بسم اللّه الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

دسترسی به داده های model از یک controller

مدرس : مهندس افشین رفوآ

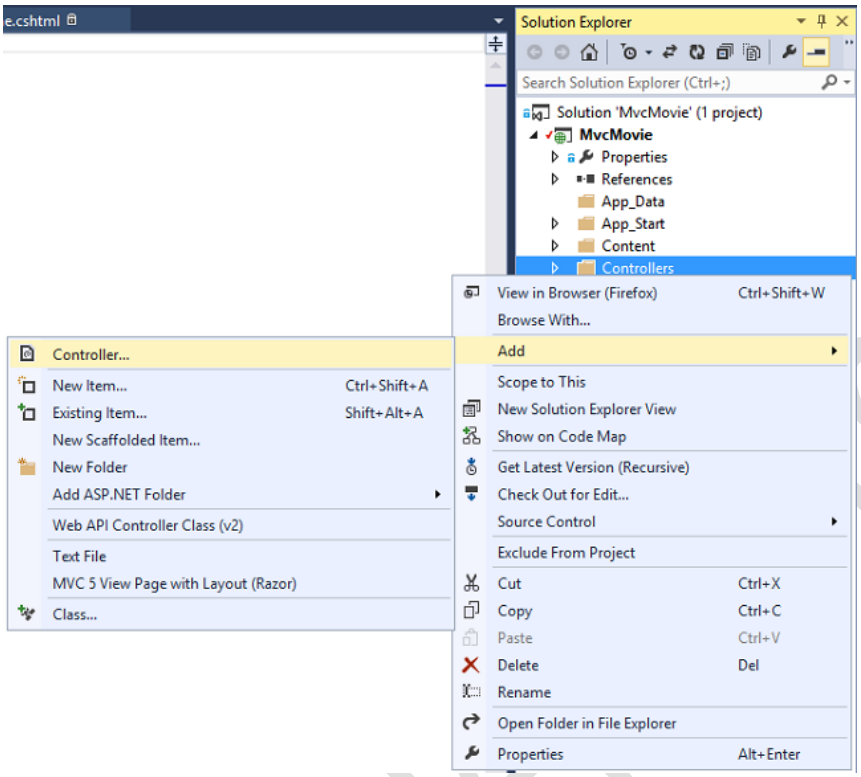
دوره آموزش MVC

دسترسی به داده های model از یک controller

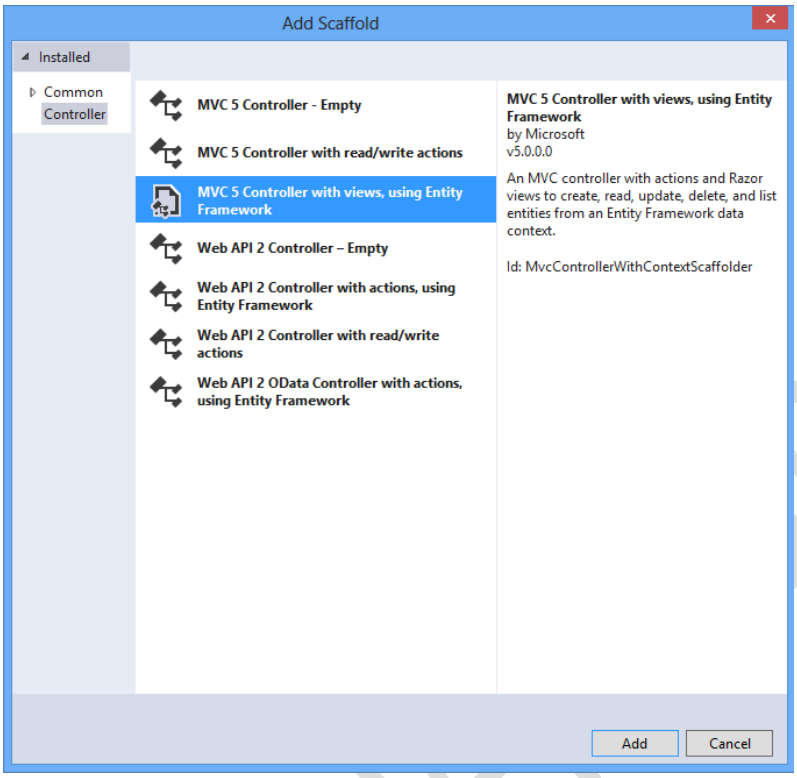
در این بخش یک کلاس MoviesController ایجاد کرده و کدی می نویسیم که اطلاعات movie را بازیابی کند و آن ها را در مرورگر با استفاده از یک view template نمایش دهد.

پیش از رفتن به مرحله ی بعد، بایستی برنامه را با زدن همزمان کلیدهای Ctrl + shift + p کامپایل کنید. اگر برنامه را کامپایل یا به اصطلاح build نکنید، در افزودن controller به خطا بر می خورید.

در پنجره ی Solution Explorer روی پوشه ی Controllers راست کلیک کرده، Add را کلیک نمایید و به دنبال آن Controller را انتخاب کنید.

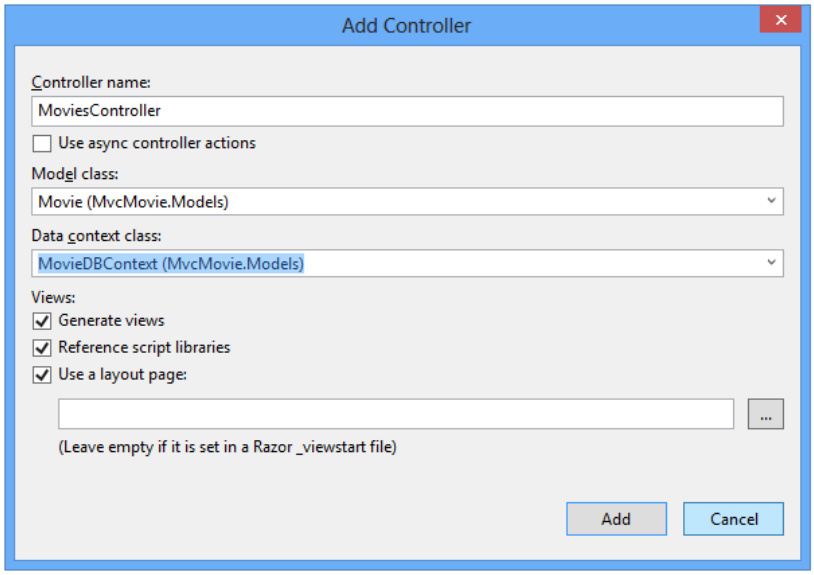


در پنجره ی محاوره ی Add Scaffold، MVC 5 Controller with views, using Entity Framework را انتخاب کرده، سپس Add را کلیک کنید.



1. در قسمت Controller name، MoviesController را انتخاب کنید.
2. در قسمت Model class، (MvcMovie.Models) Movie را انتخاب کنید.
3. برای بخش Data Context class، (MvcMovie.Models) MovieDBContext را انتخاب کنید.

تصویر زیر پنجره ی محاوره ی تکمیل شده با انتخاب های شما را نمایش می دهد.

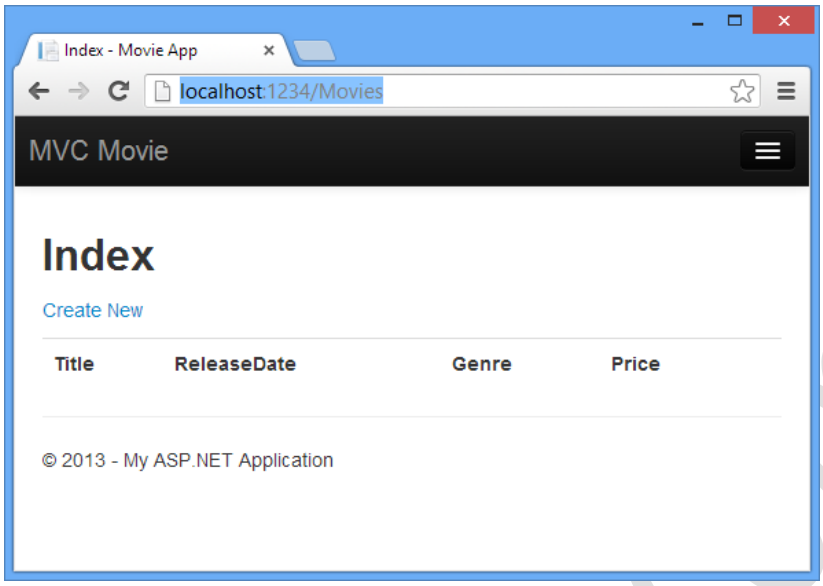


Add را کلیک کنید. (در صورت دریافت پیغام خطا، بدانید که به احتمال زیاد برنامه را پیش از افزودن controller به آن، build نکرده اید.) visual studio فایل ها و پوشه های زیر را ایجاد می کند:

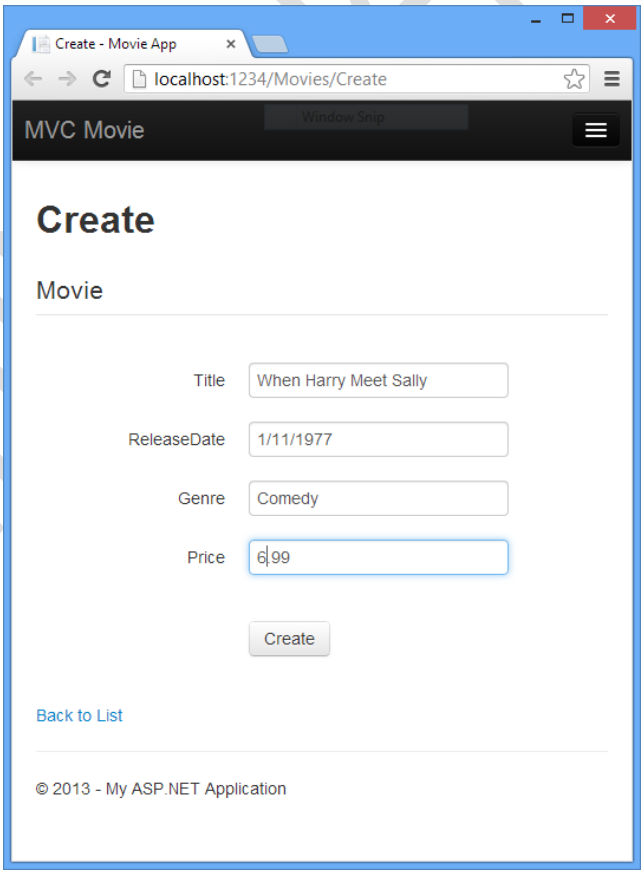
1. یک فایل به نام MoviesController.cs داخل پوشه ی Controllers.
2. یک پوشه ی Movies/Views.
3. Create.cshtml، Delete.cshtml، Details.cshtml، Edit.cshtml و Index.cshtml در پوشه ی تازه ایجاد شده ی Views\Movies.

Visual Studio به صورت خودکار action method های CRUD (ایجاد ، خواندن، بروز رسانی و حذف = چهار عملکرد اساسی مدیریت داده‌ها در برنامه‌نویسی هستند) و view های لازم را برای شما ایجاد کرد (پروسه ی ساخت خودکار action method های CRUD و view های مورد نیاز، Scaffolding نامیده می شود.) اکنون شما دارای یک برنامه ی تحت وب کاملا کاربردی هستید که به شما اجازه می دهد، برای فیلم های خود ورودی ایجاد کرده، آن ها را فهرست، ویرایش و در صورت لزوم حذف کنید.

برنامه را اجرا کرده و روی لینک MVC Movie کلیک کنید (یا با الحاق کردن /Movies به URL در نوار آدرس مرورگر، به Movies controller مراجعه کنید). از آن جایی که برنامه از routing پیش فرض استفاده می کند (که در فایل App_Start\RouteConfig.cs تعریف شده)، درخواست مرورگر http://localhost:xxxxx/Movies به action method پیش فرض Movies controller که همان Index می باشد، route (هدایت و فرستاده) می شود. به عبارتی دیگر، می توان گفت که درخواست http://localhost:xxxxx/Movies و http://localhost:xxxxx/Movies/Index اساسا یکی هستند.
نتیجه یک فهرست از فیلم هاست که هم اکنون هیچ ورودی ندارد، زیرا هنوز آیتمی به آن اضافه نشده است.

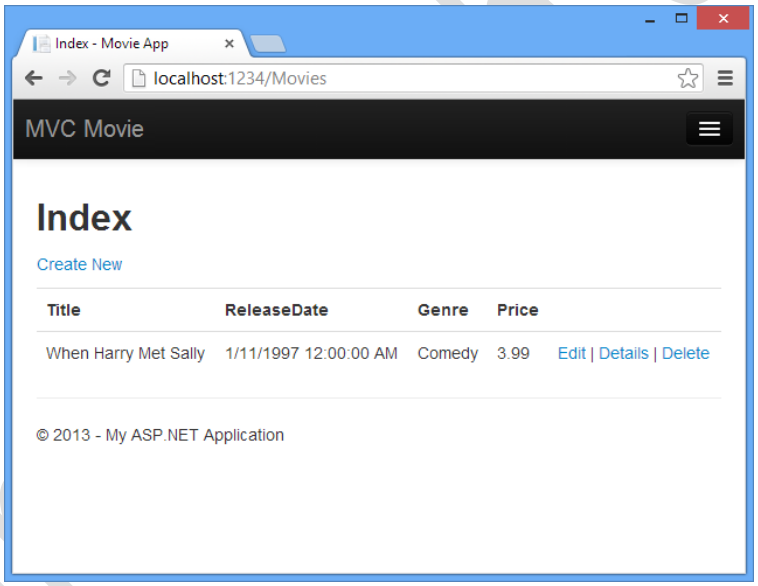


لینک Create New را انتخاب کنید. مقداری جزئیات درباره ی یک فیلم به آن اضافه کرده سپس دکمه ی Create را کلیک نمایید.



توجه: ممکن است در وارد کردن کاراکتر ممیز اعشار یا ویرگول در فیلد price با مشکل مواجه شوید. به منظور پشتیبانی از اعتبار سنجی jQuery برای مناطقی از جهان که از کاراکتر دیگری مثل " , " به عنوان نقطه ی ممیز اعشار استفاده می کنند و همچنین فرمت تاریخ غیر از انگلیسی آمریکایی دارند، بایستی globalize.js و نیز فایل cultures/globalize.cultures.js ویژه ی خود (از آدرس https://github.com/jquery/globalize) و جاوا اسکریپت را برای استفاده از Globalize.parseFloat، include کنید. در مبحث بعدی نحوه ی انجام این کار را برای شما شرح خواهم داد. در حال حاضر تنها اعداد صحیح مانند 10 را وارد می کنیم.

زمانی که اطلاعات مربوط به فیلم در پایگاه داده ثبت و ذخیره شده، کلیک بر روی دکمه ی Create باعث می شود فرم مورد نظر به سرور ارسال شود. شما سپس به آدرس /Movie URL ، redirect (هدایت) می شوید، در آنجا می توانید فیلم تازه ایجاد شده را در فهرست مشاهده کنید.



چندین فیلم دیگر به دلخواه اضافه کنید. حال لینک های Edit، Details و Delete را امتحان کنید. خواهید دید که همه لینک ها به درستی کار می کنند.

بررسی کد ایجاد شده:

فایل Controllers\MoviesController.cs را باز کرده و متد ایجاد شده ی Index را امتحان کنید. بخشی از movie controller با متد Index در زیر به نمایش گذاشته شده است.

```
public class MoviesController : Controller
{
  private MovieDBContext db = new MovieDBContext();

  // GET: /Movies/
  public ActionResult Index()
  {
    return View(db.Movies.ToList());
  }
```

درخواست به Movies controller، در پاسخ تمامی ورودی های موجود در جدول Movies را بازمی گرداند و سپس نتایج را به Index view ارسال می کند. خط زیر از کلاس MoviesController، یک بستر اجرای پایگاه داده (database context) movie نمونه سازی می کند. می توانید با استفاده از بستر اجرای پایگاه داده، از ورودی های movie، query بگیرید، آن ها را ویرایش و در صورت نیاز حذف کنید.

```
private MovieDBContext db = new MovieDBContext();
```

Model های وابسته ی شدید به نوع (Strongly Typed) و کلیدواژه ی @model

در بخش های قبلی این مقاله ی آموزشی دیدیم که چگونه یک controller می تواند داده یا اشیایی را با استفاده از شی ViewBag به view template ارسال کند. ViewBag ، یک شی پویا یا به اصطلاح dynamic است که با استفاده از سازکار late binding (اتصال پویا/با تاخیر = مکانیزمی است که طی آن متدی که برای یک شی فراخوانده می شود یا تابعی که با آرگومان صدا زده می شود توسط اسم آن در زمان اجرا مورد جستجو قرار می گیرد) اطلاعات را به یک view ارسال می کند.

MVC همچنین این اختیار را به شما می دهد که اشیا وابسته ی شدید به نوع (strongly typed) را به view template ارسال کنید. استفاده از این روش، compile-time checking (بررسی در زمان ترجمه) بهتر و قابلیت های IntelliSense غنی تری را در ویرایش گر Visual Studio به ارمغان می آورد. مکانیزم scaffolding در Visual Studio از همین روش (که همان ارسال model وابسته ی شدید به نوع می باشد) با کلاس MoviesController و view template ها، در زمان ایجاد view ها و متدهای لازم، استفاده کرد.

در فایل Controllers\MoviesController.cs، متد ایجاد شده ی Details را آزمایش کنید. متد Details در زیر نمایش داده شده است.

```
public ActionResult Details(int? id)
    {
      if (id == null)
      {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
      }
      Movie movie = db.Movies.Find(id);
      if (movie == null)
      {
        return HttpNotFound();
      }
      return View(movie);
    }
```

پارامتر id اغلب به عنوان route data ارسال می شود، برای مثال

http://localhost:1234/movies/details/1، controller را روی movie controller، action را با details

و id را روی 1 تنظیم می کند. می توان id را با یک query string مانند نمونه ی زیر ارسال کرد:

http://localhost:1234/movies/details?id=1

در صورت یافتن یک movie، یک نمونه از Movie model به Details view فرستاده می شود:

```
return View(movie);
```

محتویات فایل Views\Movies\Details.cshtml را بررسی کنید:

```
@model MvcMovie.Models.Movie
@{
  ViewBag.Title = "Details";
}
<h2>Details</h2>
<div>
  <h4>Movie</h4>
  <hr />
  <dl class="dl-horizontal">
    <dt>
      @Html.DisplayNameFor(model => model.Title)
    </dt>
    @*Markup omitted for clarity.*@
  </dl>
</div>
<p>
  @Html.ActionLink("Edit", "Edit", new { id = Model.ID }) |
  @Html.ActionLink("Back to List", "Index")
</p>
```

با افزودن دستور @model در بالای (به ابتدای) فایل view template، می توان نوع شی ای که view انتظار آن را دارد، مشخص کرد. پس از اینکه کنترلر movie را ایجاد کردید، Visual Studio به صورت خوکار دستور @model را به ابتدای فایل Details.cshtml اضافه می کند:

```
@model MvcMovie.Models.Movie
```

دستور @model با استفاده از یک شی Model که وابسته ی زیاد به نوع است، این امکان را در اختیار شما قرار می دهد که به ورودی (movie) ارسال شده توسط controller به view دسترسی داشته باشید. به عنوان مثال، در قالب Details.cshtml، کد تک تک فیلدها (movie) را با استفاده از شی Model وابسته ی شدید به نوع به توابع HTML Helper، DisplayNameFor و DisplayFor ارسال می کند. متدهای Create، Edit و view template ها نیز یک movie model object ارسال می کنند.

View template، Index.cshtml و متد Index در فایل MoviesController.cs را آزمایش کنید. توجه کنید که چگونه کد مورد نظر یک شی List، به هنگام فراخوانی متد View، در متد Index، ایجاد می کند. کد ذکر شده سپس لیست Movies را از متد Index به view ارسال می کند:

```
public ActionResult Index()
{
  return View(db.Movies.ToList());
}
```

هنگامی که movie controller را ایجاد کردید، Visual Studio به صورت خودکار دستور @model را به ابتدای فایل Index.cshtml ضمیمه می کند:

```
@model IEnumerable<MvcMovie.Models.Movie>
```

دستور @model امکان دسترسی به فهرست movie های فرستاده شده توسط controller به view را با استفاده از یک شی Model که وابسته ی شدید به نوع است فراهم می نماید. برای مثال، در قالب Index.cshtml، کد مورد نظر با اجرای دستور foreach بر روی شی model که وابسته ی شدید به نوع (strongly typed) است، درون لیست movie ها حلقه می زند:

```
@foreach (var item in Model) {
<tr>
  <td>
    @Html.DisplayFor(modelItem >= item.Title)
  </td>
  <td>
    @Html.DisplayFor(modelItem >= item.ReleaseDate)
  </td>
  <td>
    @Html.DisplayFor(modelItem >= item.Genre)
  </td>
  <td>
    @Html.DisplayFor(modelItem >= item.Price)
  </td>
  <th>
    @Html.DisplayFor(modelItem >= item.Rating)
  </th>
  <td>
    @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
    @Html.ActionLink("Details", "Details", new { id=item.ID }) |
    @Html.ActionLink("Delete", "Delete", new { id=item.ID })
  </td>
</tr>
}
```

به این خاطر که شی Model، strongly-typed است، هر یک از اشیا item در حلقه دارای نوع Movie می باشد. صرف نظر از دیگر مزایایی که به همراه دارد، قابلیت چک کردن در زمان ترجمه و پشتیبانی کامل از IntelliSense در ویرایش گر کد نیز فراهم می آید:

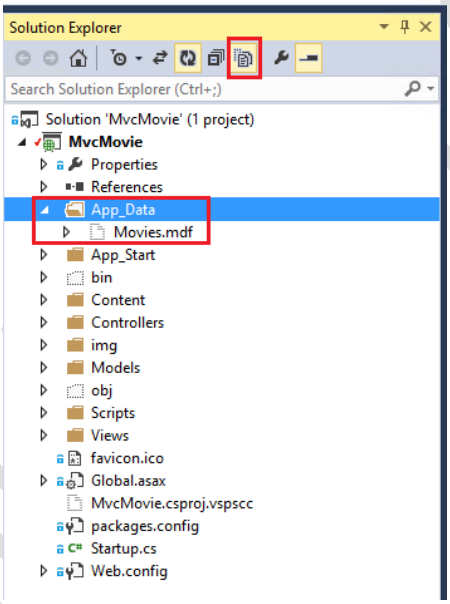
```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", ne
            @Html.ActionLink("Details", "Detail
            @Html.ActionLink("Delete", "Delete"
        </td>
    </tr>
}

</table>
```

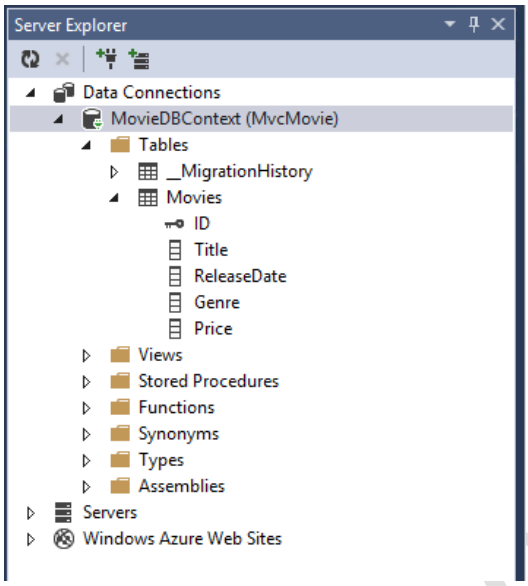
```
Equals
Genre
GetHashCode
GetType
ID
Price
ReleaseDate
Title
ToString
```

کار با نسخه ی SQL Server LocalDB

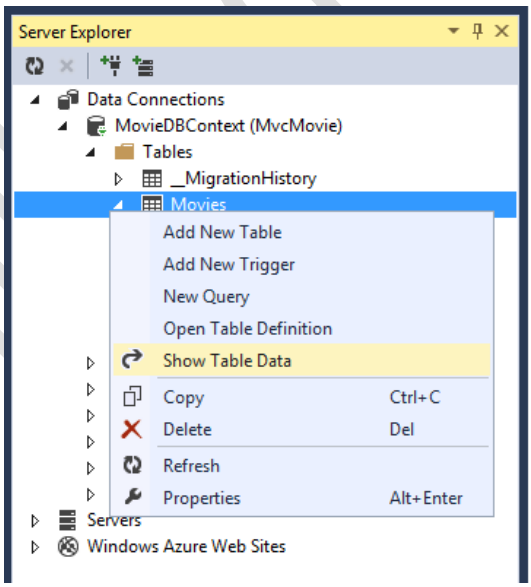
Entity Framework Code First تشخیص داد که connection string ای که ارائه شده، به پایگاه داده ی Movies اشاره می کند. اما چنین پایگاه داده ای هنوز ایجاد نشده، بنابراین Code First خود اقدام به ایجاد پایگاه داده ی مورد نظر کرد. برای اینکه بتوان بررسی کرد و دید که آیا این پایگاه داده ی movies ایجاد شده یا خیر، کافی است داخل پوشه ی App_Data را بررسی کنید. در صورت نیافتن فایل Movies.mdf، دکمه ی Show All Files را در نوار ابزار پنجره ی Solution Explorer کلیک کرده و به دنبال آن دکمه ی Refresh را کلیک نمایید، سپس پوشه ی App_Data را باز کنید.

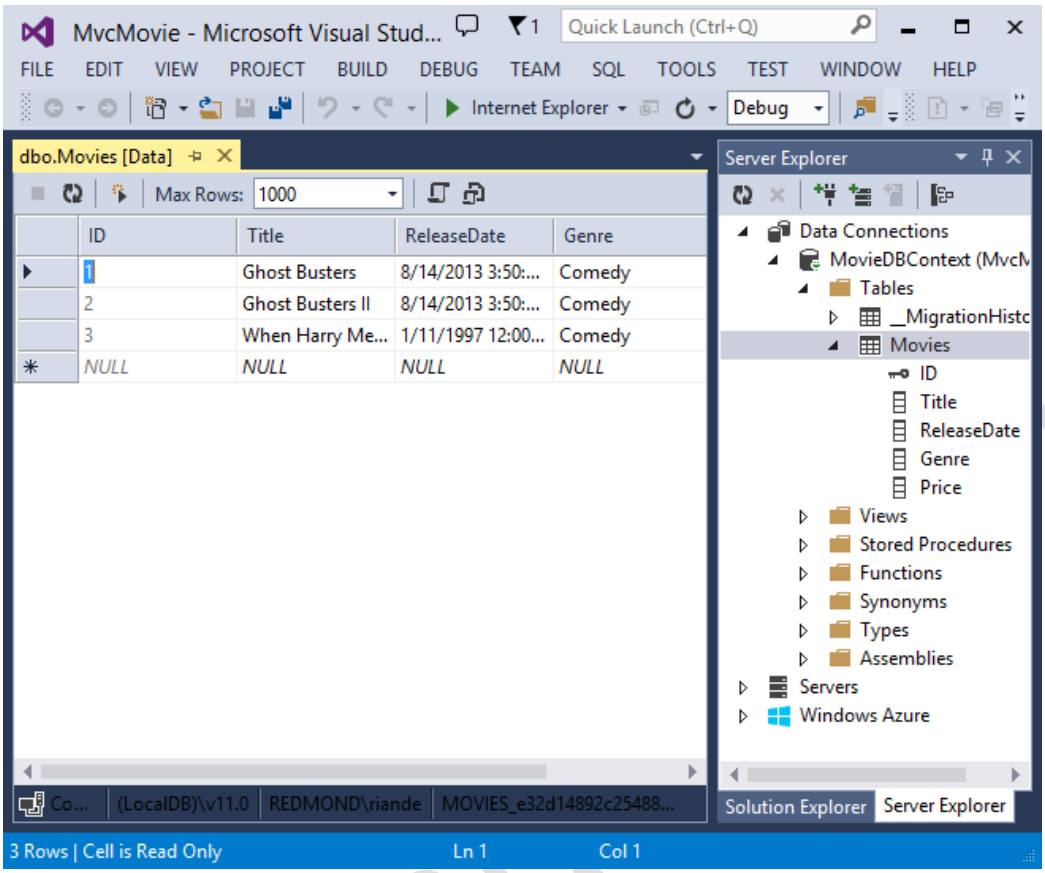


Movies.mdf را دوبار کلیک کرده تا پنجره ی SERVER EXPLORER نمایش داده شود، سپس برای مشاهده ی جدول Movies پوشه ی Tables را باز کنید. به آیکون کلید در سمت چپ ID دقت کنید، به صورت پیش فرض، EF خاصیتی که ID نام گذاری شده را به عنوان کلید اصلی (primary key) انتخاب می کند.

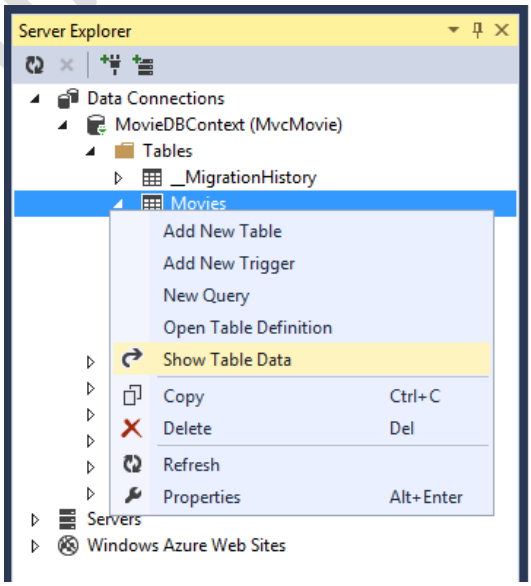


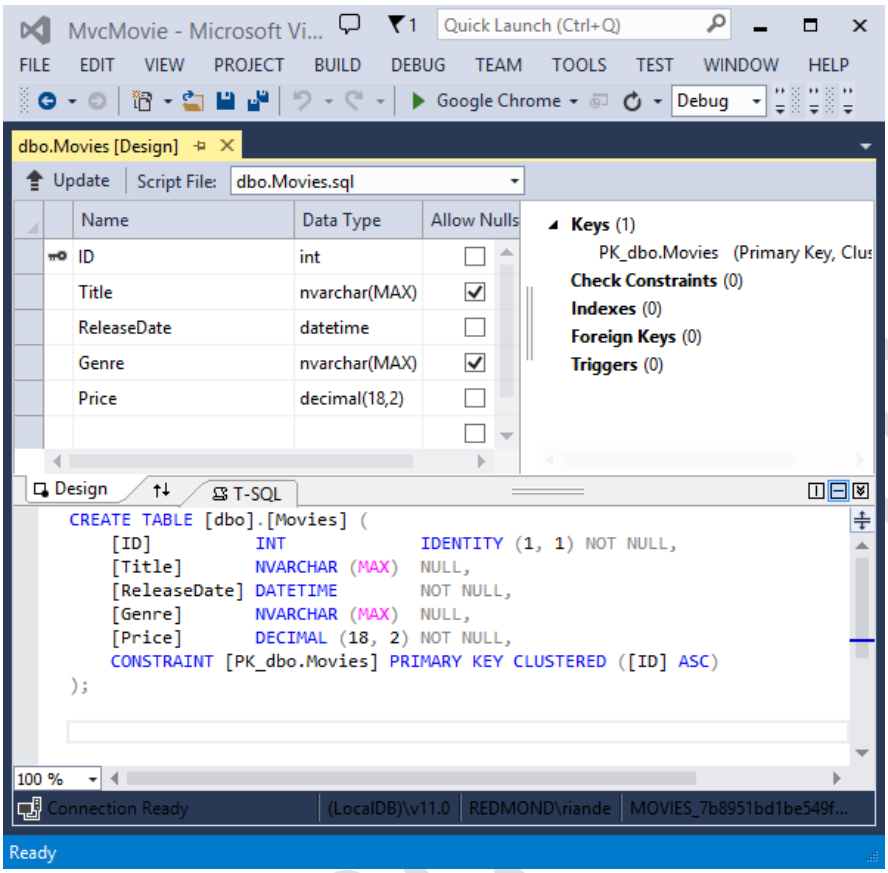
روی جدول Movies راست کلیک کرده و سپس به منظور مشاهده ی داده هایی که ایجاد کرده اید، Show Table Data را انتخاب نمایید.





روی جدول Movies راست کلیک کرده و Open Table Definition را انتخاب کنید تا ساختار جدولی که EF برای شما ایجاد کرده را مشاهده نمایید.





نگاه کنید که چگونه schema ی جدول Movies به کلاس Movie که شما قبلا ایجاد کردید، نگاشت می شود.

EF Code First به صورت خودکار این schema را بر اساس کلاس Movie، برای شما ایجاد می کند.

پس از اینکه کار انجام شد، با راست کلیک بر روی MovieDBContext و انتخاب Close Connection، اتصال را قطع کنید (اگر اتصال را نبندید، ممکن است دفعه ی بعدی که پروژه را اجرا می کنید، با خطا مواجه شوید).

اکنون شما یک پایگاه داده به همراه صفحاتی دارید که می توان توسط آن ها اطلاعات و داده ها را نمایش داد، ویرایش کرد، بروز رسانی نمود و در نهایت حذف کرد. در مبحث آموزشی بعدی، باقی کدهای scaffold را مورد بررسی قرار داده و یک متد SearchIndex و همچنین یک SearchIndex view اضافه می کنیم که به شما اجازه ی جستجو movie ها در پایگاه داده را می دهد.