Code First

What is the .NET Entity Framework?

The Entity Framework **enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, without having to concern themselves with the underlying database tables and columns where this data is stored**.

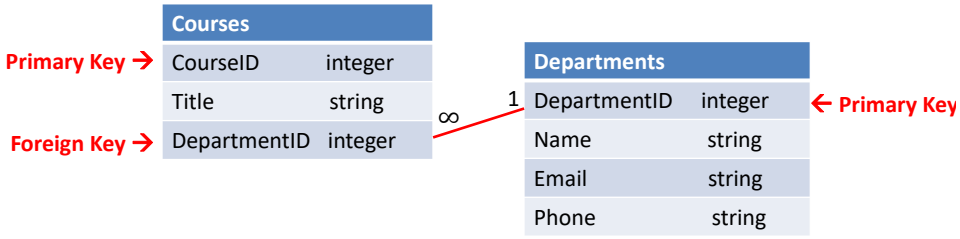
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview

The Entity Framework is involved whenever an ASP.NET MVC application accesses a database.

Writing a **Code First** application, however, **goes a step further** by asking Visual Studio and the Entity Framework to **create an entire application for you**. You need only to provide the model classes with appropriate navigation properties and validation if you want, create corresponding controllers for your entities, and if you choose the right options, an entire application (**along with a database**) will be created for you.

Let's say that we want to create an application that will allow us to view, add, edit, and delete **courses** along with their corresponding **departments**.

We would have a table called **Courses** and another one called **Departments**.



In order to have EntityFramework get everything working correctly automatically, we need to add some **navigation properties**.

Courses	
CourseID	integer
Title	string
DepartmentID	integer

∞ 1

Departments	
DepartmentID	integer
Name	string
Email	string
Phone	string

```
public class Course
 {
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int DepartmentID { get; set; }
    public virtual Department Department { get; set; }
 }
```

The Department property of a Course entity will hold the Department entity that is related to the Course.

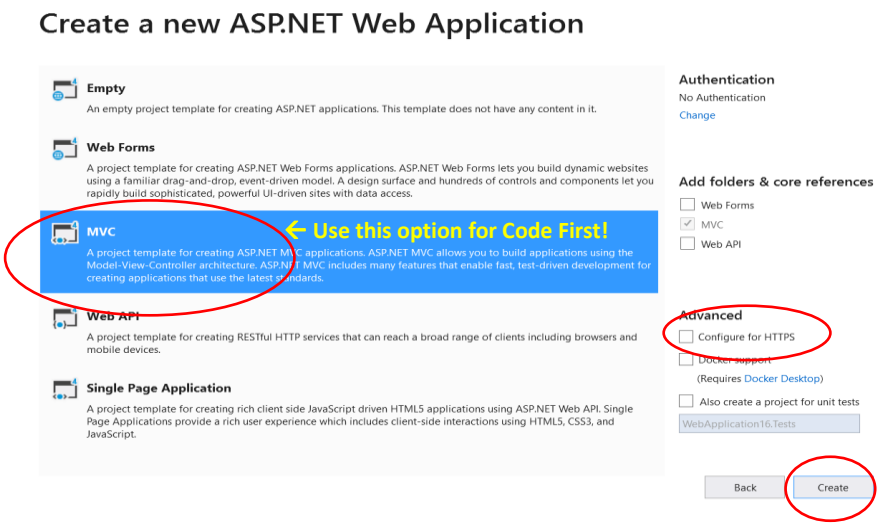
Getting your navigation properties right **is the key** to having Code-First work correctly!

```
public class Department
 {
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
 }
```

And the Courses entity of the Departments entity will hold all of the Course entities that are related to the Department.

Let's create our MVC application using Code First.

1. Create a new ASP.NET Web Application and choose the **MVC** project template.



2. Add a class for each of the entities. (Highlight the Modelfolder, right-click and "Add Class".)

Course.cs

Let's add some validation here just for fun. Be sure and add the using statement at the top:

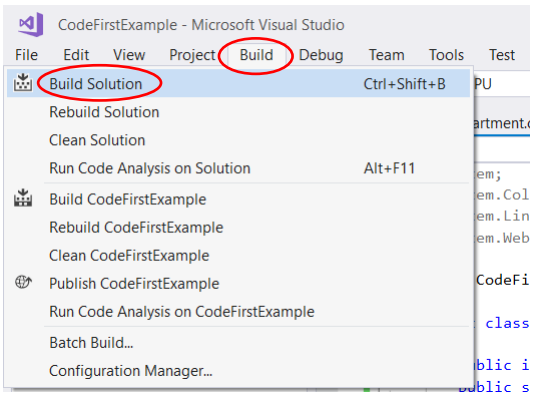
```
using System.ComponentModel.DataAnnotations;
```

```
public class Course
 {
    public int CourseID { get; set; }
    [Required(ErrorMessage = "Title is required.")]
    public string Title { get; set; }
    public int DepartmentID { get; set; }
    public virtual Department Department { get; set; }
 }
```

Department.cs

```
public class Department
 {
        public int DepartmentID { get; set; }
        public string Name { get; set; }
         [Required(ErrorMessage = "Email is required.")]
         [EmailAddress(ErrorMessage = "Please enter a valid email address.")]
          public string Email { get; set; }
          [Required(ErrorMessage = "Phone is required.")]
          [Phone(ErrorMessage = "Please enter a valid phone number.")]
          public string Phone { get; set; }
          public virtual ICollection<Course> Courses { get; set; }
 }
```

Build the solution before you continue with step 3 so that your project will know about these new classes you just created. If you forget, you will get an error that will remind you.



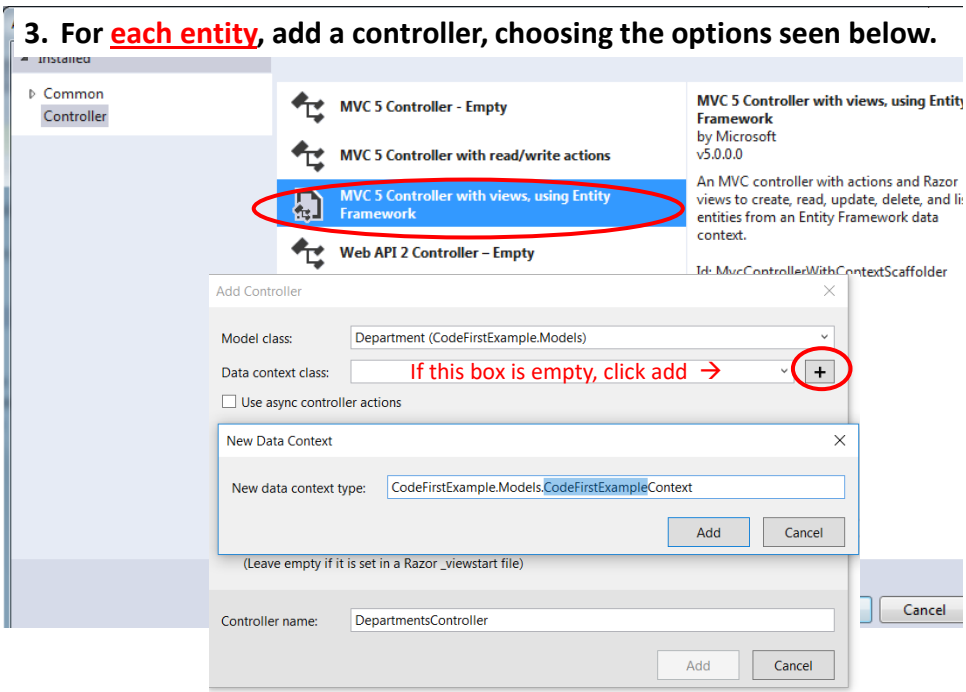
For each model class, you will need a corresponding controller.

Visual Studio will name the controller for you. In the case of the **Course** model class, Visual Studio will name the controller **CoursesController**.

In the case of the **Department** model class, Visual Studio will name the controller **DepartmentsController**.

To create a controller, right-click on the Controllers folder and choose "Add… New Item > Controller".

3. For each entity, add a controller, choosing the options seen below.



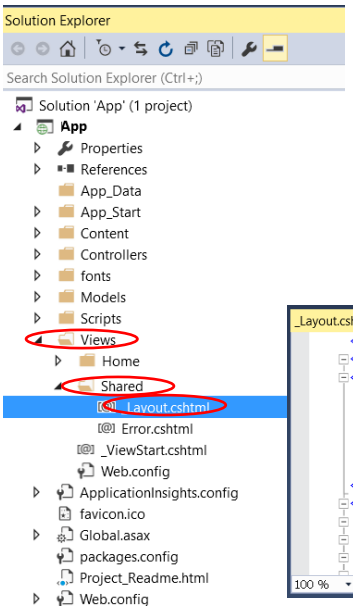
You will have to build the solution again before you create your second controller.

Notice that many action methods and views were created for you.

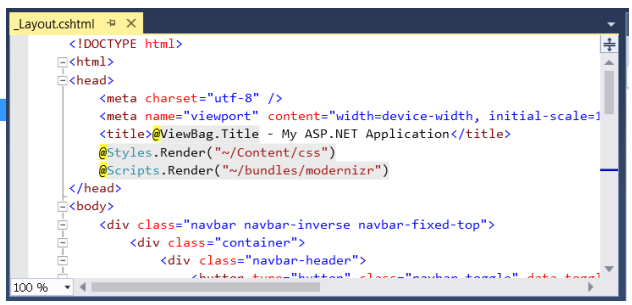
If you ran the application now, you would find that the database has been created for you! The database would be created based on how your entities are defined.

Let's not run it yet though!

Before we have the Entity Framework create the database for us, let's add some links to the menu on our "**Layout Page**".

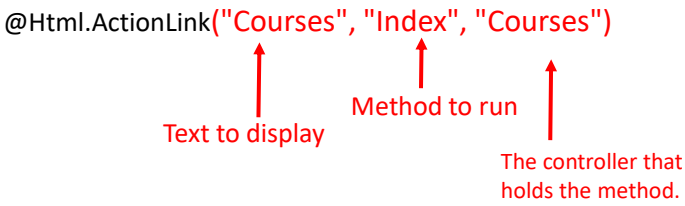


A layout page provides an overall container for pages on your site. For example, the layout page can contain the header, navigation area, and footer. The layout page includes a placeholder where the main content goes.
docs.microsoft.com

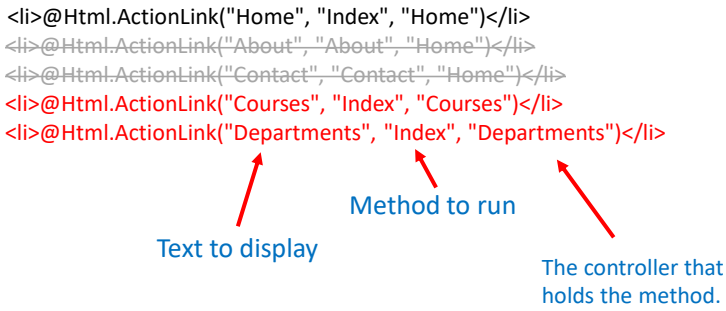


To add the links we need, we are going to use an **HTML Helper method**.

HTML Helper methods "help" us by creating the HTML code that is needed at runtime for a specific element. We don't have to remember the HTML tags, we just call the helper methods.



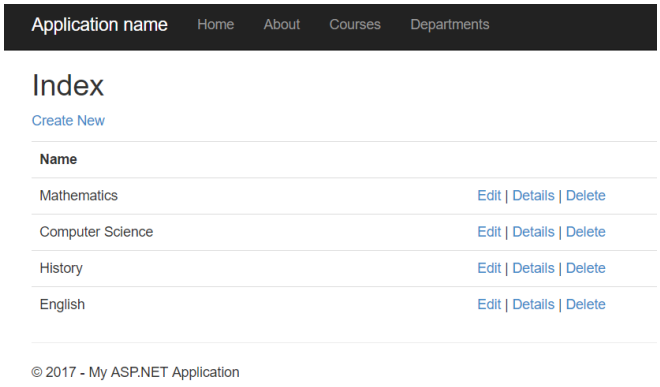
4. Add two links in _Layout.cshtml and take out any links that are unnecessary.



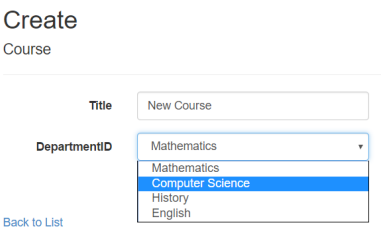
Now run the application.

Many views and action methods have been automatically create that will allow us to view, add, edit, and delete **courses** along with their corresponding **departments**.

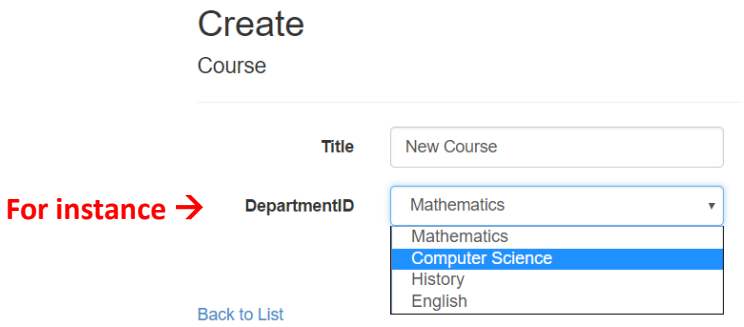
Click on Departments and add some departments. The database will be created automatically.



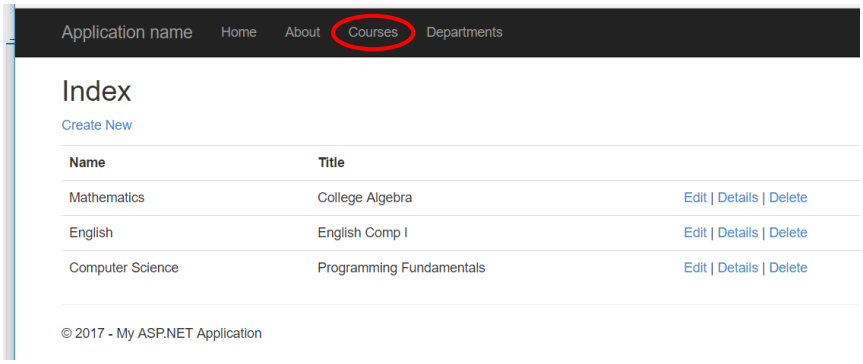
After creating some departments, create some courses.



Although the entire application has been created for you, you may notice some things that you want to change.

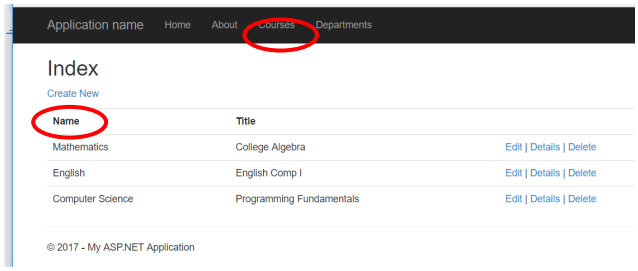


Another problem is that when you ask to see a list of courses, the first column is labeled Name but contains NOT the course name, but the DEPARTMENT name. The actual course name is in the second column which is not what we would expect to see.

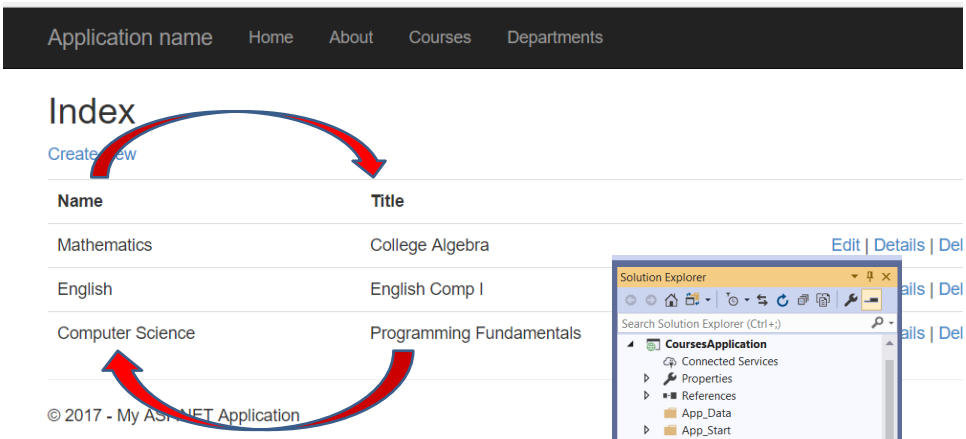


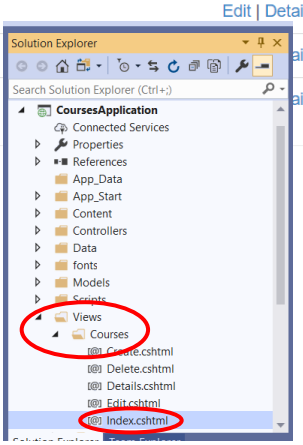
We would like the heading to say Department instead of Name.

Add the bold statements to your Department Model:




```
using System;
using System.Collections.Generic;
using System.ComponentModel;          ← We have to add this
using System.ComponentModel.DataAnnotations;
namespace CodeFirstExample.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }
        [DisplayName("Department")]
        public string Name { get; set; }
        public virtual ICollection<Course> Courses { get; set; }
    }
}
```





Now we need to go into the index view under Courses and change the order of the columns!

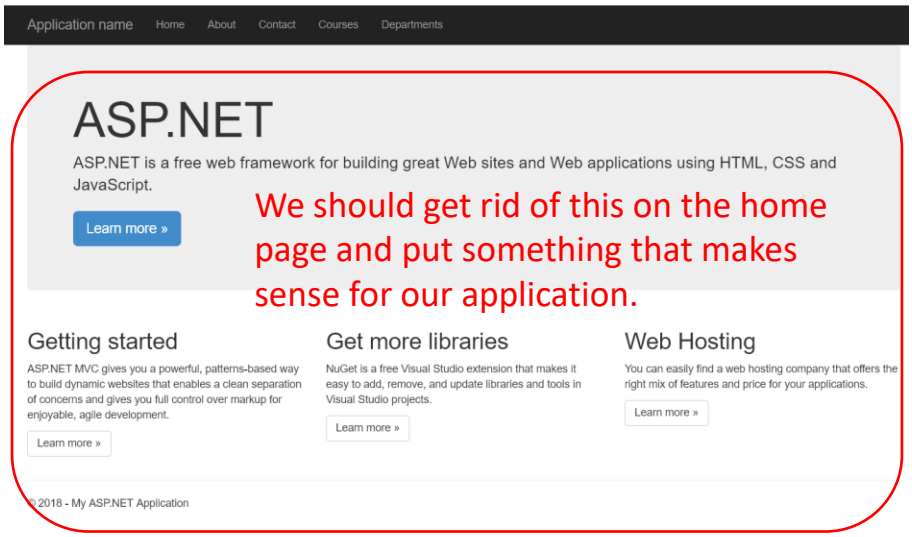
```
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Department.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Title)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Department.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.CourseID }) |
            @Html.ActionLink("Details", "Details", new { id=item.CourseID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.CourseID })
        </td>
    </tr>
}

</table>
```

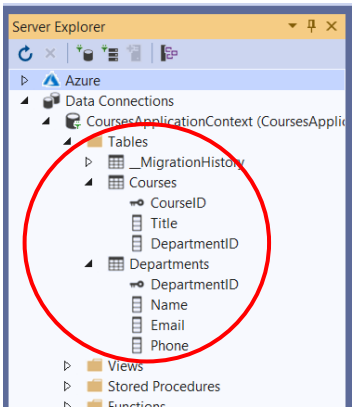
Switch

Switch

There are several other Courses views that need to be changed as well.

What else might we want to change?





The database was created for us!

We can see and access the database that was created for us by expanding the data connection over in the Server Explorer on the left.

Here are the steps to create a Code-First Application using ASP.NET MVC.

1. Add the model classes.
2. "Build" the solution.
3. Add the corresponding controllers. "Build" after the first two controllers are created.
4. Add some links on the layout page.
5. Add display names if necessary.
6. Change the column order if necessary.
7. Other changes within some of the views will also be necessary.