیست هیچ کلاس model ای برای آن ایجاد کنید. موجودیت های Instructor و Course دارای این نوع رابطه ی چند به چند هستند. همان طور که مشاهده می کنید، هیچ کلاس entity ای بین آن دو وجود ندارد:



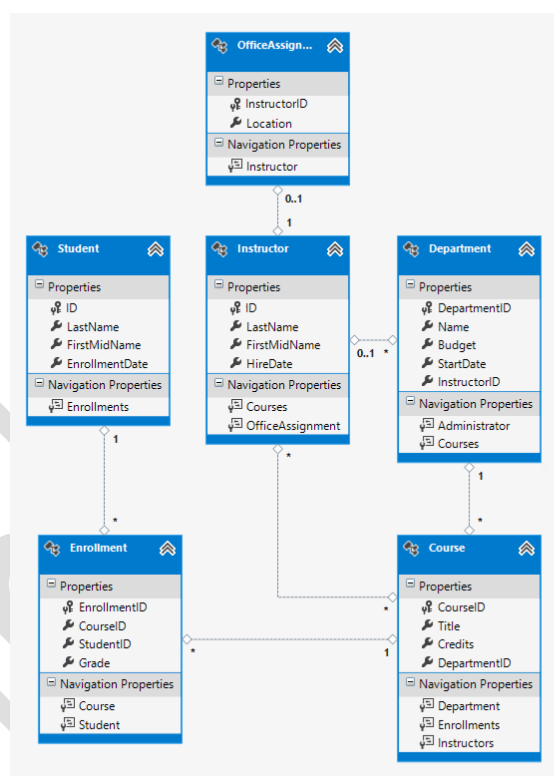
با این حال، همان طور که در جدول زیر مشاهده می کنید به یک جدول واسط (join table) در پایگاه داده نیاز داریم:



EF خود به صورت اتوماتیک جدول CourseInstructor را ایجاد می کند. شما آن را با خواندن و روز آوری navigation property های Instructor.Courses و Course.Instructors، به صورت غیر مسقیم خوانده و بروز رسانی می کنید.

نمودار entity – نشانگر ارتباط ها

تصویر زیر نموداری را نمایش می دهد که Entity Framework Power Tools برای مدل تکمیل شده ی School ایجاد کرده:



علاوه بر خط های ارتباط چند به چند (* به *) و خط های ارتباط یک به چند (1 به *)، شما یک رابطه ی یک به صفر یا یک به یک را مابین موجودیت های Instructor و OfficeAssignment (1 to 0..1) و نیز رابطه ی صفر به چند یا یک به چند (0..1 to *) را بین موجودیت های Instructor و Department مشاهده می کنید.

تنظیم سفارشی data model با افزودن کدهایی به Database Context

حال entity های جدیدی را به کلاس SchoolContext افزوده و با فراخوانی های fluent API برخی از نگاشت ها (mapping) را سفارشی تنظیم می کنیم. این رابط برنامه سازی کاربردی یا API به این خاطر fluent (خوانا)

قلمداد می شود که اغلب با گنجاندن یا ترکیب یک سری فراخوانی متد در یک دستور واحد صورت می پذیرد (Fluent API = راهی است برای پیاده‌سازی رابط برنامه‌نویسی یا همان API که کدهایی با خوانایی بهتر را پشتیبانی کند.)، نمونه ی آن را زیر مشاهده می کنید:

```
modelBuilder.Entity&lt;course>()
  .HasMany(c => c.Instructors).WithMany(i => i.Courses)
  .Map(t => t.MapLeftKey("CourseID")
  .MapRightKey("InstructorID")
  .ToTable("CourseInstructor"));
```

در این آموزش از fluent API صرفا برای نگاشت پایگاه داده بهره می گیرم، کاری که نمی توان با attribute ها انجام داد. با این وجود، می توان از fluent API به منظور مشخص کردن قالب بندی (formatting)، اعتبارسنجی (validation) و قوانین نگاشت که به وسیله ی attribute ها نیز انجام پذیر است، استفاده کنید. اعمال برخی از attribute ها نظیر MinimumLength با fluent API ممکن نیست. همان طور که قبلا شرح داده شد، MinimumLength شِمای (schema) پایگاه داده را تغییر نمی دهد، بلکه صرفا یک قانون اعتبارسنجی در سمت سرویس دهنده اعمال می نماید.

برخی از برنامه نویسان ترجیح می دهند fluent API را فقط در راستای مرتب نگه داشتن کلاس های entity خود بکار ببرند. می توان در صورت تمایل attribute ها و fluent API را با هم ترکیب کرده و بکار برد، برخی از تنظیمات هستند که فقط با استفاده از fluent API امکان پذیر می باشد، به طور کلی توصیه می شود که یکی از این دو روش را استفاده کنید و در استفاده از آن ثابت قدم باشید.

جهت افزودن موجودیت های جدید به data model و اجرای database mapping کافی است کد زیر را جایگزین کد موجود در DAL\SchoolContext.cs کنید:

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

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

Tahlildadeh

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

      modelBuilder.Entity&lt;Course>()
        .HasMany(c => c.Instructors).WithMany(i => i.Courses)
        .Map(t => t.MapLeftKey("CourseID")
          .MapRightKey("InstructorID")
          .ToTable("CourseInstructor"));
    }
  }
}
```

دستور جدید در متد OnModelCreating، جدول واسط رابطه ی چند به چند (many-to-many join table) را پیکربندی می کند:

- برای رابطه ی چند به چندی که بین موجودیت های Instructor و Course وجود دارد، کد مورد نظر اسم جدول و ستون ها را برای جدول واسط مشخص می کند. Code First قادر است رابطه ی چند به چند را بدون کمک این کد برای شما پیکربندی کند، اما اگر این کد را صدا نکنید، در آن صورت اسم های پیش فرض همچون InstructorInstructorID برای ستون InstructorID انتخاب می شود.

```
modelBuilder.Entity&lt;Course>()
 .HasMany(c => c.Instructors).WithMany(i => i.Courses)
 .Map(t => t.MapLeftKey("CourseID")
   .MapRightKey("InstructorID")
   .ToTable("CourseInstructor"));
```

کد زیر نمایش می دهد چگونه می توان بجای استفاده از attribute ها، از fluent API به منظور مشخص کردن رابطه ی بین موجودیت های Instructor و OfficeAssignment بهره گرفت:

```
  modelBuilder.Entity&lt;Instructor>()
 .HasOptional(p => p.OfficeAssignment).WithRequired(p => p.Instructor);
```

مقداردهی اولیه ی (Seed) پایگاه داده با داده های آزمایشی

کد موجود در فایل Migrations\Configuration.cs را با کد زیر جایگزین کرده تا بتوانید داده های اولیه (seed data) را برای موجودیت های جدیدی که ایجاد کرده اید، ارائه نمایید.

```
namespace ContosoUniversity.Migrations
{
  using ContosoUniversity.Models;
```

Tahlildadeh

```
using ContosoUniversity.DAL;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;

internal sealed class Configuration : DbMigrationsConfiguration&lt;SchoolContext>
{
  public Configuration()
  {
    AutomaticMigrationsEnabled = false;
  }

  protected override void Seed(SchoolContext context)
  {
    var students = new List&lt;Student>
    {
      new Student { FirstMidName = "Carson",   LastName = "Alexander",
        EnrollmentDate = DateTime.Parse("2010-09-01") },
      new Student { FirstMidName = "Meredith", LastName = "Alonso",
        EnrollmentDate = DateTime.Parse("2012-09-01") },
      new Student { FirstMidName = "Arturo",   LastName = "Anand",
        EnrollmentDate = DateTime.Parse("2013-09-01") },
      new Student { FirstMidName = "Gytis",    LastName = "Barzdukas",
        EnrollmentDate = DateTime.Parse("2012-09-01") },
      new Student { FirstMidName = "Yan",      LastName = "Li",
        EnrollmentDate = DateTime.Parse("2012-09-01") },
      new Student { FirstMidName = "Peggy",    LastName = "Justice",
        EnrollmentDate = DateTime.Parse("2011-09-01") },
      new Student { FirstMidName = "Laura",    LastName = "Norman",
        EnrollmentDate = DateTime.Parse("2013-09-01") },
      new Student { FirstMidName = "Nino",     LastName = "Olivetto",
        EnrollmentDate = DateTime.Parse("2005-09-01") }
    };


    students.ForEach(s => context.Students.AddOrUpdate(p => p.LastName, s));
    context.SaveChanges();

    var instructors = new List&lt;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") },
```

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 165 واحد7
88146323 - 88446780 - 88146330

```
  new Instructor { FirstMidName = "Roger",   LastName = "Zheng",
     HireDate = DateTime.Parse("2004-02-12") }
};
instructors.ForEach(s => context.Instructors.AddOrUpdate(p => p.LastName, s));
context.SaveChanges();

var departments = new List&lt;Department>
{
  new Department { Name = "English",    Budget = 350000,
     StartDate = DateTime.Parse("2007-09-01"),
     InstructorID  = instructors.Single( i => i.LastName == "Abercrombie").ID },
  new Department { Name = "Mathematics", Budget = 100000,
     StartDate = DateTime.Parse("2007-09-01"),
     InstructorID  = instructors.Single( i => i.LastName == "Fakhouri").ID },
  new Department { Name = "Engineering", Budget = 350000,
     StartDate = DateTime.Parse("2007-09-01"),
     InstructorID  = instructors.Single( i => i.LastName == "Harui").ID },
  new Department { Name = "Economics",   Budget = 100000,
     StartDate = DateTime.Parse("2007-09-01"),
     InstructorID  = instructors.Single( i => i.LastName == "Kapoor").ID }
};
departments.ForEach(s => context.Departments.AddOrUpdate(p => p.Name, s));
context.SaveChanges();

var courses = new List&lt;Course>
{
  new Course {CourseID = 1050, Title = "Chemistry",     Credits = 3,
    DepartmentID = departments.Single( s => s.Name == "Engineering").DepartmentID,
    Instructors = new List&lt;Instructor>()
  },
  new Course {CourseID = 4022, Title = "Microeconomics", Credits = 3,
    DepartmentID = departments.Single( s => s.Name == "Economics").DepartmentID,
    Instructors = new List&lt;Instructor>()
  },
  new Course {CourseID = 4041, Title = "Macroeconomics", Credits = 3,
    DepartmentID = departments.Single( s => s.Name == "Economics").DepartmentID,
    Instructors = new List&lt;Instructor>()
  },
  new Course {CourseID = 1045, Title = "Calculus",      Credits = 4,
    DepartmentID = departments.Single( s => s.Name == "Mathematics").DepartmentID,
    Instructors = new List&lt;Instructor>()
  },
  new Course {CourseID = 3141, Title = "Trigonometry",   Credits = 4,
    DepartmentID = departments.Single( s => s.Name == "Mathematics").DepartmentID,
    Instructors = new List&lt;Instructor>()
  },
  new Course {CourseID = 2021, Title = "Composition",    Credits = 3,
    DepartmentID = departments.Single( s => s.Name == "English").DepartmentID,
    Instructors = new List&lt;Instructor>()
  },
```

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 165 واحد7
88146323 - 88446780 - 88146330

```
   new Course {CourseID = 2042, Title = "Literature",    Credits = 4,
    DepartmentID = departments.Single( s => s.Name == "English").DepartmentID,
    Instructors = new List&lt;Instructor>()
   },
 };
 courses.ForEach(s => context.Courses.AddOrUpdate(p => p.CourseID, s));
 context.SaveChanges();

 var officeAssignments = new List&lt;OfficeAssignment>
 {
   new OfficeAssignment {
      InstructorID = instructors.Single( i => i.LastName == "Fakhouri").ID,
      Location = "Smith 17" },
   new OfficeAssignment {
      InstructorID = instructors.Single( i => i.LastName == "Harui").ID,
      Location = "Gowan 27" },
   new OfficeAssignment {
      InstructorID = instructors.Single( i => i.LastName == "Kapoor").ID,
      Location = "Thompson 304" },
 };
 officeAssignments.ForEach(s => context.OfficeAssignments.AddOrUpdate(p => p.InstructorID, s));
 context.SaveChanges();

 AddOrUpdateInstructor(context, "Chemistry", "Kapoor");
 AddOrUpdateInstructor(context, "Chemistry", "Harui");
 AddOrUpdateInstructor(context, "Microeconomics", "Zheng");
 AddOrUpdateInstructor(context, "Macroeconomics", "Zheng");

 AddOrUpdateInstructor(context, "Calculus", "Fakhouri");
 AddOrUpdateInstructor(context, "Trigonometry", "Harui");
 AddOrUpdateInstructor(context, "Composition", "Abercrombie");
 AddOrUpdateInstructor(context, "Literature", "Abercrombie");

 context.SaveChanges();

 var enrollments = new List&lt;Enrollment>
 {
   new Enrollment {
      StudentID = students.Single(s => s.LastName == "Alexander").ID,
      CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
      Grade = Grade.A
   },
    new Enrollment {
      StudentID = students.Single(s => s.LastName == "Alexander").ID,
      CourseID = courses.Single(c => c.Title == "Microeconomics" ).CourseID,
      Grade = Grade.C
    },
    new Enrollment {
      StudentID = students.Single(s => s.LastName == "Alexander").ID,
      CourseID = courses.Single(c => c.Title == "Macroeconomics" ).CourseID,
```

```
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Alonso").ID,
            CourseID = courses.Single(c => c.Title == "Calculus" ).CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Alonso").ID,
            CourseID = courses.Single(c => c.Title == "Trigonometry" ).CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Alonso").ID,
            CourseID = courses.Single(c => c.Title == "Composition" ).CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Anand").ID,
            CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Anand").ID,
            CourseID = courses.Single(c => c.Title == "Microeconomics").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Barzdukas").ID,
            CourseID = courses.Single(c => c.Title == "Chemistry").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Li").ID,
            CourseID = courses.Single(c => c.Title == "Composition").CourseID,
            Grade = Grade.B
        },
        new Enrollment {
            StudentID = students.Single(s => s.LastName == "Justice").ID,
            CourseID = courses.Single(c => c.Title == "Literature").CourseID,
            Grade = Grade.B
        }
    };

    foreach (Enrollment e in enrollments)
    {
        var enrollmentInDataBase = context.Enrollments.Where(
            s =>
                s.Student.ID == e.StudentID &&
                s.Course.CourseID == e.CourseID).SingleOrDefault();
        if (enrollmentInDataBase == null)
```

```
        {
          context.Enrollments.Add(e);
        }
      }
      context.SaveChanges();
    }

    void AddOrUpdateInstructor(SchoolContext context, string courseTitle, string instructorName)
    {
      var crs = context.Courses.SingleOrDefault(c => c.Title == courseTitle);
      var inst = crs.Instructors.SingleOrDefault(i => i.LastName == instructorName);
      if (inst == null)
        crs.Instructors.Add(context.Instructors.Single(i => i.LastName == instructorName));
    }
  }
}
```

همان طور که در درس اول سری آموزشی حاضر تشریح شد، بیشتر این کد صرفا entity object های موجود را بروز رسانی کرده یا entity object جدید ایجاد می کند و همچنین داده های نمونه را در propertyها آنگونه که مورنیاز تست می باشد، بارگذاری می نماید. اما لازم است توجهی به نحوه ی مدیریت موجودیت course که دارای رابطه ی چند به چند با موجودیت instructor می باشد، داشته باشید:

```
var courses = new List&lt;Course>
{
  new Course {CourseID = 1050, Title = "Chemistry",     Credits = 3,
   DepartmentID = departments.Single( s => s.Name == "Engineering").DepartmentID,
   Instructors = new List&lt;Instructor>()
  },
  ...
};
courses.ForEach(s => context.Courses.AddOrUpdate(p => p.CourseID, s));
context.SaveChanges();
```

به هنگام ایجاد یک شی Course، navigation property به نام Instructors را با نوشتن کد Instructors = new List<Instructor>() به عنوان یک مجموعه ی تهی مقداردهی اولیه می کنیم. با انجام این کار، این امکان فراهم می آید که موجودیت های Instructor که به شی Course مربوط هستند را به وسیله ی متد Instructors.Add اضافه کرد. اگر یک لیست خالی ایجاد نکرده باشید، در آن صورت قادر نخواهید بود این ارتباطات را اضافه کنید، زیرا که خاصیت Instructors تهی (null) بوده و از متد Add بهره مند نخواهد بود. همچنین می توانستید مقداردهی اولیه آن لیست را به constructor (سازنده) اضافه کنید.

افزودن migration و بروز رسانی پایگاه داده

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 165 واحد7
88146323 - 88446780 - 88146330

دستور add-migration را در پنجره ی PMC وارد کنید (هنوز لازم به اجرای دستور update-database نیست):

```
add-Migration ComplexDataModel
```

اگر سعی کنید دستور update-database را الان اجرا کنید، با پیغام خطای زیر مواجه می شوید:

The ALTER TABLE statement conflicted with the FOREIGN KEY constraint "FK_dbo.Course_dbo.Department_DepartmentID". The conflict occurred in database "ContosoUniversity", table "dbo.Department", column 'DepartmentID'.

دستور ALTER TABLE با محدودیت FOREIGN KEY تداخل پیدا کرد "FK_dbo.Course_dbo.Department_DepartmentID". این تداخل در پایگاه داده ی "ContosoUniversity"، در جدول "dbo.Department" و در ستون 'DepartmentID' رخ داد.

گاهی اوقات به هنگام اجرای migration با داده های از پیش موجود، بایستی stub data (فایل های stub به فایل هایی گفته می شود که به نظر کاربر بر روی دیسک مستقر بوده و بلافاصله قابل استفاده می باشد، در حالی که کل یا بخشی از آن در یک محل ذخیره سازی متفاوت قرار دارد.) را داخل پایگاه داده درج کرده تا شرط محدودیت های foreign key برقرار شود. این همان کاری است که اکنون می بایست انجام داد. کد ایجاد شده درون متد ComplexDataModel Up، یک کلید خارجی به نام DepartmentID که non-nullable می باشد، به جدول Course اضافه می کند. به این خاطر که در زمان اجرای کد، از قبل سطرهایی در جدول Course وجود دارد، عملیات AddColumn با شکست مواجه می شود. دلیل آن هم این است که SQL Server نمی داند چه مقداری در ستون قرار دهد که null نباشد (قابلیت تهی بودن را نداشته باشد). از این رو می بایست کد را طوری ویرایش کنید که به ستون جدید یک مقدار پیش فرض تخصیص یابد و یک stub department (دپارتمان ساختگی) به نام "Temp" ایجاد کنیم که به عنوان دپارتمان پیش فرض عمل کند. در نتیجه، سطرهای جاری جدول Course همگی پس از اجرای متد Up، به دپارتمان "Temp" متصل و مربوط می باشند. شما می توانید آن ها را از طرق (در) متد Seed به دپارتمان مربوطه ربط داد.

فایل <timestamp>_ComplexDataModel.cs را ویرایش کرده، خط کدی را که ستون DepartmentID را به جدول Course اضافه می کند(با استفاده از comment) حذف کنید، سپس کد هایلایت شده ی زیر را بجای آن قرار دهید:

```
CreateTable(
    "dbo.CourseInstructor",
    c => new
        {
            CourseID = c.Int(nullable: false),
            InstructorID = c.Int(nullable: false),
        })
    .PrimaryKey(t => new { t.CourseID, t.InstructorID })
    .ForeignKey("dbo.Course", t => t.CourseID, cascadeDelete: true)
    .ForeignKey("dbo.Instructor", t => t.InstructorID, cascadeDelete: true)
    .Index(t => t.CourseID)
    .Index(t => t.InstructorID);

// ایجاد یک department که به یک Cource به آن اشاره کند. //
  Sql("INSERT INTO dbo.Department (Name, Budget, StartDate) VALUES ('Temp', 0.00, GETDATE())");
 //مقدار پیش فرض برای کلید خارجی که به Department ایجاد شده در بالا اشاره میکند.//
  AddColumn("dbo.Course", "DepartmentID", c => c.Int(nullable: false, defaultValue: 1));
  //AddColumn("dbo.Course", "DepartmentID", c => c.Int(nullable: false));

  AlterColumn("dbo.Course", "Title", c => c.String(maxLength: 50));
```

زمانی که متد Seed اجرا می شود، این متد سطرهایی را در جدول Department وارد کرده، سپس سطرهای از پیش موجود جدول Course را به سطرهای جدید جدول Department ربط می دهد. اگر هیچ دوره ی آموزشی ای را از طریق رابط کاربری یا UI اضافه نکرده اید، در آن صورت دیگر نیازی به مقدار پیش فرض یا department ای به نام "Temp" درستون Course.DepartmentID ندارید. برای دادن این احتمال که ممکن است کسی از طریق اپلیکیشن دوره های آموزشی ای (course) را اضافه کرده باشد و مقابله با آن، می بایست کد متد Seed را آپدیت کنید. با این کار مطمئن می شوید که تمام سطرهای جدول Course (نه فقط آن دسته از سطرهایی که در دفعات قبلی اجرای متد Seed درج گردیده)، پیش از اینکه مقدار پیش فرض را از ستون پاک کرده و دپارتمان "Temp" را حذف کنید، دارای مقادیر DepartmentID صحیح باشند.

پس از اتمام ویرایش فایل <timestamp>_ComplexDataModel.cs، دستور update-database را در پنجره ی PMC وارد کرده تا migration اجرا شود.

update-database

نکته: ممکن است حین انتقال (migrate) داده ها و ایجاد تغییرات در schema با خطاهای دیگری مواجه شوید. در صورت دریافت خطاهای migration که قادر به رفع آن ها نیستید، می توانید اسم پایگاه داده را در connection string اصلاح کنید و یا به طور کلی پایگاه داده را حذف نمایید. ساده ترین روش تغییر اسم پایگاه داده در فایل Web.config می باشد. در مثال زیر تغییر اسم پایگاه داده به CU_Test را مشاهده می کنید:

```
    &lt;add name="SchoolContext" connectionString="Data Source=(LocalDb)\v11.0;Initial
Catalog=CU_Test;Integrated Security=SSPI;"
   providerName="System.Data.SqlClient" />
```

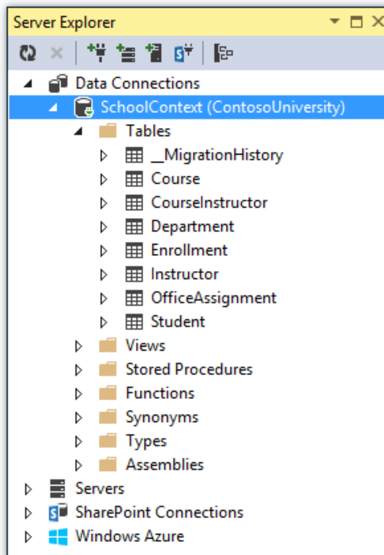
با یک پایگاه داده ی جدید، دیگر داده ای برای انتقال (migrate) وجود ندارد و دستور update-database به طور کامل و بدون رخداد خطا اجرا می شود.

در صورت عدم موفقیت روش گفته شده، می توانید با وارد کردن دستور زیر در پنجره ی PMC پایگاه داده را دوباره راه اندازی (re-initialize) کنید:

update-database -TargetMigration:0

پایگاه داده ی مورد نظر را در پنجره ی Server Explorer باز کرده، سپس گره ی Tables را باز کنید. خواهید دید که تمامی جداول ایجاد شده اند. (اگر Server Explorer را قبلا باز کرده اید، اکنون دکمه ی Refresh را فشار دهید.)

Tahlildadeh



شما کلاس model برای جدول CourseInstructor ایجاد نکرده اید. همان طور که پیش تر شرح داده شد، این یک جدول واسط برای رابطه ی چند به چند بین موجودیت های Instructor و Course می باشد.

روی جدول CourseInstructor راست کلیک کرده و گزینه ی Show Table Data را انتخاب نمایید تا مطمئن شوید که داده هایی که به موجب افزودن موجودیت های Instructor به navigation property به نام Course.Instructors بایستی اضافه شده باشند، در آن موجود می باشند.

dbo.CourseInstructor [Data]

Max Rows: 1000

CourseID	InstructorID
2021	1
2042	1
1045	2
1050	3
3141	3
1050	4
4022	5
4041	5
NULL	NULL

خلاصه

اکنون دارای یک data model و پایگاه داده ی مربوطه ی پیچیده تری هستیم. در دروس بعدی با روش های مختلف دیگری برای دسترسی به داده های مربوط آشنا خواهید شد.

Tahlildadeh