بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

# فعال سازی ابزار Code First Migrations

مدرس : مهندس افشین رفوآ

دوره آموزش MVC

## فعال سازی ابزار Code First Migrations و مستقر سازی و گسترش برنامه در اینترنت با EF – MVC5

تا به حال برنامه به صورت محلی بر روی IIS Express (نسخه ی Express سرویس‌های اطلاعات اینترنتی) و در محیط رایانه ی شخصی شما اجرا می شد. جهت قرار دادن برنامه در دسترس عموم در محیط اینترنت، بایستی آن را بر روی یک ارائه دهنده ی میزبان وب (web hosting provider) مستقر سازید (نصب کنید). در این مبحث به تشریح فعال سازی Code First Migrations خواهیم پرداخت. قابلیت Migration به شما این امکان را می دهد که data model را تغییر داده و تغییرات ایجاد شده ی خود را در محیط production مستقر ساخته و اعمال کنید. این کار را با بروز رسانی database schema و بدون اینکه پایگاه داده را حذف و مجدد ایجاد کنیم، انجام خواهیم داد.

## فعال سازی ابزار Code First Migrations

هنگامی که یک برنامه ی جدید می نویسید، data model شما بارها تغییر می کند. هر بار که model عوض می شود، پایگاه داده و model همگامی بایکدیگر را از دست می دهند. EF را طور تنظیم کرده اید که با هر بار تغییر data model به صورت خودکار پایگاه داده را حذف کرده و مجدد ایجاد کند. هرگاه که کلاس های entity را حذف/اضافه یا اصلاح می کنید و یا کلاس DbContext را تغییر می دهید، دفعه ی بعد که برنامه را اجرا می

کنید، متوجه خواهید شد که به صورت خودکار پایگاه داده ی جاری را حذف می کند، یک پایگاه داده ی جدید ایجاد می کند که با model مطابقت داشته باشد، سپس آن را با داده های آزمایشی مقداردهی اولیه (seed) می نماید.

این روش همگام (sync) نگه داشتن پایگاه داده با data model تا زمانی که برنامه را در محیط production مستقر می کنید، به درستی عمل می کند.

هنگامی که برنامه در محیط production در حال اجرا است، برنامه داده هایی را که مایل به نگه داشتن آن ها هستید را ذخیره می کند. قطعا هر بار که تغییری را (مانند افزودن یک سطر جدید) اعمال می کنید، نمی خواهید که تمامی اطلاعات برنامه از دست برود. ابزار Code First Migrations با فراهم آوردن امکان بروز رسانی database schema بجای حذف و ایجاد مجدد پایگاه داده، این مشکل را برطرف می سازد. در این درس، برنامه را نصب خواهیم ساخت. برای این منظور ابتدا باید Migration را فعال کنید.

1. در مرحله ی اول، initializer ای را که قبلا تنظیم کرده بودید با حذف المان های contexts اضافه شده به فایل Web.config برنامه، غیر فعال سازید.

```
<entityframework>
    <!--<contexts>
     <context type="ContosoUniversity.DAL.SchoolContext, ContosoUniversity">
      <databaseInitializer type="ContosoUniversity.DAL.SchoolInitializer, ContosoUniversity" />
     </context>
    </contexts>-->
    <defaultconnectionfactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
      <parameters>
        <parameter value="v11.0" />
      </parameters>
    </defaultconnectionfactory>
    <providers>
      <provider invariantname="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
    </providers>
          </entityframework>
```

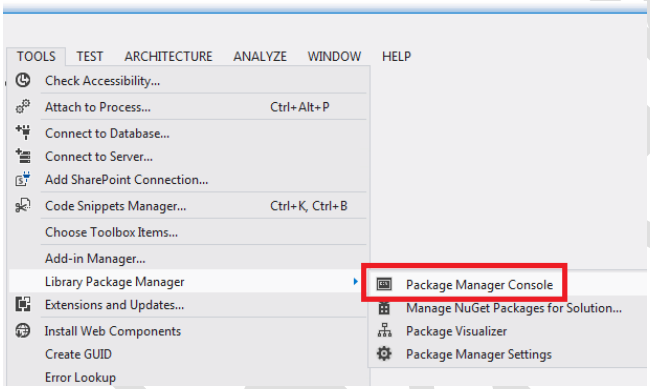
2. همچنین اسم پایگاه داده را در connection string داخل فایل Web.config، به ContosoUniversity2 تغییر دهید.

```
<connectionstrings>
```

```
    <add name="SchoolContext" connectionstring="Data Source=(LocalDb)\v11.0;Initial
Catalog=ContosoUniversity2;Integrated Security=SSPI;" providername="System.Data.SqlClient" />
  </connectionstrings>
```

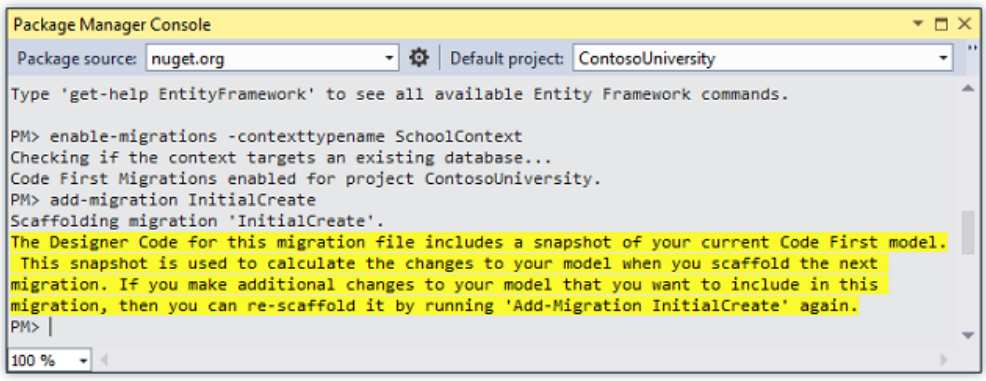
این تغییر پروژه را جوری تنظیم می کند که با اولین انتقال (migration) یک پایگاه داده ی جدید ایجاد شود.

3. از منو Tools، Library Package Manager و سپس Package Manager Console را انتخاب نمایید.




4. پس از PM>، دستورات زیر را وارد کنید:

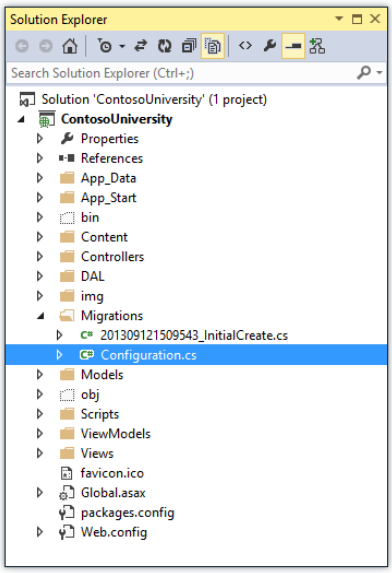
```
enable-migrations
add-migration InitialCreate
```




دستور enable-migrations یک پوشه ی Migrations در پروژه ی ContosoUniversity ایجاد می کند، سپس در آن پوشه یک فایل Configuration.cs جای گذاری می نماید که شما می توانید با ویرایش آن Migrations را پیکربندی کنید.

اگر گام قبلی را که شما را به تغییر اسم پایگاه داده ارجاع می دهد، جا انداخته یا انجام ندهید، Migration پایگاه داده ی موجود را یافته و خودکار دستور add-migration را اجرا می کند. جای نگرانی نیست؛ تنها معنی

آن این است که کدهای migration پیش از نصب (deploy) پایگاه داده، تست نخواهد شد. در آینده که دستور update-database را اجرا می کنید، خواهید دید که با هیچ مشکل خاصی مواجه نمی شوید زیرا که پایگاه داده از قبل موجود می باشد.




درست مانند کلاس initializer، کلاس Configuration دارای یک متد Seed می باشد.

```
internal sealed class Configuration : DbMigrationsConfiguration<ContosoUniversity.DAL.SchoolContext>
{
  public Configuration()
  {
    AutomaticMigrationsEnabled = false;
  }

  protected override void Seed(ContosoUniversity.DAL.SchoolContext context)
  {
//این متد بعد از انتقال به آخرین نسخه  صدا زده میشود

// helper متد الحاقی AddOrUpdate() . استفاده کنید  شما میتوانید از DbSet<T>
//.برای جلوگیری از ایجاد داده های تکراری اولیه.برای مثال
    //
    //  context.People.AddOrUpdate(
    //    p => p.FullName,
    //    new Person { FullName = "Andrew Peters" },
    //    new Person { FullName = "Brice Lambson" },
    //    new Person { FullName = "Rowan Miller" }
    //  );
    //
```

```
    }
}
```

متد Seed به شما این اجازه را می دهد که داده های آزمایشی (test data) را پس از اینکه Code First پایگاه داده را ایجاد یا بروز رسانی کرد، وارد کرده یا آپدیت نمایید. متد نام برده به هنگام ایجاد پایگاه داده و هر بار که database schema پس از تغییر data model بروز رسانی می شود، فراخوانده می شود.

## تنظیم متد Seed

به هنگام حذف و ساخت مجدد پایگاه داده پس از هر بار تغییر model، متد Seed کلاس initializer را به منظور درج داده های آزمایشی بکار می بریم، زیرا که پس از هر بار تغییر model ، پایگاه داده حذف شده و تمامی داده های آزمایشی از دست می روند. در صورت استفاده از Code First Migration، داده های آزمایشی حتی پس از تغییرات پایگاه داده نیز حفظ می شوند، از این رو لزومی ندارد داده های آزمایشی را داخل متد Seed قرار دهید. درواقع اگر از Migrations برای نصب پایگاه داده در محیط production استفاده می کنید، به هیچ وجه نباید از متد Seed برای درج داده های آزمایشی استفاده کنید، زیرا این متد در production اجرا می شود. در چنین شرایطی متد Seed را فقط جهت درج آن داده هایی در پایگاه داده استفاده می کنیم که در production به آن ها نیاز داریم. برای مثال، می خواهید پایگاه داده به هنگام مهیا شدن در محیط production، اسم دپارتمان هایی را در جدول Department، داشته باشد.

در این آموزش، از Migrations برای نصب (deployment) استفاده می شود. متد Seed در هر حال داده های آزمایشی را درج می کند تا شما بتوانید نحوه ی عملکرد برنامه را بدون اینکه مجبور به درج دستی مقدار زیادی از اطلاعات شوید، مشاهده نمایید.

1. محتویات فایل Configuration.cs را با کد زیر جایگزین نمایید. این کار داده های آزمایشی را در پایگاه داده ی جدید بارگذاری می نماید.

```
namespace ContosoUniversity.Migrations
{
    using ContosoUniversity.Models;
    using System;
    using System.Collections.Generic;
    using System.Data.Entity;
```

```
using System.Data.Entity.Migrations;
using System.Linq;

internal sealed class Configuration : DbMigrationsConfiguration<ContosoUniversity.DAL.SchoolContext>
{
  public Configuration()
  {
    AutomaticMigrationsEnabled = false;
  }

  protected override void Seed(ContosoUniversity.DAL.SchoolContext context)
  {
    var students = new List<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("2005-08-11") }
    };
    students.ForEach(s => context.Students.AddOrUpdate(p => p.LastName, s));
    context.SaveChanges();

    var courses = new List<Course>
    {
      new Course {CourseID = 1050, Title = "Chemistry",      Credits = 3, },
      new Course {CourseID = 4022, Title = "Microeconomics", Credits = 3, },
      new Course {CourseID = 4041, Title = "Macroeconomics", Credits = 3, },
      new Course {CourseID = 1045, Title = "Calculus",       Credits = 4, },
      new Course {CourseID = 3141, Title = "Trigonometry",   Credits = 4, },
      new Course {CourseID = 2021, Title = "Composition",    Credits = 3, },
      new Course {CourseID = 2042, Title = "Literature",     Credits = 4, }
    };
    courses.ForEach(s => context.Courses.AddOrUpdate(p => p.Title, s));
    context.SaveChanges();

    var enrollments = new List<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();
      }
    }
}
```

متد Seed شی database context را به عنوان پارامتر ورودی اخذ کرده، سپس کد داخل متد با استفاده از آن شی، موجودیت یا entity های جدید را به پایگاه داده اضافه می کند. به ازای هر نوع موجودیت ( entity type)، کد یک مجموعه ی جدیدی از موجودیت ها را خلق می کند، آن ها را به خاصیت (property) DbSet مربوطه الصاق کرده، سپس تغییرات را در پایگاه داده ی مورد نظر ذخیره می سازد. ضرورتی ندارد متد SaveChanges را پس از هر مجموعه موجودیت (به روشی که اینجا صورت گرفته) صدا بزنید، اما انجام این کار به شما کمک می کند که اگر حین نوشته شدن کد در پایگاه داده، خطایی یا استثنایی رخ داد، ریشه ی یک مشکل را پیدا کرده و آسان تر آن را برطرف سازید.

برخی دستورات که وظیفه ی درج یا ورود اطلاعات و داده ها را بر عهده دارند، از متد AddOrUpdate برای اجرای عملیات upsert استفاده می کنند. به این خاطر که متد Seed هر بار که دستور update-database را اجرا می کنید، اغلب پس از هر بار migration، run می شود نمی توانید اطلاعات مورد نظر را درج کنید، دلیل آن هم این است که سطرهایی که سعی بر اضافه کردن آن ها را دارید، بعد از اولین مرتبه migration که پایگاه داده را می سازد، همان زمان ساخته شده و از پیش موجود می باشد. عملیات upsert مانع از صدور خطاهایی می شود که در صورت سعی بر درج یک سطر از قبل موجود، رخ می دهد اما در این میان هر تغییری را که ممکن است حین تست برنامه اعمال کرده باشید را نیز بازنویسی (override) می کند. اگر نخواهیم این اتفاق برای

داده های آزمایشی در برخی جداول روی دهد، چی؟ در برخی مواقع که داده ها را حین تست تغییر می دهید، می خواهید که تغییرات اعمال شده پس از بروز رسانی پایگاه داده، باقیمانده یا حفظ شوند. برای این منظور از عملیات درج شرطی (conditional insert) کمک می گیریم: یک سطر تنها به شرطی وارد شود، که آن سطر از قبل وجود نداشته باشد. متد Seed از هر دو روش استفاده می کند.

اولین پارامتری که به متد AddOrUpdate فرستاده می شود، تعیین کننده ی آن خاصیتی است که بررسی می کند آیا سطر از قبل موجود می باشد یا خیر. به عنوان مثال، خاصیت LastName (برای داده های آزمایشی student که در این درس بکار می بریم) برای این منظور بسیار مناسب است زیرا که هر یک از last name های موجود در لیست منحصر بفرد می باشد:

```
context.Students.AddOrUpdate(p => p.LastName, s)
```

کد فوق فرض می گیرد که تمامی last name ها منحصر بفرد هستند. چنانچه یک student جدید را با last name تکراری به صورت دستی اضافه نمایید، دفعه ی بعدی که یک migration اجرا می کنید، با خطای زیر مواجه می شوید:

```
Sequence contains more than one element
```

کدی که موجودیت های Enrollment را ایجاد می کند، فرض می گیرد که شما مقدار ID در entity های مجموعه ی students را دارید، با صرف نظر از این قضیه که شما اصلا این خاصیت را در کدی که ایجاد کننده ی مجموعه هست، تنظیم نکرده اید.

```
new Enrollment {
  StudentID = students.Single(s => s.LastName == "Alexander").ID,
  CourseID = courses.Single(c => c.Title == "Chemistry" ).CourseID,
  Grade = Grade.A
```

می توانید خاصیت ID را اینجا بکار ببرید زیرا که مقدار ID، زمانی که متد SaveChanges را برای مجموعه ی students صدا می زنید، تنظیم شده است. EF در زمان درج یک موجودیت در پایگاه داده، مقدار کلید اصلی (primary key value) را بازیابی کرده و خاصیت ID موجودیت (entity) مقیم در حافظه را بروز رسانی کند.

کدی که هریک از موجودیت های Enrollment را به Enrollment entity set اضافه می کند، متد AddOrUpdate را بکار نمی برد. کد مزبور بررسی می کند که آیا موجودیت مورد نظر از قبل موجود می باشد یا خیر و چنانچه entity از قبل وجود نداشته باشد آن را درج می کند. این روش تمامی تغییراتی را که به enrollment grade اعمال می نمایید را با استفاده از رابط کاربری برنامه (App UI) حفظ می کند. همچنین این کد مجموعه دستور را به ازای هر یک از آیتم های عضو Enrollment List تکرار می کند و در صورت یافت نشدن enrollment در پایگاه داده، آن را به پایگاه داده اضافه می کند. اولین باری که پایگاه داده را بروز رسانی می کنید، پایگاه داده تهی خواهد بود، از این رو کد هر یک از enrollment ها را به آن اضافه می کند.

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

2. حال پروژه را بازسازی (Build) کنید.

## اجرای اولین Migration

هنگامی که دستور add-migration را اجرا می کنید، Migrations کدی را ایجاد می کند که پایگاه داده را از نو می سازد. این کد در پوشه ی Migrations و داخل فایلی به نام timestamp>_InitialCreate.cs نیز موجود می باشد. متد Up از کلاس InitialCreate، جداول پایگاه داده را که با data model entity set ها متناظر می باشد را ایجاد می کند. متد Down نیز آن جداول را حذف می کند.

```
    public partial class InitialCreate : DbMigration
{
  public override void Up()
  {
    CreateTable(
      "dbo.Course",
      c => new
        {
          CourseID = c.Int(nullable: false),
          Title = c.String(),
```

```
            Credits = c.Int(nullable: false),
        })
    .PrimaryKey(t => t.CourseID);

  CreateTable(
    "dbo.Enrollment",
    c => new
      {
        EnrollmentID = c.Int(nullable: false, identity: true),
        CourseID = c.Int(nullable: false),
        StudentID = c.Int(nullable: false),
        Grade = c.Int(),
      })
    .PrimaryKey(t => t.EnrollmentID)
    .ForeignKey("dbo.Course", t => t.CourseID, cascadeDelete: true)
    .ForeignKey("dbo.Student", t => t.StudentID, cascadeDelete: true)
    .Index(t => t.CourseID)
    .Index(t => t.StudentID);

  CreateTable(
    "dbo.Student",
    c => new
      {
        ID = c.Int(nullable: false, identity: true),
        LastName = c.String(),
        FirstMidName = c.String(),
        EnrollmentDate = c.DateTime(nullable: false),
      })
    .PrimaryKey(t => t.ID);

  }

  public override void Down()
  {
    DropForeignKey("dbo.Enrollment", "StudentID", "dbo.Student");
    DropForeignKey("dbo.Enrollment", "CourseID", "dbo.Course");
    DropIndex("dbo.Enrollment", new[] { "StudentID" });
    DropIndex("dbo.Enrollment", new[] { "CourseID" });
    DropTable("dbo.Student");
    DropTable("dbo.Enrollment");
    DropTable("dbo.Course");
  }
}
```

Migrations متد Up را فرامی خواند تا تغییرات data model را برای یک migration، اعمال کند. هنگامی که یک دستور را وارد می کنید تا بروز رسانی صورت گرفته به حال اول بازگردد، Migrations تابع Down را صدا می زند.

این همان migration اولیه است که شما با وارد کردن دستور add-migration InitialCreate، ایجاد کردید. پارامتر مورد نظر (در این مثال InitialCreate) صرفا اسم فایل را مشخص می کند و مطابق با میل شما تنظیم می شود، اما معمولا از واژه یا عبارتی استفاده می شود که نشانگر عملیات مربوطه در migration می باشد. به عنوان مثال در migration بعدی می توان اسم این پارامتر را "AddDepartmentTable" انتخاب کرد.

اگر migration اولیه را ایجاد کرده اید در حالی که پایگاه داده از قبل موجود می باشد، در آن صورت کد ساخت دیتابیس ایجاد می شود، اما از آنجایی پایگاه داده ی مورد نظر از پیش با data model منطبق (match) می باشد، اجرای آن ضرورتی ندارد. حال اگر برنامه را در محیط دیگری که پایگاه داده در آن از پیش موجود نمی باشد، مستقر سازید، در آن صورت کد مربوطه اجرا شده و پایگاه داده را ایجاد می کند. از این حیث توصیه می شود اول آن کد را آزمایش کرده و از عملکرد صحیح آن اطمینان حاصل نمایید. اگر بخاطر داشته باشید، پیش تر اسم پایگاه داده را در connection string تغییر دادیم. این کار را به این دلیل انجام دادیم تا migrations یک پایگاه داده ی جدید را از صفر بسازد.

1. دستور زیر را در پنجره ی Package Manager Console وارد کنید:

```
update-database
```

دستور update-database، متد Up را اجرا کرده و پایگاه داده را می سازد، سپس با اجرای تابع Seed، پایگاه داده را پر (populate) می کند. همین پروسه به صورت خودکار پس از اینکه شما برنامه را نصب (deploy) می کنید، در محیط production مجددا اجرا می شود.

2. با استفاده از پنجره ی Server Explorer، پایگاه داده را بررسی (inspect) کرده تا از کارکرد صحیح آن مطمئن شوید.