## **CRUD Operations using Entity Framework in ASP.NET MVC**

##### **Database Tables used in this Demo:**

In this demo, we are going to use the following **Department** and **Employee** table. So, please use the below SQL script to create and populate these 2 tables

**Create** **table** Department

(

Id int **primary key** identity,

Name nvarchar(50)

)

**Insert** **into** Department values('IT')

**Insert** **into** Department values('HR')

**Insert** **into** Department values('Payroll')

**Create** **table** Employee

(

EmployeeId int **Primary Key** Identity(1,1),

Name nvarchar(50),

Gender nvarchar(10),

City nvarchar(50),

Salary decimal(18,2),

DepartmentId int

)

**Alter** **table** Employee add **foreign key** (DepartmentId) references Department(Id)

**Insert** **into** Employee values('Mark','Male','London',1000,1)

**Insert** **into** Employee values('John','Male','Chennai',2000,3)

**Insert** **into** Employee values('Mary','Female','New York',3000,3)

**Insert** **into** Employee values('Mike','Male','Sydeny',4000,2)

**Insert** **into** Employee values('Scott','Male','London',3000,1)

**Insert** **into** Employee values('Pam','Female','Falls Church',2000,2)

**Insert** **into** Employee values('Todd','Male','Sydney',1000,1)

**Insert** **into** Employee values('Ben','Male','New Delhi',4000,2)

**Insert** **into** Employee values('Sara','Female','London',5000,1)

##### **Create a new ASP.NET MVC 5 Web application:**

Open **File => New =>Project** and then select **Installed => Templates => Visual C#.** Then select **ASP.NET Web Application**. Provide the project name and location where you want to save your application and finally click on the **OK** button as shown in the below image.

CRUD Operations using Entity Framework in ASP.NET MVC

From the next screen that is from select a template screen, select **Empty** as the project template. From add folder and core reference section check the **MVC** checkbox and click on **OK** as shown below.

Selecting Project Template in ASP.NET MVC Application

Once you click on the OK button, it will take some time to create the project for us.

##### **Adding ADO.NET Entity Data Model**

Right-click on Models folder then select **Add => New Item** from the context menu that will open the **Add New Item** window. Select the **Data** tab from the left panel and then choose **ADO.NET Entity Data Model** from the middle panel. Provide a meaningful name for your data model and click on the **Add** button as shown in the below image.

Adding ADO.NET Entity Data Model in ASP.NET MVC Application

##### **Selecting The Entity Framework Approach to interact with the database.**

Once you click on the Add button it will ask you to choose the entity framework approach. Here, we are going to use the database first approach as we already created the required database tables. So, from the Entity Data Model Wizard, select “**Generate from database**” option and click “**Next**” as shown below.

Selecting Database First Approach in Entity Framework

##### **Creating Database Connection:**

On “**Choose your data connection screen**” and click on the “**New Connection**” button which will open the connection properties window. Here, we are going to interact with the SQL Server database. So, from the Data Source select **Microsoft SQL Server (SqlClient)**. Provide your SQL Server name. Choose the Authentication type. Here, I am choosing **Windows Authentication**. Then select the database to which you are going to interact from the **select or enter a database name** drop-down list. Finally, click on the “**OK**” as shown below.

Creating Database Connection in Entity Framework Database First Approach

Then provide a meaningful connection string name such as “**EmployeeDBContext**” and click on the “**Next**” button as shown below.

CRUD Operations using Entity Framework in ASP.NET MVC

##### **Choose Entity Framework Version:**

Here, we are going to use Entity Framework 6. So, from the Choose Your Version screen, choose the **Entity framework 6.x**  and click on the **Next** button as shown below.

Selecting Entity Framework Version in ASP.NET MVC Application

##### **Selecting Database Objects and Settings:**

On “**Choose your database objects and Settings**” screen, expand the “**Tables**” and then select “**Department**” and “**Employee**” tables. Set Model Namespace as **Models** and click on the “**Finish**” button as shown in the below image.

Selecting Database Objects and Settings in Entity Framework

At this point, we should have Departmentand Employee entities generated as shown below.

CRUD Operations using Entity Framework in ASP.NET MVC Application

##### **Creating MVC 5 Controller:**

Right-click on the “**Controllers**” folder and select **Add – Controller** from the context menu. Then select **MVC 5 Controller with views, using Entity Framework** and click on the **Add** button as shown in the below image.

Creating MVC 5 Controller using Entity Framework

**On the next screen set the below details**

1. **Model class = Employee (CRUD\_Using\_EF.Models)**
2. **Data Context Class = EmployeeDBContext(CRUD\_Using\_EF.Models)**
3. **Controller Name = EmployeeController**
4. **Rest values are as it is and click on the Add button as shown in the below image**

**Setting Controller Properties to Create Views and Action Methods**

At this point, we should have the following files automatically added.

**EmployeeController.cs** file in “**Controllers**” folder. **Index, Create, Edit, Detail and Delete** views in the “**Employee**” folder which is inside the Views folder.

##### **Below is the EmployeeController code**

**namespace** *CRUD\_Using\_EF.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

**private** EmployeeDBContext db = new EmployeeDBContext**()**;

// GET: Employee

**public** ActionResult Index**()**

**{**

var employees = db.Employees.Include**(**e =**>** e.Department**)**;

**return** View**(**employees.ToList**())**;

**}**

// GET: Employee/Details/5

**public** ActionResult Details**(int**? id**)**

**{**

**if** **(**id == **null)**

**{**

**return** new HttpStatusCodeResult**(**HttpStatusCode.BadRequest**)**;

**}**

Employee employee = db.Employees.Find**(**id**)**;

**if** **(**employee == **null)**

**{**

**return** HttpNotFound**()**;

**}**

**return** View**(**employee**)**;

**}**

// GET: Employee/Create

**public** ActionResult Create**()**

**{**

ViewBag.DepartmentId = new SelectList**(**db.Departments, "Id", "Name"**)**;

**return** View**()**;

**}**

// POST: Employee/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

**[**HttpPost**]**

**[**ValidateAntiForgeryToken**]**

**public** ActionResult Create**([**Bind**(**Include = "EmployeeId,Name,Gender,City,Salary,DepartmentId"**)]** Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

db.Employees.Add**(**employee**)**;

db.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

ViewBag.DepartmentId = new SelectList**(**db.Departments, "Id", "Name", employee.DepartmentId**)**;

**return** View**(**employee**)**;

**}**

// GET: Employee/Edit/5

**public** ActionResult Edit**(int**? id**)**

**{**

**if** **(**id == **null)**

**{**

**return** new HttpStatusCodeResult**(**HttpStatusCode.BadRequest**)**;

**}**

Employee employee = db.Employees.Find**(**id**)**;

**if** **(**employee == **null)**

**{**

**return** HttpNotFound**()**;

**}**

ViewBag.DepartmentId = new SelectList**(**db.Departments, "Id", "Name", employee.DepartmentId**)**;

**return** View**(**employee**)**;

**}**

// POST: Employee/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

**[**HttpPost**]**

**[**ValidateAntiForgeryToken**]**

**public** ActionResult Edit**([**Bind**(**Include = "EmployeeId,Name,Gender,City,Salary,DepartmentId"**)]** Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

db.Entry**(**employee**)**.State = EntityState.Modified;

db.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

ViewBag.DepartmentId = new SelectList**(**db.Departments, "Id", "Name", employee.DepartmentId**)**;

**return** View**(**employee**)**;

**}**

// GET: Employee/Delete/5

**public** ActionResult Delete**(int**? id**)**

**{**

**if** **(**id == **null)**

**{**

**return** new HttpStatusCodeResult**(**HttpStatusCode.BadRequest**)**;

**}**

Employee employee = db.Employees.Find**(**id**)**;

**if** **(**employee == **null)**

**{**

**return** HttpNotFound**()**;

**}**

**return** View**(**employee**)**;

**}**

// POST: Employee/Delete/5

**[**HttpPost, ActionName**(**"Delete"**)]**

**[**ValidateAntiForgeryToken**]**

**public** ActionResult DeleteConfirmed**(int** id**)**

**{**

Employee employee = db.Employees.Find**(**id**)**;

db.Employees.Remove**(**employee**)**;

db.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

**protected** **override** **void** Dispose**(bool** disposing**)**

**{**

**if** **(**disposing**)**

**{**

db.Dispose**()**;

**}**

**base**.Dispose**(**disposing**)**;

**}**

**}**

**}**

At this point if you run the application you will get an error stating – **The resource cannot be found**. This is because by default the application goes to the “**Home**” controller and “**Index**” action. To fix this, open “**RouteConfig.cs**” file from the “App\_Start” folder and set the controller as “**Employee**” as shown below.

RouteConfig File in ASP.NET MVC Application

Run the application again. Notice that all the employees are listed on the index view. We can also create a new employee, edit an employee, view their full details and delete an employee as well. However, there are a few issues with each of the views which we will address in our upcoming articles.

## **Customizing Auto-Generated Index and Create View**

In this article, I am going to discuss **customizing the auto-generated index and create views** in the ASP.NET MVC application. Please read our previous article before proceeding to this article as this article is a continuation part of our previous article. In our previous article, we discussed the step-by-step procedure to perform [**CRUD operations using Entity Framework**](https://dotnettutorials.net/lesson/crud-operations-using-entity-framework/) in ASP.NET MVC Application.

##### **Customizing Index View:**

Run the application and navigate to the index view which should display the data as shown below.

Customizing Auto Generated Index View in ASP.NET MVC Application

As shown in the above “**Index**” view, it is using “**Name**” as the column header for both **employee** and **department** name. This is because the “**Name**” column is used in both the database tables (**Employee and Department**) and the entity framework used these column names to generate the “**Name**” property in both **Employee and Department** classes that are auto-generated as shown below.

###### **Employee.cs File**

**namespace** *CRUD\_Using\_EF.Models*

**{**

**using** System;

**using** System.Collections.Generic;

**public** **partial** **class** Employee

**{**

**public** **int** EmployeeId **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **string** Gender **{** **get**; **set**; **}**

**public** **string** City **{** **get**; **set**; **}**

**public** Nullable**<decimal>** Salary **{** **get**; **set**; **}**

**public** Nullable**<int>** DepartmentId **{** **get**; **set**; **}**

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

**}**

**}**

###### **Department.cs File**

**namespace** *CRUD\_Using\_EF.Models*

**{**

**using** System;

**using** System.Collections.Generic;

**public** **partial** **class** Department

**{**

**[**System.Diagnostics.CodeAnalysis.SuppressMessage**(**"Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors"**)]**

**public** Department**()**

**{**

this.Employees = new HashSet**<**Employee**>()**;

**}**

**public** **int** Id **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**[**System.Diagnostics.CodeAnalysis.SuppressMessage**(**"Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly"**)]**

**public** **virtual** ICollection**<**Employee**>** Employees **{** **get**; **set**; **}**

**}**

**}**

Here, we need to change the department column header to “**Department Name**” instead of just “**Name**“. In order to do this add a class file with the name **PDepartment.cs** within the “**Models**” folder. Once you created the **PDepartment.cs** class file then simply copy and paste the following code in it.

**namespace** *CRUD\_Using\_EF.Models*

**{**

**[**MetadataType**(**typeof**(**DepartmentMetaData**))]**

**public** **partial** **class** Department

**{**

**}**

**public** **class** DepartmentMetaData

**{**

**[**Display**(**Name = "Department Name"**)]**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**}**

With the above changes in place, run the application and notice the column name is displayed as Department Name. This is achieved by using the “**Display**” attribute that is present in the “**System.ComponentModel.DataAnnotations**” namespace.

If you are wondering why can’t we apply the “Display” attribute directly to the auto-generated “**Department**” class instead of creating another partial “**Department**” and **DepartmentMetaData** class. We can do it. There is nothing stopping us from doing it but every time the **Department** class is auto-generated our custom changes will be lost. This is the reason for creating another partial class and applying our changes.

##### **Customizing the auto-generated create view.**

At the moment none of the fields on the “**Create**” view are required. That means when we click on the “**Create**” button without filling out any data NULL values are stored in all the columns of the Employee table.

##### **How to make these fields on the “Create” view required?**

Add **[Required]** attribute to the properties of the “**Employee**” class. The “**Employee**” class that is present in “**EmployeeDataModel.Designer.cs**” is auto-generated by the entity framework. There is no point in adding the [**Required**] attribute to this class as we will lose the changes if the class is auto-generated again.

To achieve this add a class file with the name **PEmployee.cs** within the “**Models**” folder. Once you created the **PEmployee.cs** class file, then simply copy and paste the following code into it.

**using** *System.ComponentModel.DataAnnotations;*

**namespace** *CRUD\_Using\_EF.Models*

**{**

**[**MetadataType**(**typeof**(**EmployeeMetaData**))]**

**public** **partial** **class** Employee

**{**

**}**

**public** **class** EmployeeMetaData

**{**

**[**Required**]**

**public** **string** Name **{** **get**; **set**; **}**

**[**Required**]**

**public** **string** Gender **{** **get**; **set**; **}**

**[**Required**]**

**public** **string** City **{** **get**; **set**; **}**

**[**Required**]**

**public** **string** Salary **{** **get**; **set**; **}**

**[**Required**]**

**public** **int** DepartmentId **{** **get**; **set**; **}**

**}**

**}**

At this point run the application and click on the “**Create**” button without filling out any data. Notice that we get validation error messages as expected as shown in the below image.

Customizing Auto Generated Create View in ASP.NET MVC Application

If you want “**Select Department**” as the first item in the “**Department**” dropdownlist on “**Create**” view then Replace the following code in the Create.cshtml view.

**@Html.DropDownList(“DepartmentId”,null , htmlAttributes: new { @class = “form-control” })**with  
**@Html.DropDownList(“DepartmentId”,null , “Select Department”, htmlAttributes: new { @class = “form-control” })**

Notice that a textbox is used for gender. It is an ideal situation to have a drop down list for gender rather than a textbox. To achieve this make the following changes to the “**Create.cshtml**” view. Replace the following code.

**@Html.EditorFor(model => model.Gender, new { htmlAttributes = new { @class = “form-control” } })**with  
**@Html.DropDownList(“Gender”, new List<SelectListItem>  
{  
new SelectListItem { Text = “Male”, Value=”Male” },  
new SelectListItem { Text = “Female”, Value=”Female” }  
}, “Select Gender”, new { @class = “form-control” })**

##### **Below is the complete code for Create.cshtml view**

@model CRUD\_Using\_EF.Models.Employee

@{

ViewBag.Title = "Create";

}

**<h2>**Create**</h2>**

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

**<div** class="form-horizontal"**>**

**<h4>**Employee**</h4>**

**<hr** **/>**

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

**<div** class="form-group"**>**

@Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })

**<div** class="col-md-10"**>**

@Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })

**</div>**

**</div>**

**<div** class="form-group"**>**

@Html.LabelFor(model => model.Gender, htmlAttributes: new { @class = "control-label col-md-2" })

**<div** class="col-md-10"**>**

@Html.DropDownList("Gender", new List**<SelectListItem>**

{

new SelectListItem { Text = "Male", Value="Male" },

new SelectListItem { Text = "Female", Value="Female" }

}, "Select Gender", new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.Gender, "", new { @class = "text-danger" })

**</div>**

**</div>**

**<div** class="form-group"**>**

@Html.LabelFor(model => model.City, htmlAttributes: new { @class = "control-label col-md-2" })

**<div** class="col-md-10"**>**

@Html.EditorFor(model => model.City, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.City, "", new { @class = "text-danger" })

**</div>**

**</div>**

**<div** class="form-group"**>**

@Html.LabelFor(model => model.Salary, htmlAttributes: new { @class = "control-label col-md-2" })

**<div** class="col-md-10"**>**

@Html.EditorFor(model => model.Salary, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Salary, "", new { @class = "text-danger" })

**</div>**

**</div>**

**<div** class="form-group"**>**

@Html.LabelFor(model => model.DepartmentId, "DepartmentId", htmlAttributes: new { @class = "control-label col-md-2" })

**<div** class="col-md-10"**>**

@Html.DropDownList("DepartmentId",null , "Select Department", htmlAttributes: new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.DepartmentId, "", new { @class = "text-danger" })

**</div>**

**</div>**

**<div** class="form-group"**>**

**<div** class="col-md-offset-2 col-md-10"**>**

**<input** type="submit" value="Create" class="btn btn-default" **/>**

**</div>**

**</div>**

**</div>**

}

**<div>**

@Html.ActionLink("Back to List", "Index")

**</div>**

**<script** src="~/Scripts/jquery-1.10.2.min.js"**></script>**

**<script** src="~/Scripts/jquery.validate.min.js"**></script>**

**<script** src="~/Scripts/jquery.validate.unobtrusive.min.js"**></script>**

In the next article, I am going to discuss how to **customize the auto-generated Edit view** in the ASP.NET MVC application. Here, in this article, I try to explain customizing the auto-generated index and create views in the ASP.NET MVC application. I hope this article will help you with your needs.