Introduction

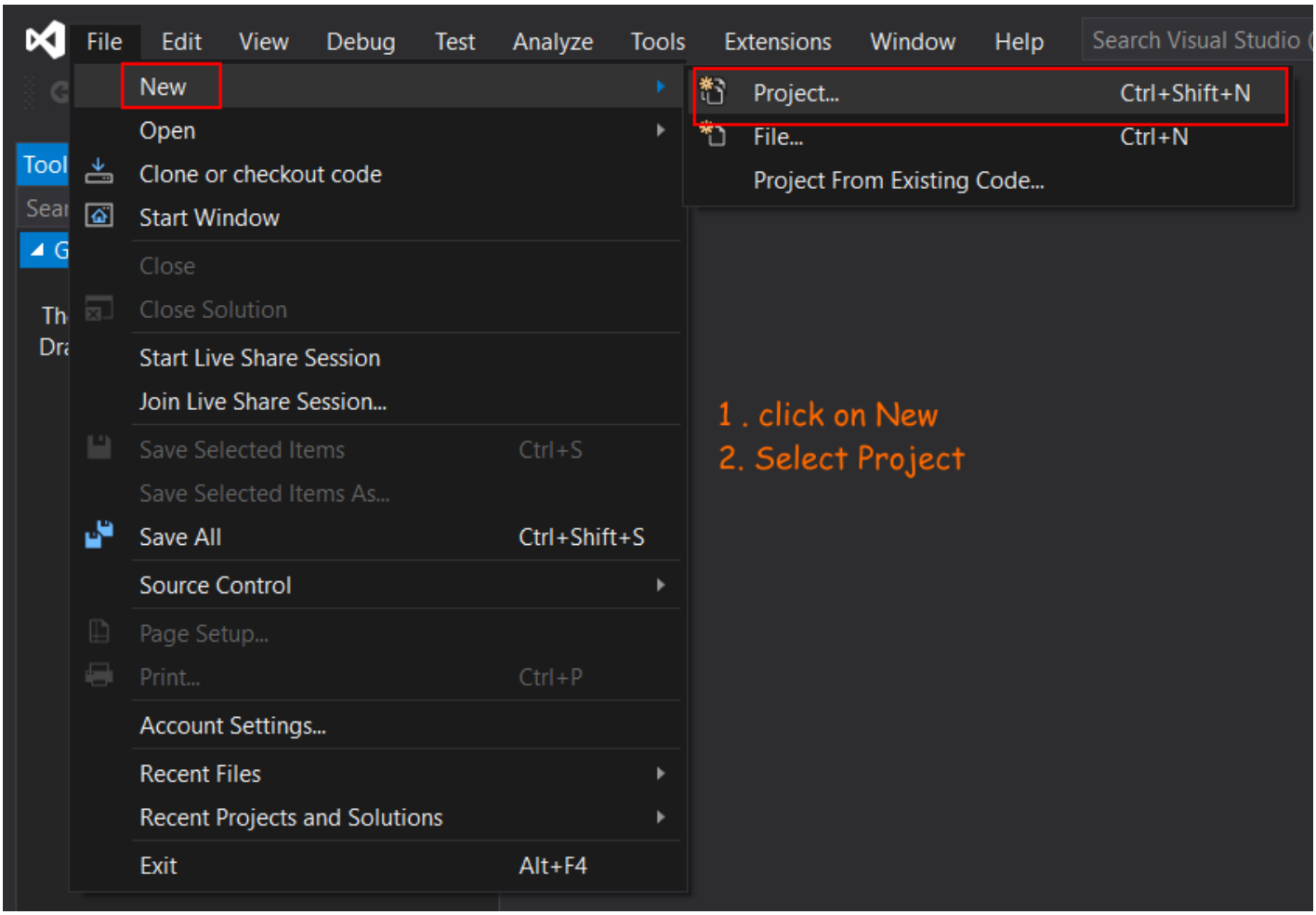
In this article I will explain how we can make CRUD operations in ASP.Net MVC. In this article we will use a data base first approach and perform CRUD operations using entity framework. I hope you like this article and get some information from it.

Step 1

Open Visual Studio. Here I use Visual Studio 2019, you can use any one as your system.

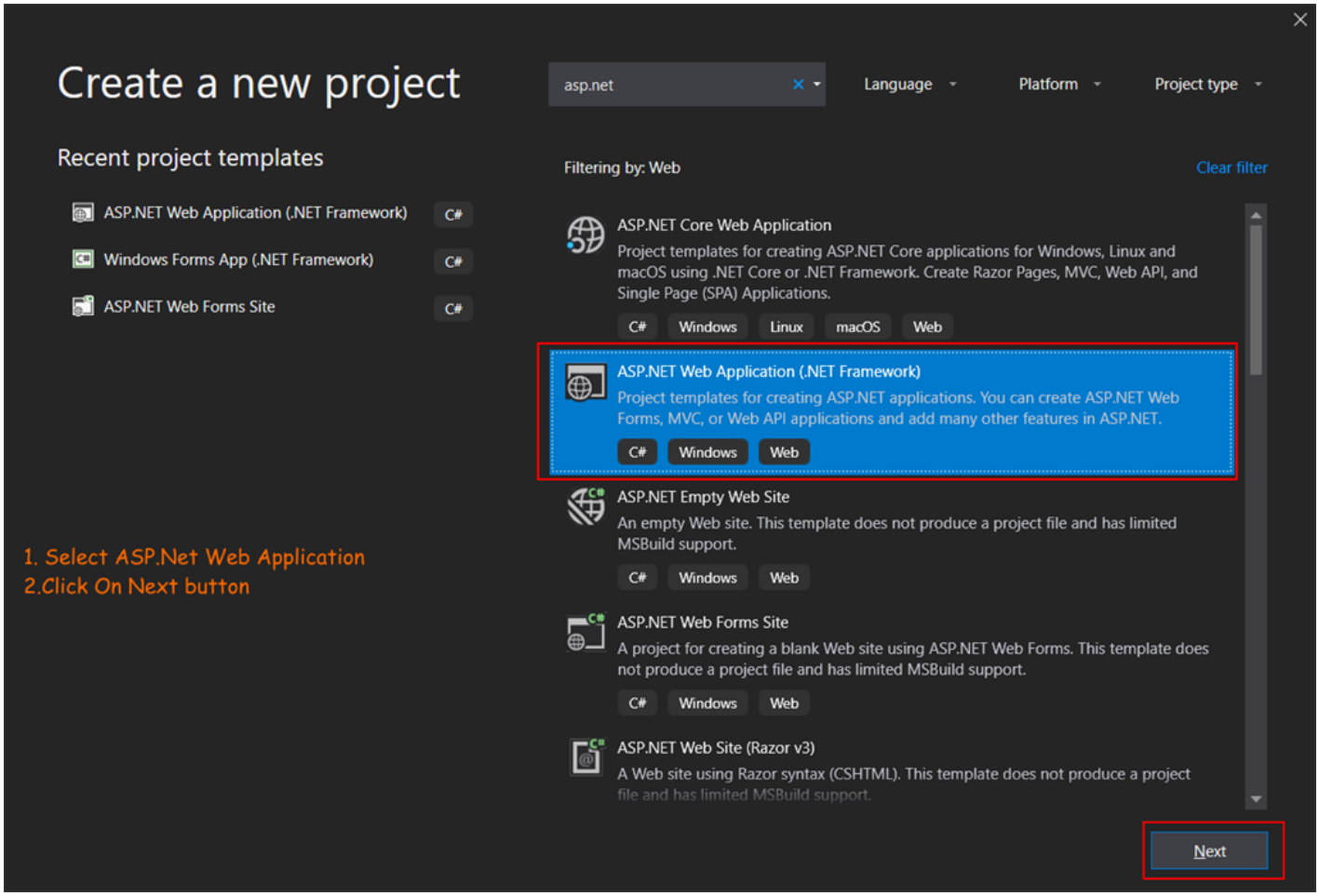
Step 2

Create a new project by clicking on File>New>Project.



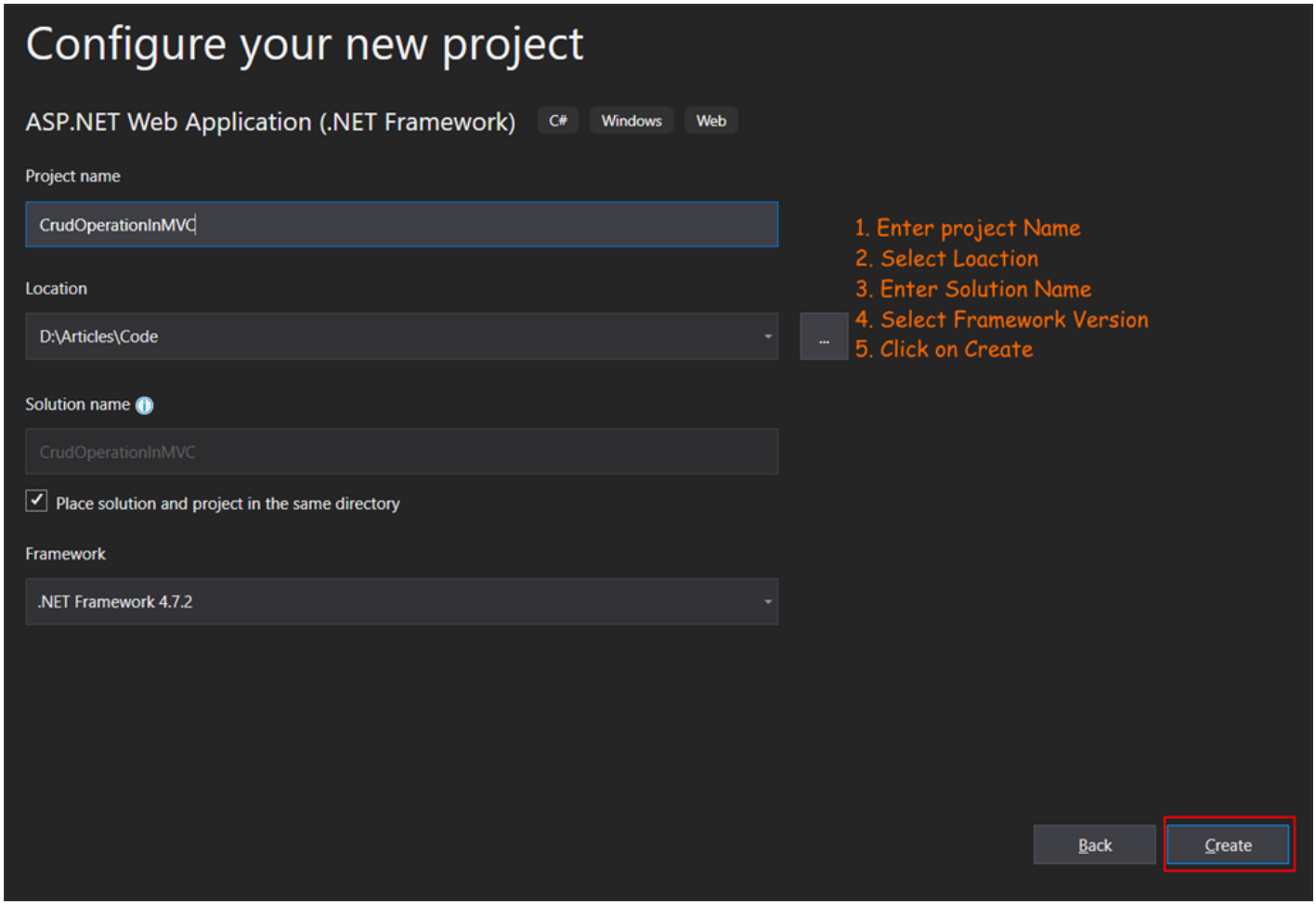
Step 3

Select Asp.Net Web Application and click on next button.



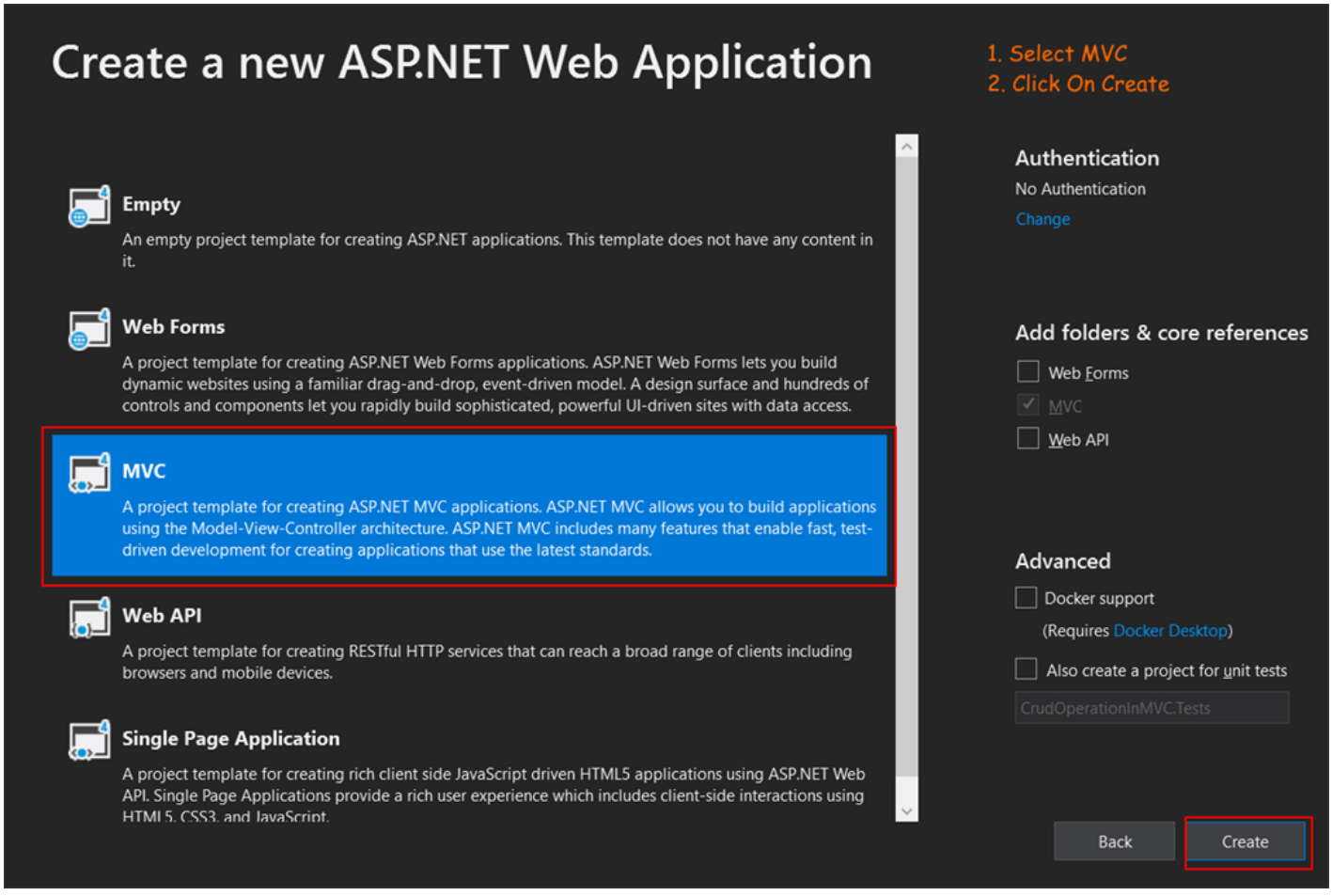
Step 4

In the next screen you need to enter a few details like your project name, project location where you want to save your project, solution name and .Net Framework version. After entering all details click on Create button.

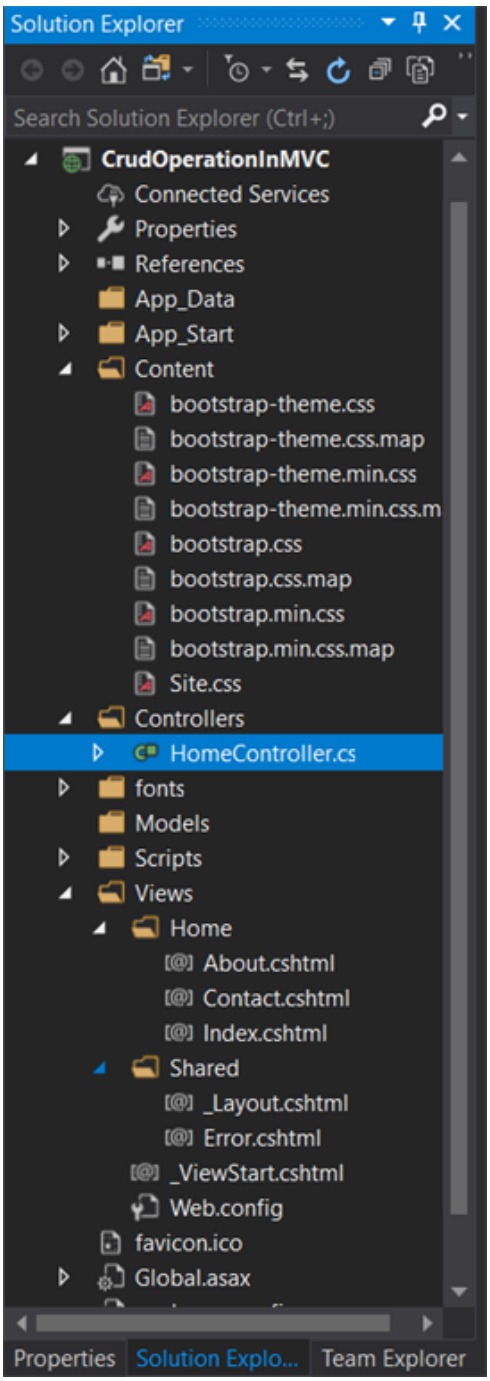


Step 5

In Next select MVC project and click on Create button.



Now your project is ready and you can see project structure in below image. Here I deleted all view from home controller because we created a new for our requirement.



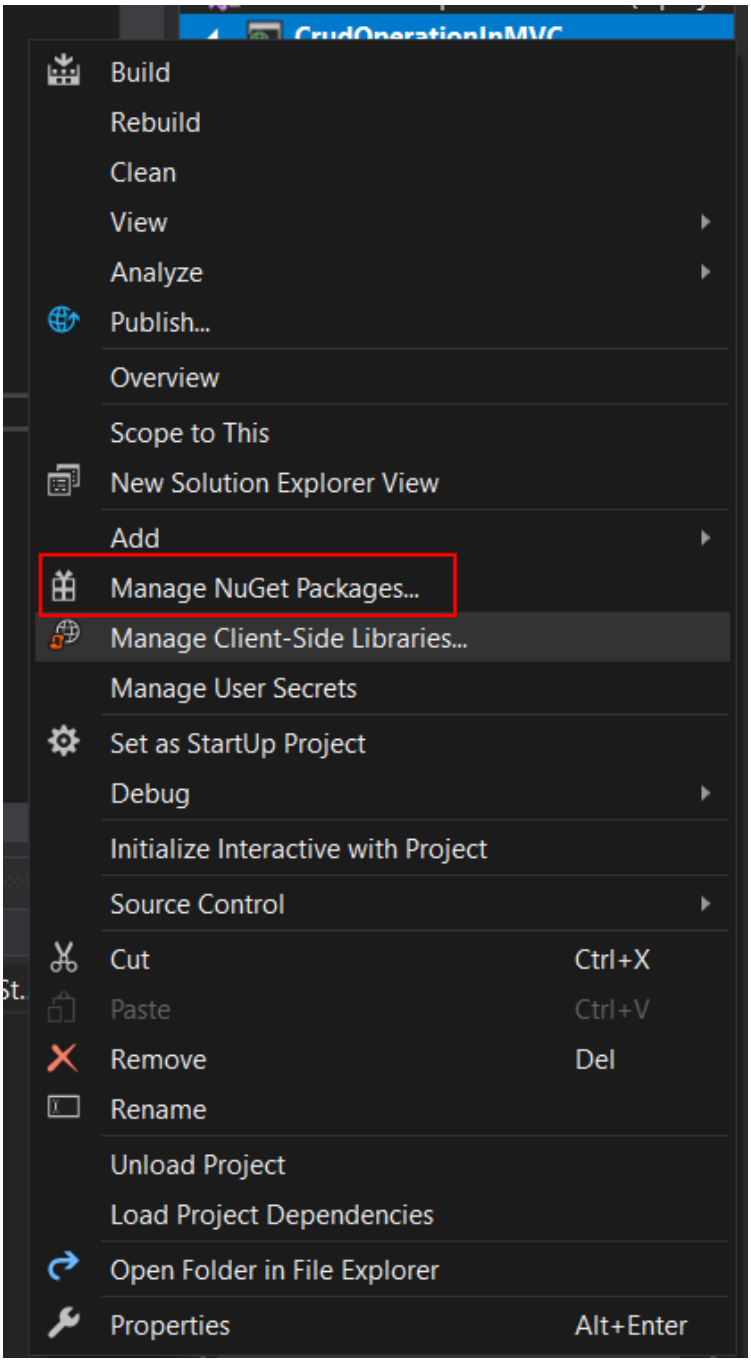
Step 6

Now in this project we will use database first approach so we need to create a database and a table. Here I created a database name with Tutorials and table Employee with the following columns as you can see in the below image.

Column Name	Data Type	Allow Nulls
EmployeeId	int	☐
EmployeeName	nvarchar(50)	☑
EmployeeSalary	numeric(10, 0)	☑
EmployeeCity	nvarchar(20)	☑

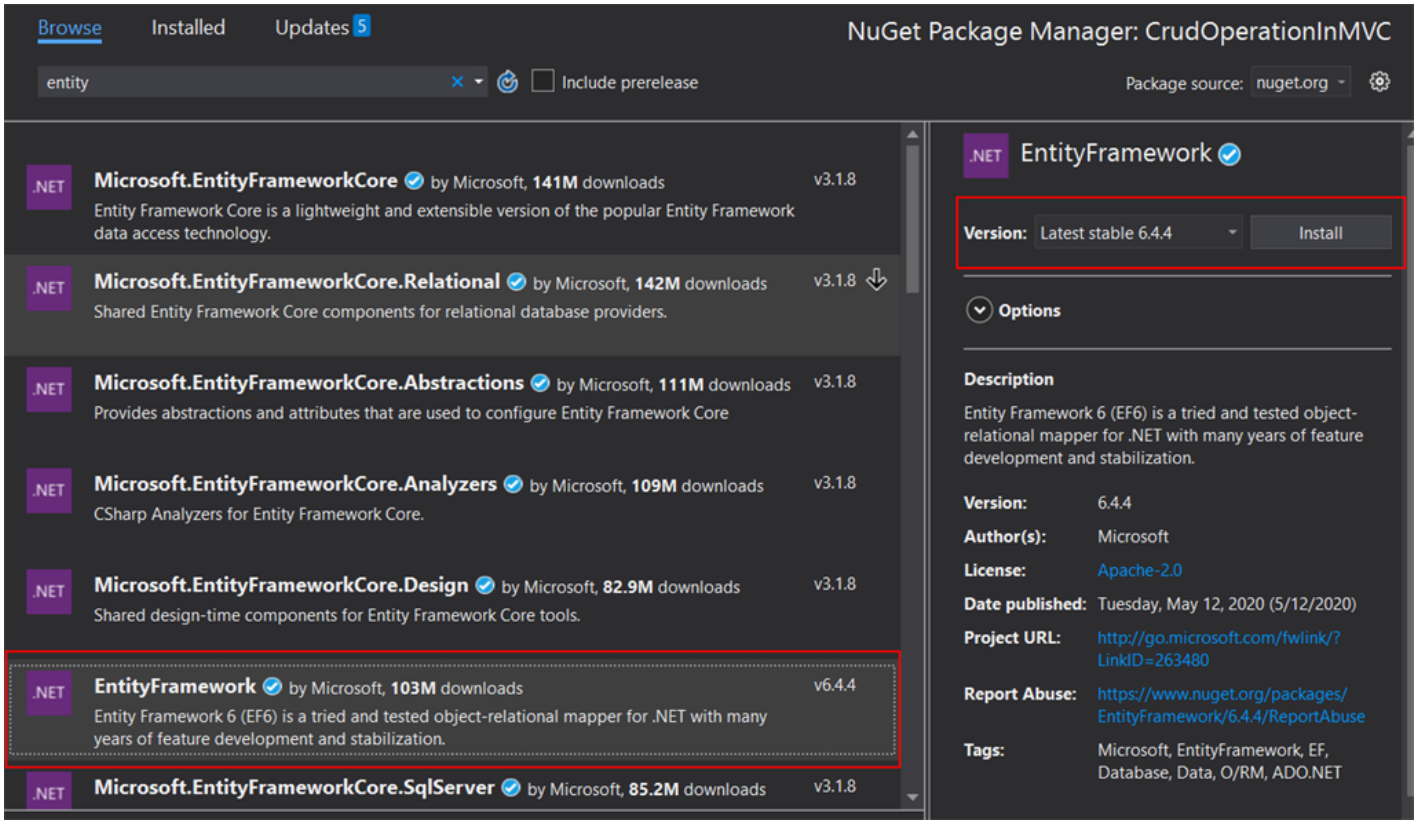
Step 7

Now we need to add Entity Framework in our project for performing some operation. For adding Entity Framework click on project name in solution explore then click on Manage NuGet Packages.



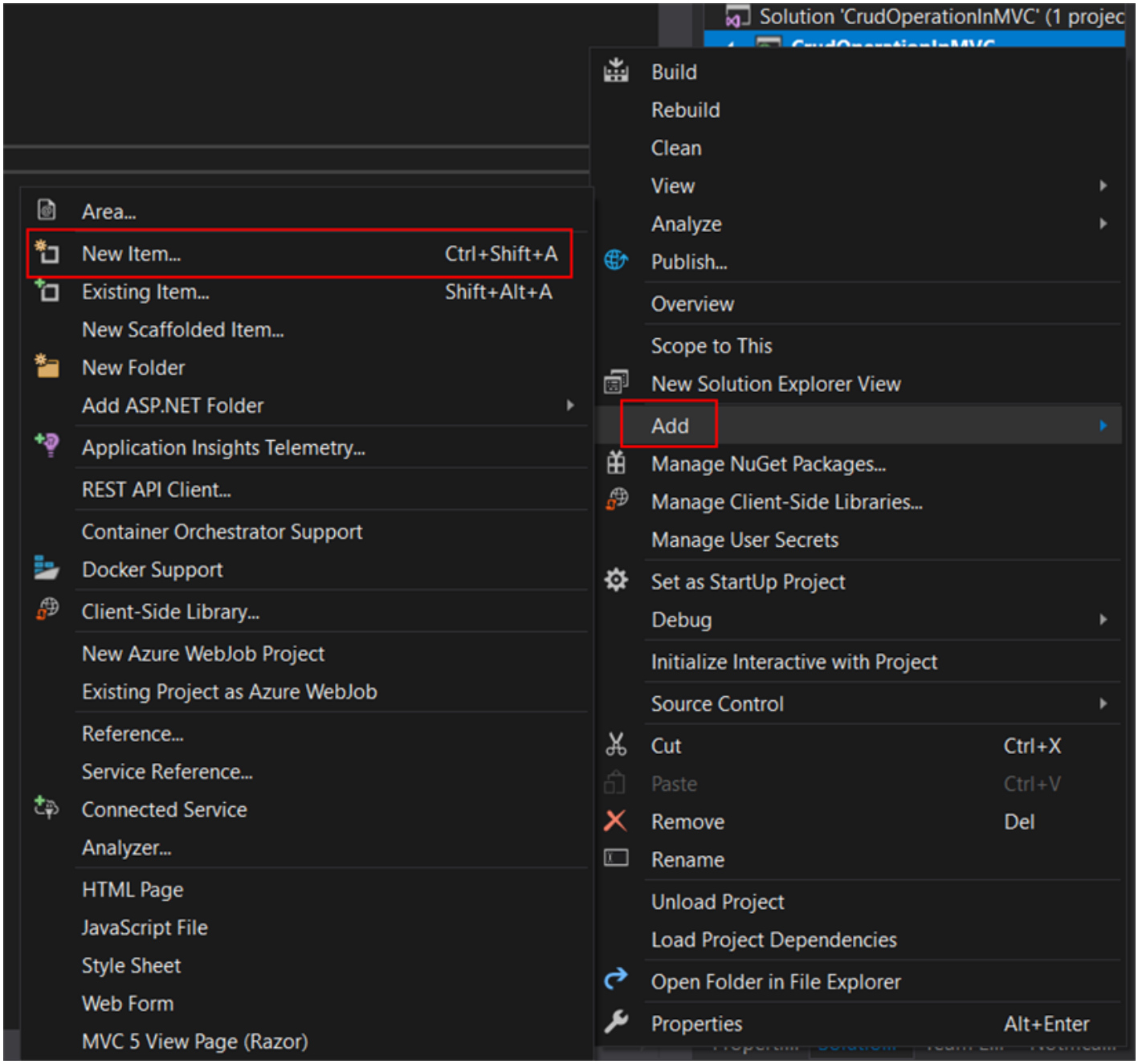
Step 8

Now you see a window with many NuGet Packages. Search for Entity framework and click on install button and after a few seconds Entity Framework is installed in your project.



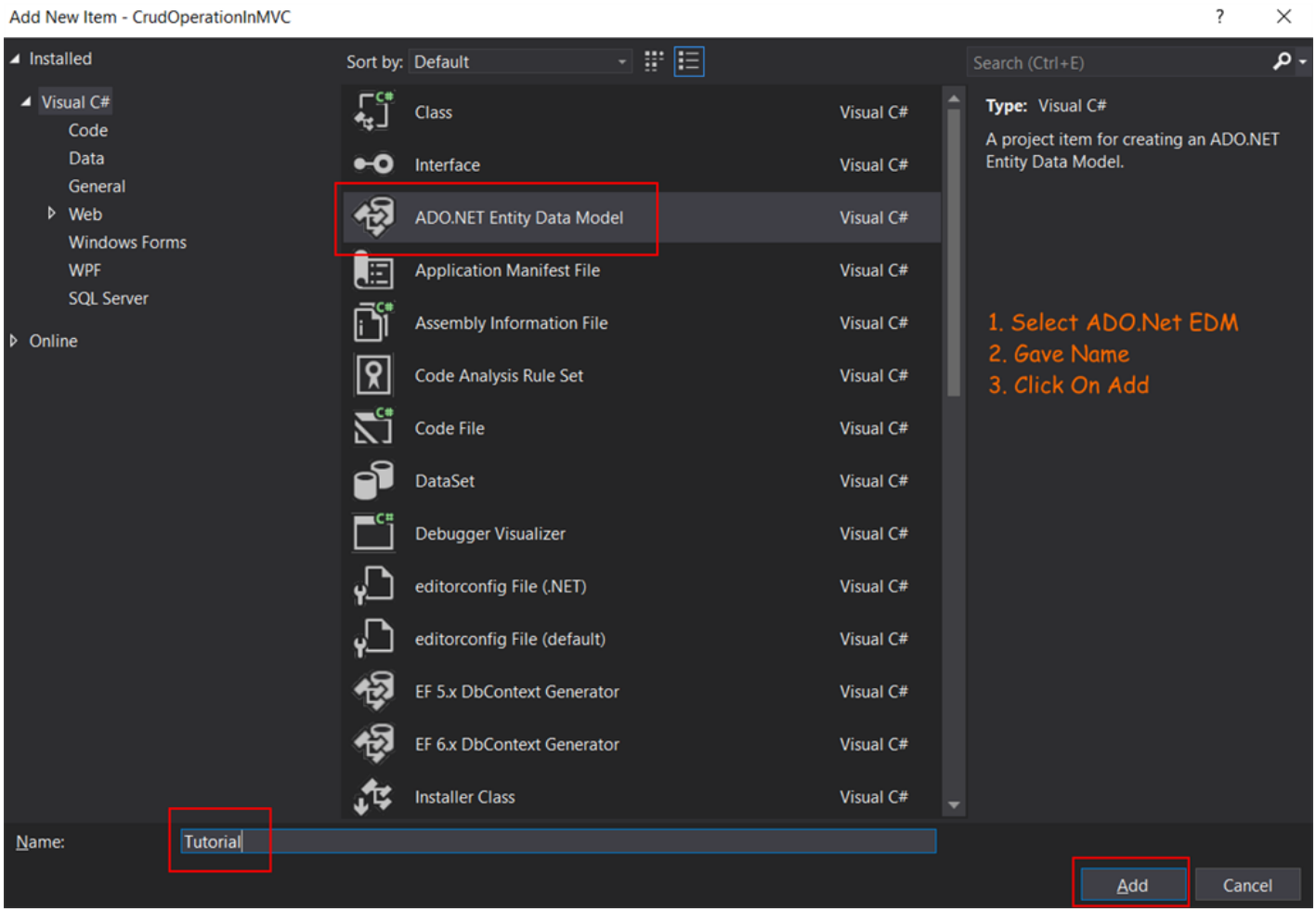
Step 9

Now we need to add ADO.Net Entity Data Model in our project. For adding ADO.Net Entity Data Model click on your project name then click on Add and then click on new Item or you can use short cut key Ctrl+Shift+A .



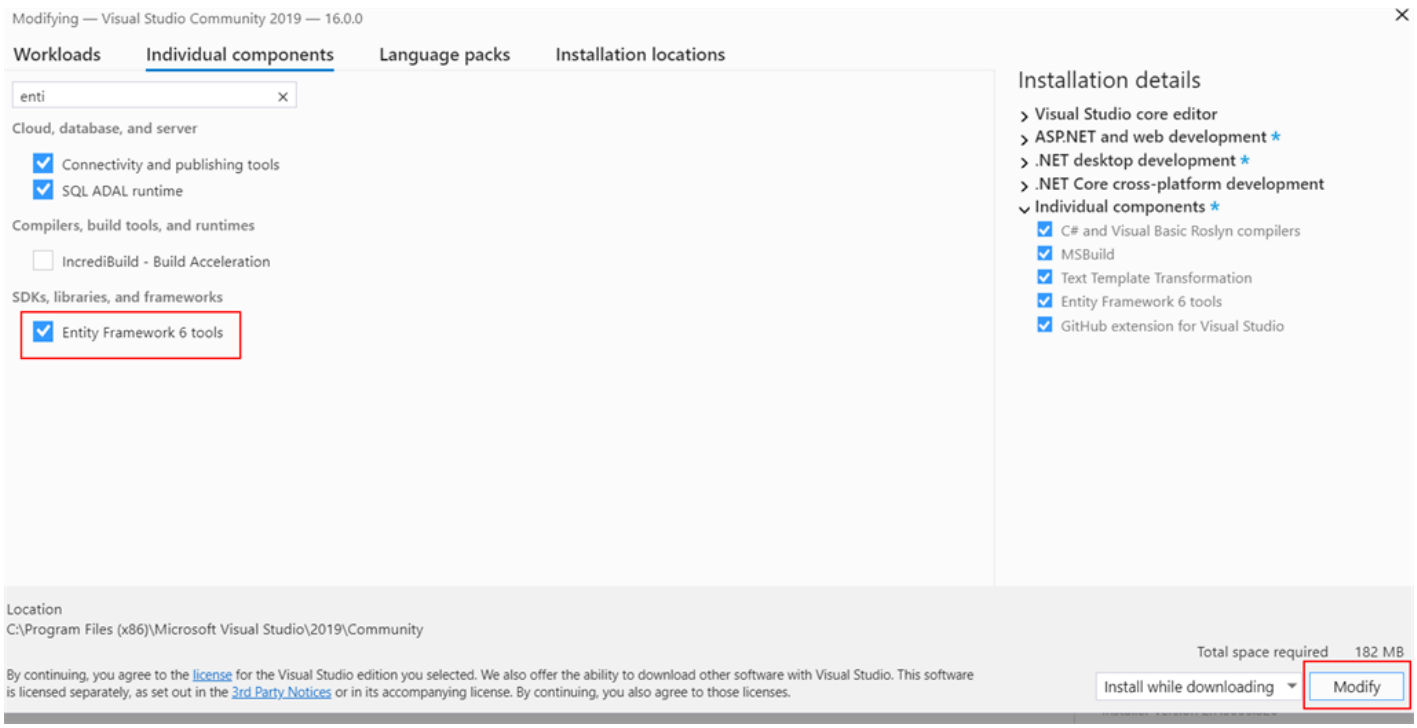
Step 10

Now you see a popup window. Select ADO.Net Entity Data Model, give appropriate name and click on add button.



Note

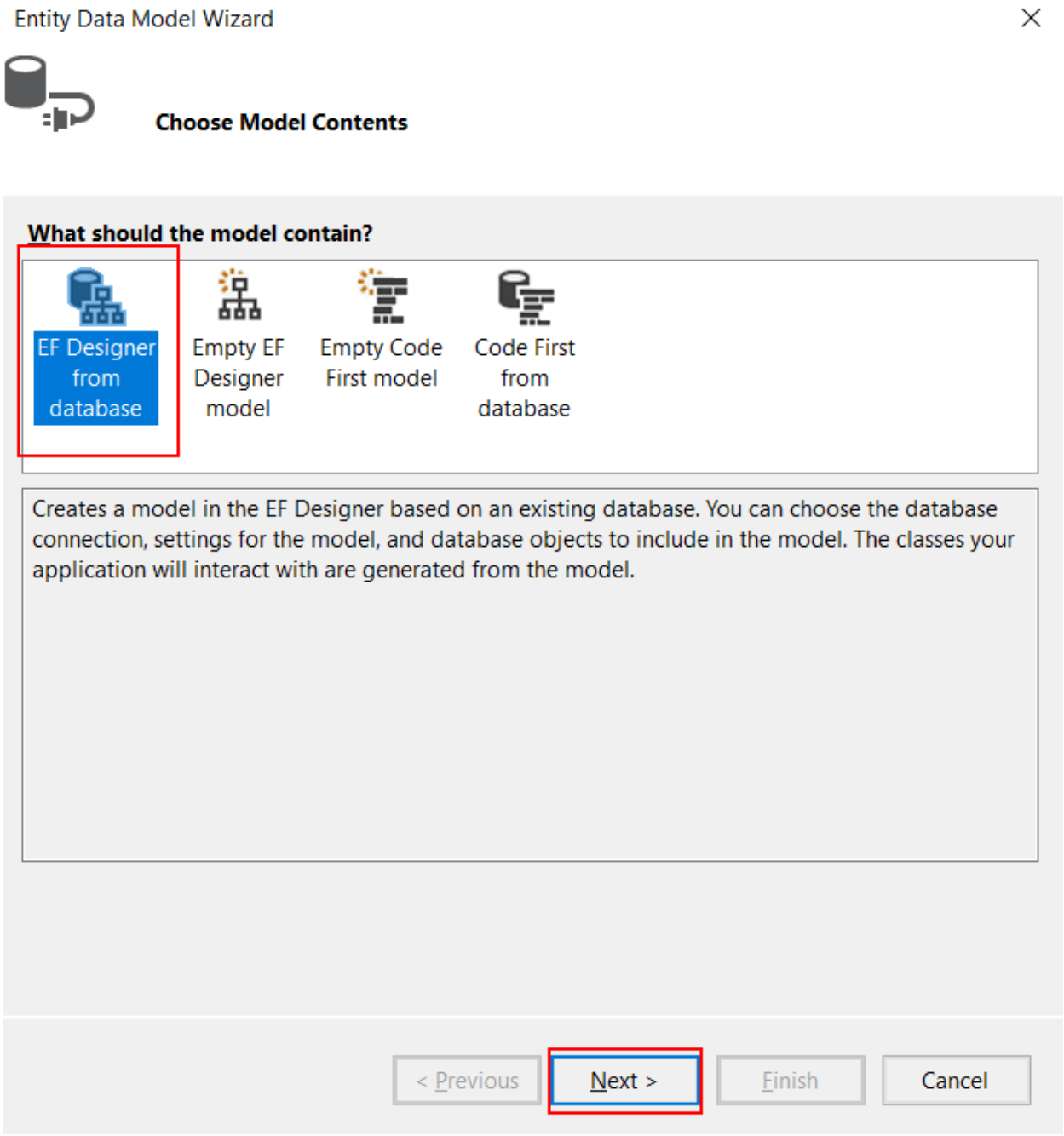
There is a change that in visual studio 2019 you cannot see ADO.Net Entity Data Model option. For adding ADO.Net Entity Data Model in visual studio open visual studio install , select modify option by clicking on more button. Now you see a new window click on Individual Component tab, search for Entity Framework tool then select it and click on modify button.



Now it starts downloading. If your visual studio is open then save all files because it will close visual studio before installing updates.

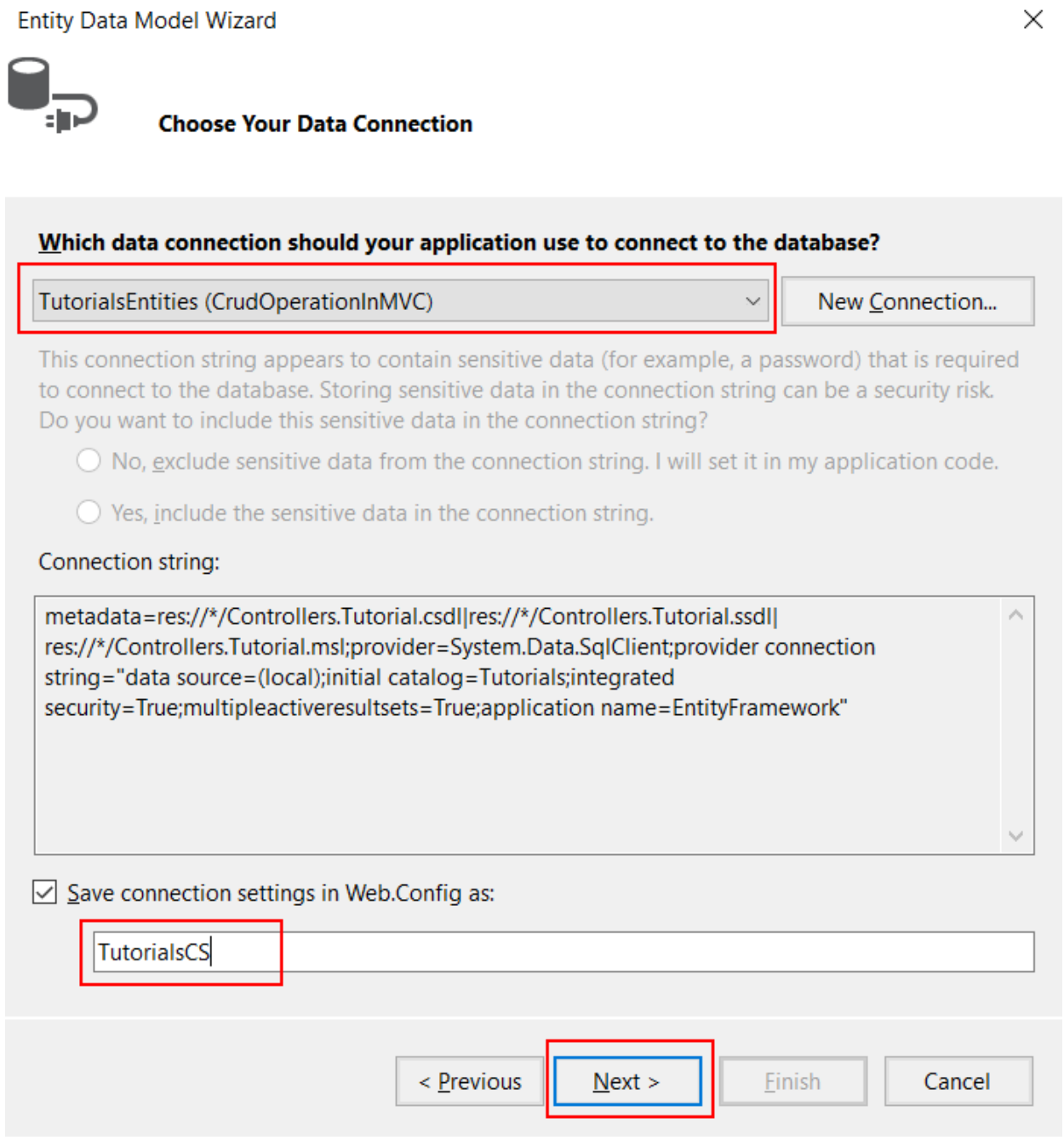
Step 11

After adding ADO.Net Entity Data Model you will see a popup window. Select EF Designer From database and click on next button.



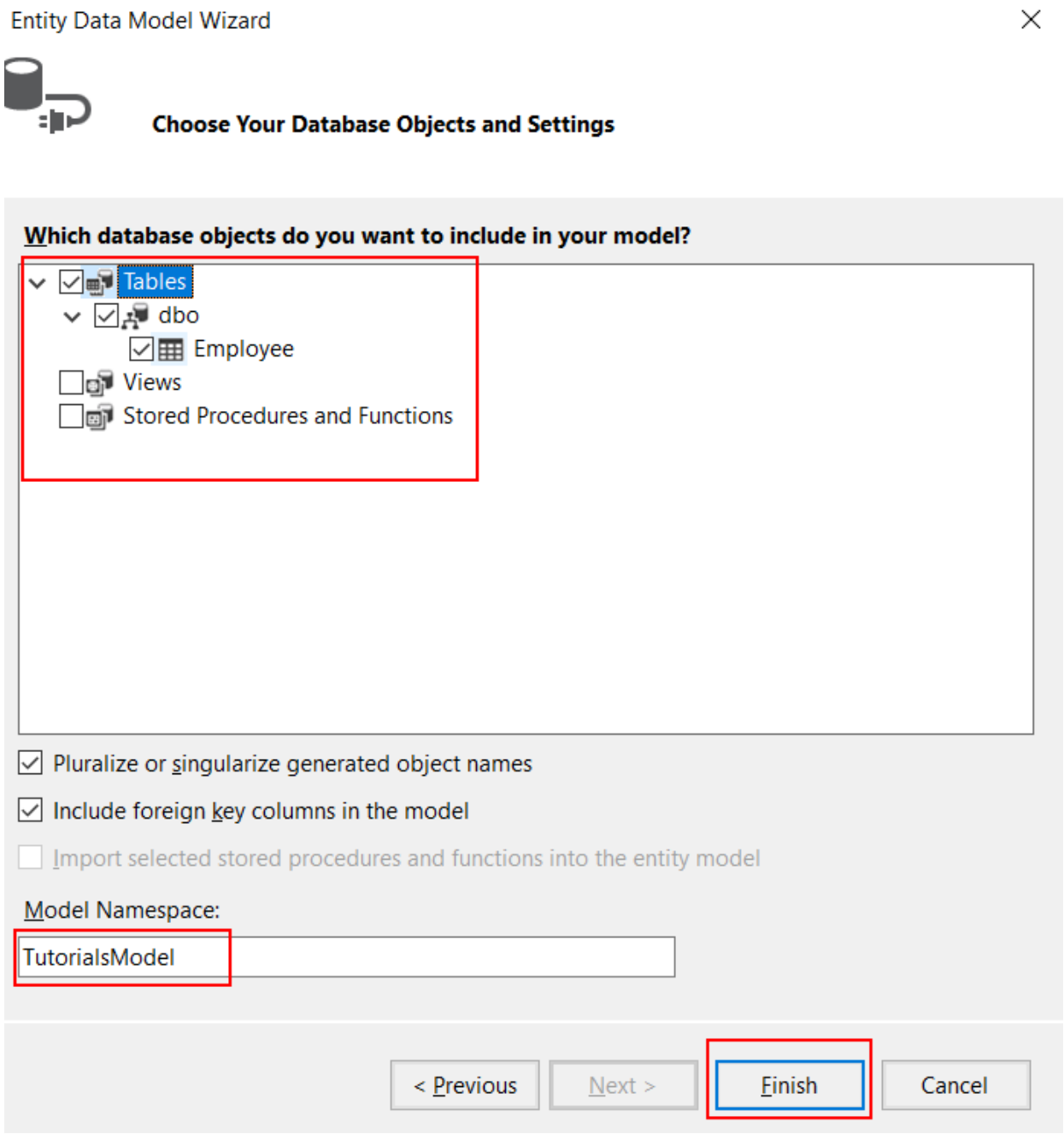
Step 12

Now select your database from drop down and give connection string name, and click on Next button.

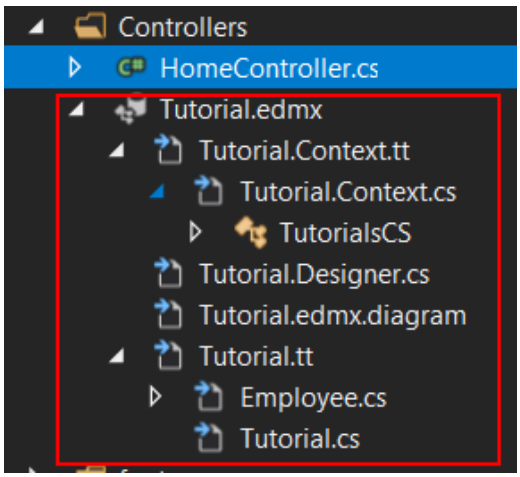


Step 13

Select tables which you want to add in project, here I have only one table so I select that, gave Model Namespace name and Click on finish button.



Now Entity Framework creates an edmx file in your project as you can see in the below image.



Here is Employee Model.

```
01. public partial class Employee
02. {
03.
```

```
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }
        public Nullable<decimal> EmployeeSalary { get; set; }
        public string EmployeeCity { get; set; }
}
```

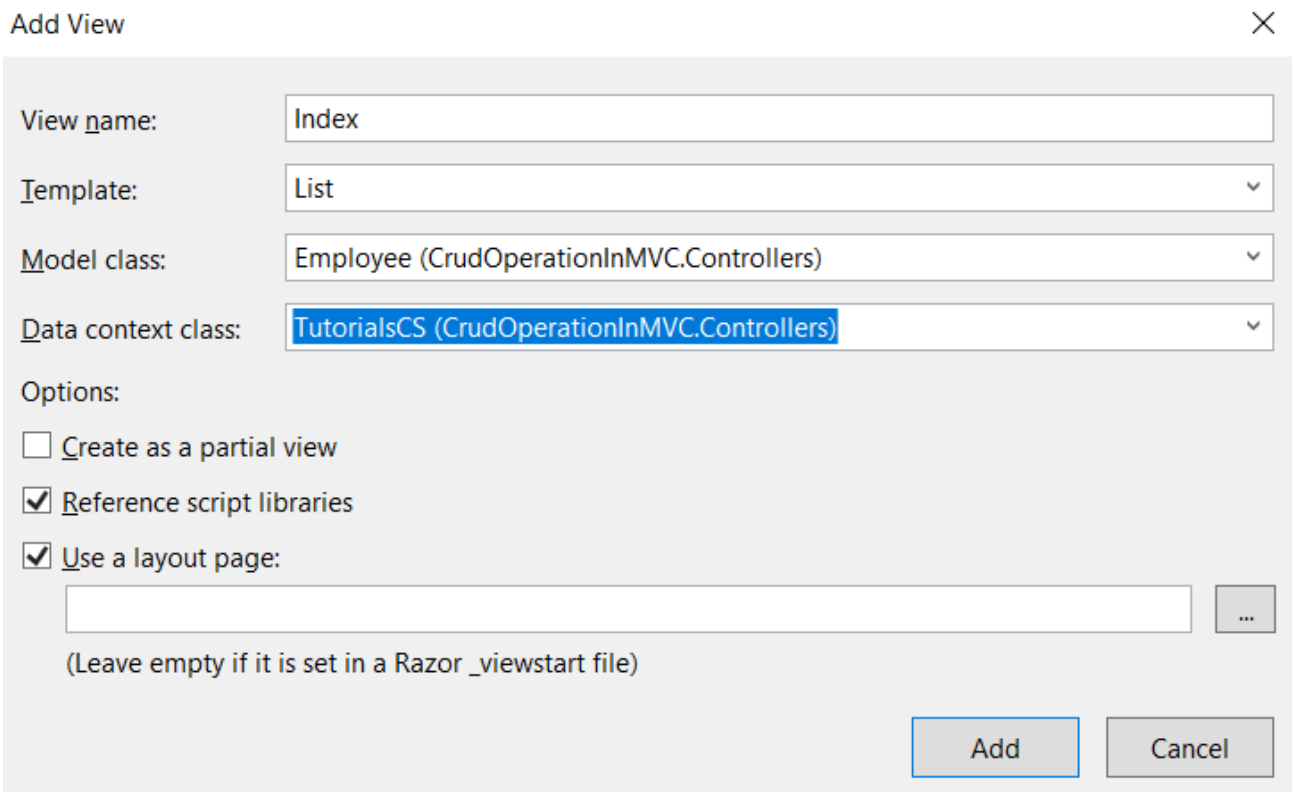
Step 14

Now delete all method from your controller and add index method as show in the below code.

```
public ActionResult Index()
{
    var listofData = _context.Employees.ToList();
    return View(listofData);
}
```

Step 15

Now add view by right clicking in method then click on Add View and you will see a new popup window with the following options. And then click on Add button.



- View Name: Name of your Cs Html file
- Template: There are many templates which give pre defined design like list, create, edit, delete, details select list for index method
- Model Class: Select your model class which we generated in the previous step.
- Data Context Class: Select your data context class, here I have TutorialCS whose name we gave in previous step.
- Use Layout Page: if you check this item compiler add layout file from _Shared folder in this view. You can select different layout file if you have more than one .

Now Compiler generates view for list modifier design if you want. Design file code is below.

```
@model IEnumerable<CrudOperationInMVC.Controllers.Employee>

@{
    ViewBag.Title = "Index";
}

<div>Index</div>
```

```
<p class="text-right">
    @Html.ActionLink("Create New", "Create")
</p>

@if (ViewBag.Message != null)
{

<div class="alert alert-success" role="alert">
    @ViewBag.Message
</div>
}


<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.EmployeeName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.EmployeeSalary)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.EmployeeCity)
        </th>
        <th></th>
    </tr>


    @if (Model != null && Model.Count() > 0)
    {
        foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.EmployeeName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.EmployeeSalary)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.EmployeeCity)
                </td>
                <td>
                    @Html.ActionLink("Edit", "Edit", new { id = item.EmployeeId }) |
                    @Html.ActionLink("Details", "Detail", new { id = item.EmployeeId
                    @Html.ActionLink("Delete", "Delete", new { id = item.EmployeeId
                </td>
            </tr>
        }
    }
    else
    {
        <td colspan="4" class="text-center">
<b>No Data Available . Please Add Data By CLick On Create Button</b></td>
    }
</table>
```

Step 16

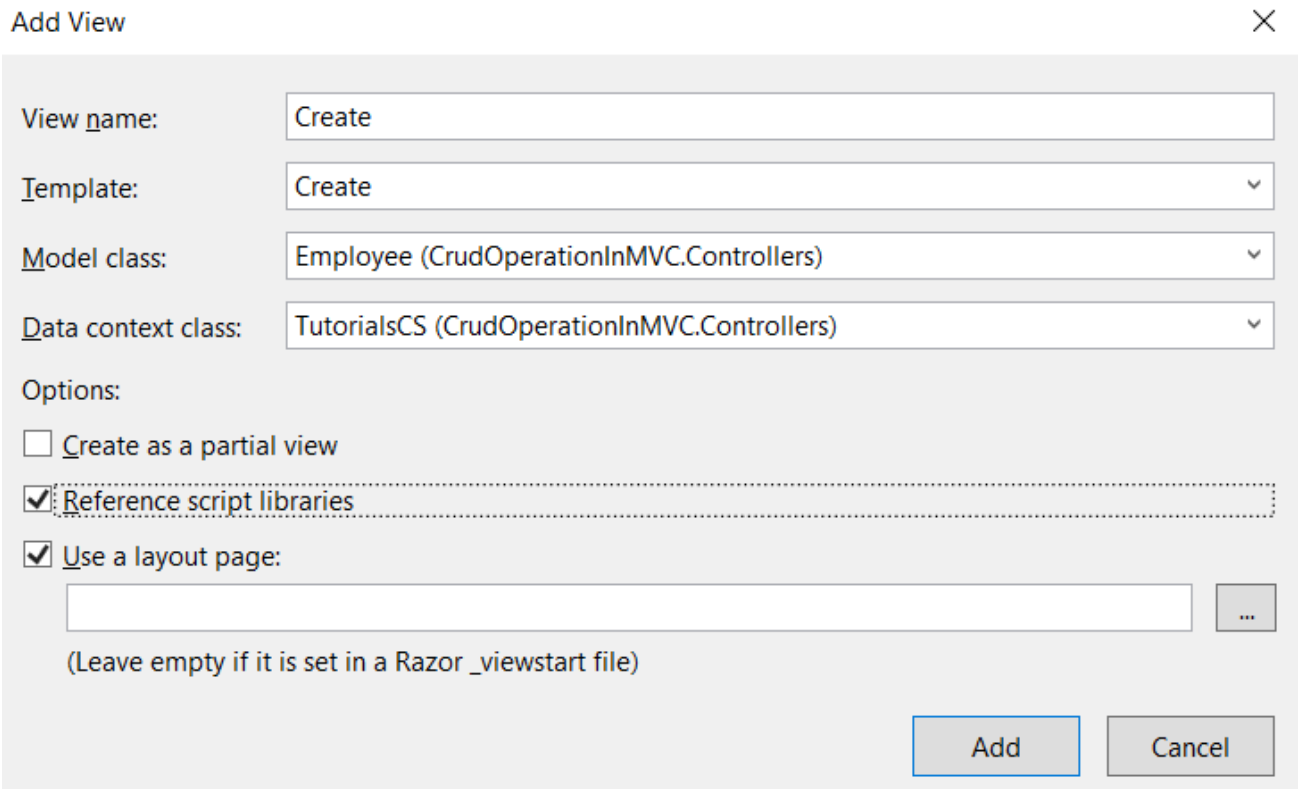
Now we create a view for adding a new record. Here we need to create two Action methods, one is get method and the second is post method. Get method is called when we click on create button and return view. Post method is used for saving data in the table.

```
[HttpGet]
 public ActionResult Create()
 {
     return View();
 }

 [HttpPost]
 public ActionResult Create(Employee model)
 {
     _context.Employees.Add(model);
     _context.SaveChanges();
     ViewBag.Message = "Data Insert Successfully";
     return View();
 }
```

Code Explanation

- In post method we pass Employee class as a parameter as model name which gets all data from user.
- Then we add that model in TutoriaCS using _context object which we create top of controller.
- SaveChanges() method use for save all changes which we make in entity like add, delete, update etc.
- You can see output of create below.

For Adding view for Create right click in method click on add view and select template Create.

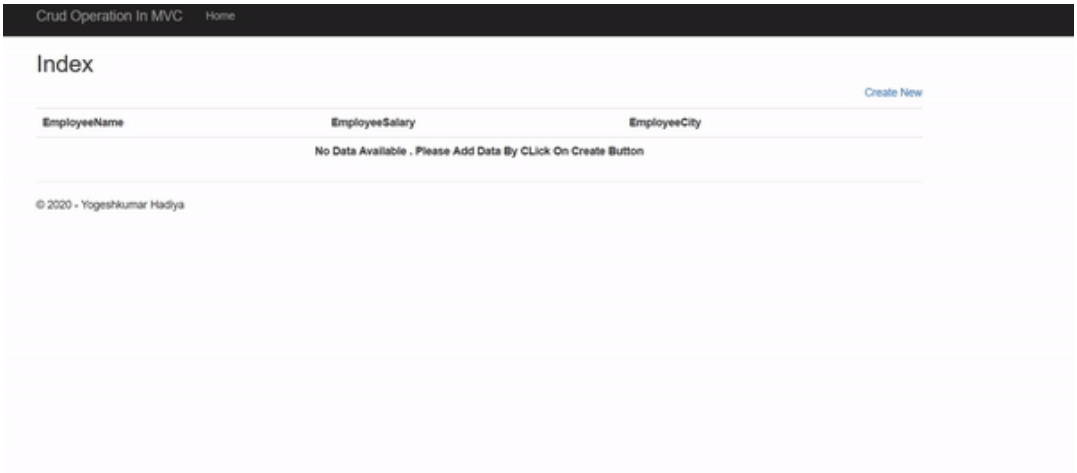


Design for create view is below,

```
@model CrudOperationInMVC.Controllers.Employee

@{
    ViewBag.Title = "Create";
}

<div>Create</div>

@if (ViewBag.Message != null)
{
<div class="alert alert-success" role="alert">
 @ViewBag.Message
</div>
```

```
}


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <div>Employee</div>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.EmployeeName, htmlAttributes: new { @class
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.EmployeeName, new { htmlAttributes =
control" } })
                @Html.ValidationMessageFor(model => model.EmployeeName, "", new { @c
danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.EmployeeSalary, htmlAttributes: new { @cla
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.EmployeeSalary, new { htmlAttributes
control" } })
                @Html.ValidationMessageFor(model => model.EmployeeSalary, "", new {
danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.EmployeeCity, htmlAttributes: new { @class
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.EmployeeCity, new { htmlAttributes =
control" } })
                @Html.ValidationMessageFor(model => model.EmployeeCity, "", new { @c
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>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Output



Step 17

Now we will implement Edit functionality. For edit we also create two methods, get and post type. Get type method gets data from database by recording it and passing it in view. Post method saves updated data in database. Code for both methods is below.

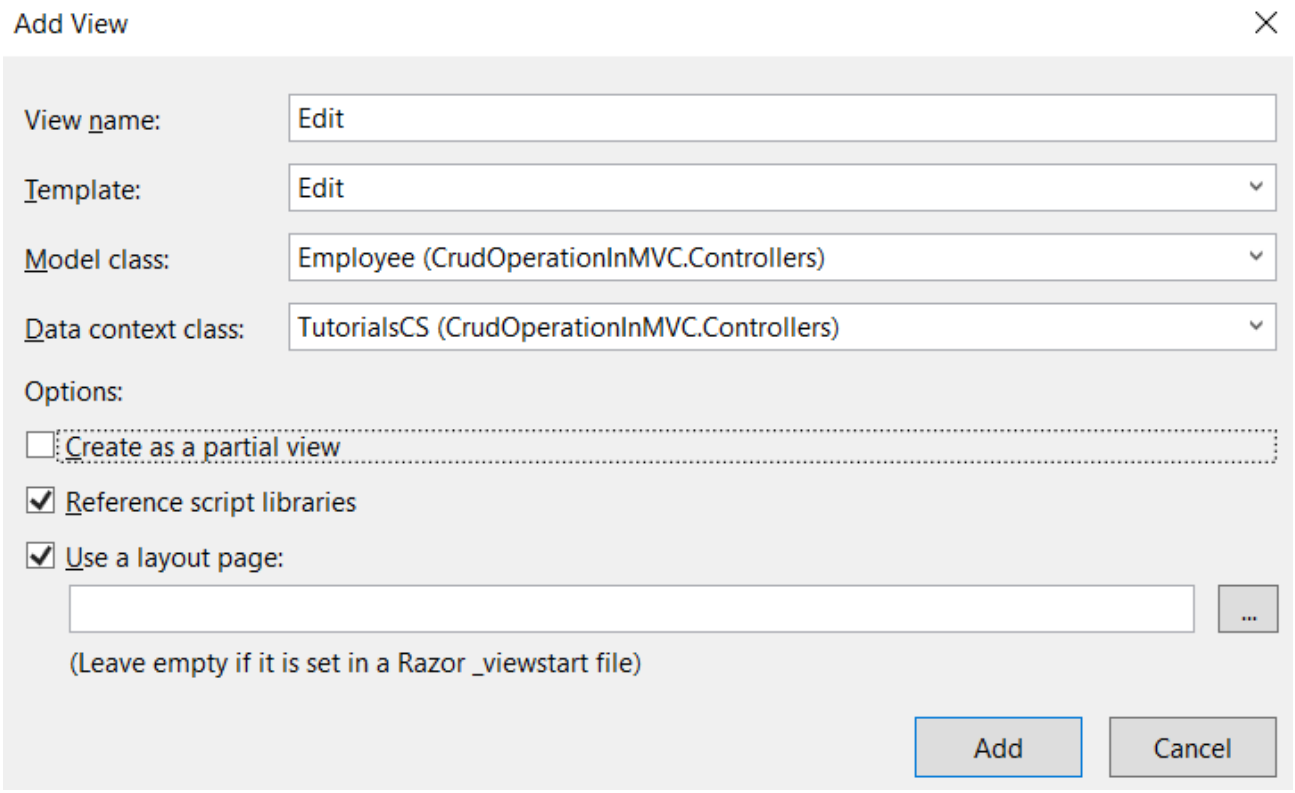
```
[HttpGet]
public ActionResult Edit(int id)
{
    var data = _context.Employees.Where(x => x.EmployeeId == id).FirstOrDefault();
    return View(data);
}
[HttpPost]
public ActionResult Edit(Employee Model)
{
    var data = _context.Employees.Where(x => x.EmployeeId == Model.EmployeeId).First(
    if (data != null)
    {
        data.EmployeeCity = Model.EmployeeCity;
        data.EmployeeName = Model.EmployeeName;
        data.EmployeeSalary = Model.EmployeeSalary;
        _context.SaveChanges();
    }

    return RedirectToAction("index");
}
```

Code Explanation

- In get type method we get an id as a parameter using that we get first record from database and initialize in data variable then return that variable in view
- In post type method we get Employee type model as a parameter; we get record from database with EmployeeId which comes with parameter.
- If record is found in database with that id then we initialize upcoming data in database data.
- And save changes using SaveChanges Method. And return to index method using RedirectToAction method.

Step 18

Now we need to add view for editing. For this right click in Edit method click on Add View and select Edit Template' all other options are the same as index.

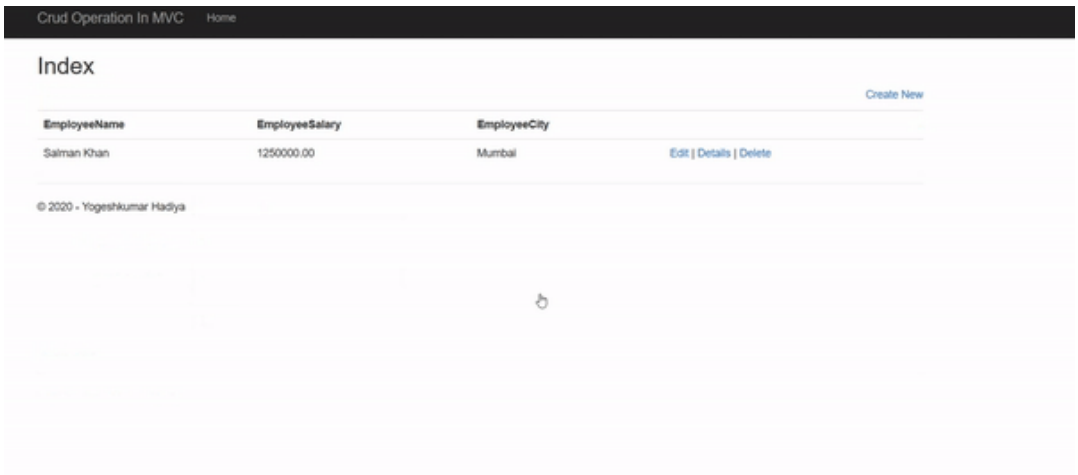


Code of View for Edit is below.

```
@model CrudOperationInMVC.Controllers.Employee

@{
    ViewBag.Title = "Edit";
}

<div>Edit</div>



@if (ViewBag.Message != null)
{

    <div class="alert alert-success" role="alert">
        @ViewBag.Message
    </div>
}


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <div>Employee</div>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.EmployeeId)

        <div class="form-group">
            @Html.LabelFor(model => model.EmployeeName, htmlAttributes: new { @class
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.EmployeeName, new { htmlAttributes =
control" } })
                @Html.ValidationMessageFor(model => model.EmployeeName, "", new { @c
danger" })
```

```
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.EmployeeSalary, htmlAttributes: new { @cla
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.EmployeeSalary, new { htmlAttributes
control" } })
                @Html.ValidationMessageFor(model => model.EmployeeSalary, "", new {
danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.EmployeeCity, htmlAttributes: new { @class
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.EmployeeCity, new { htmlAttributes =
control" } })
                @Html.ValidationMessageFor(model => model.EmployeeCity, "", new { @c
danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

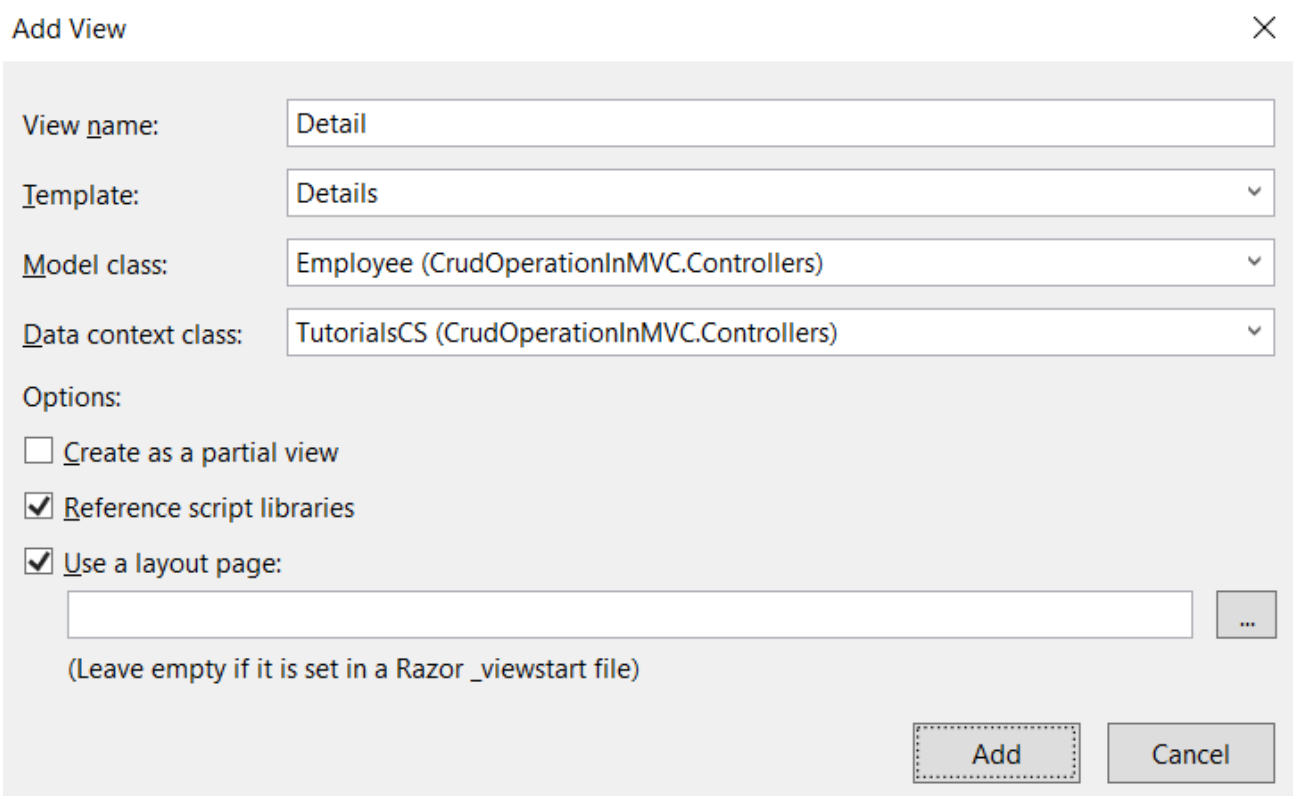
Output



Step 19

Now we create view for viewing employee data. So create Action method detail with parameter id . Using this we get data from database and pass it in view. Add view by right clicking in method and selecting template Details.

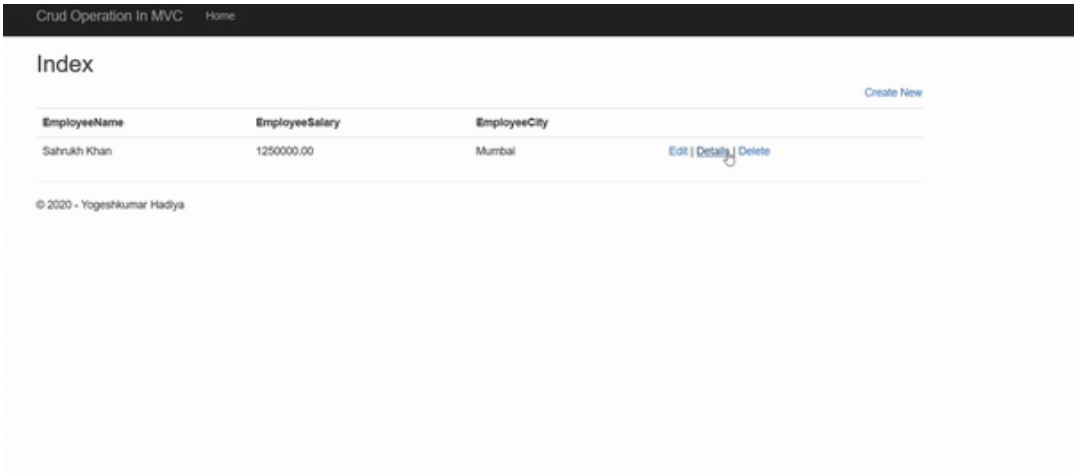
```
01. public ActionResult Detail(int id)
02. {
03.     var data = _context.Employees.Where(x => x.EmployeeId == id).FirstOrDefault();
04.     return View(data);
05. }
```




```
01. @model CrudOperationInMVC.Controllers.Employee
02.
03. @{
04.     ViewBag.Title = "Detail";
05. }
06.
07. <div>Detail</div>
08.
09. <div>
10.     <div>Employee</div>
11.     <hr />
12.     <dl class="dl-horizontal">
13.         <dt>
14.             @Html.DisplayNameFor(model => model.EmployeeName)
15.         </dt>
16.
17.         <dd>
18.             @Html.DisplayFor(model => model.EmployeeName)
19.         </dd>
20.
21.         <dt>
22.             @Html.DisplayNameFor(model => model.EmployeeSalary)
23.         </dt>
24.
25.         <dd>
26.             @Html.DisplayFor(model => model.EmployeeSalary)
27.         </dd>
28.
29.         <dt>
30.             @Html.DisplayNameFor(model => model.EmployeeCity)
31.         </dt>
32.
33.         <dd>
```

```
            @Html.DisplayFor(model => model.EmployeeCity)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new { id = Model.EmployeeId }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

Output



Step 20

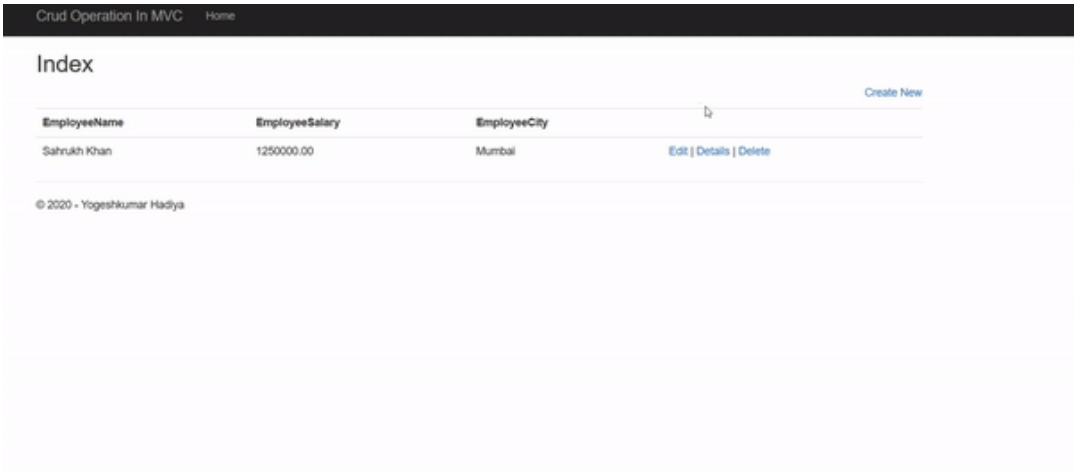
Now we create Action Method for delete which takes id as n parameter.

```
public ActionResult Delete(int id)
{
    var data = _context.Employees.Where(x => x.EmployeeId == id).FirstOrDefault();
    _context.Employees.Remove(data);
    _context.SaveChanges();
    ViewBag.Messsage = "Record Delete Successfully";
    return RedirectToAction("index");
}
```

Code Explanation

- Here first we get data by id from database.
- Remove that entity which we get from database.
- Save changes using SaveChanges() method.
- And redirect to Index method.

Output



Here are the whole controller codes.

```
01. using System;
02. using System.Collections.Generic;
03. using System.Linq;
04. using System.Web;
05. using System.Web.Mvc;
06. 
07. namespace CrudOperationInMVC.Controllers
08. {
09. 
10. 
11. 
12.     public class HomeController : Controller
13.     {
14.         TutorialsCS _context = new TutorialsCS();
15. 
16.         public ActionResult Index()
17.         {
18.             var listofData = _context.Employees.ToList();
19.             return View(listofData);
20.         }
21. 
22.         [HttpGet]
23.         public ActionResult Create()
24.         {
25.             return View();
26.         }
27. 
28.         [HttpPost]
29.         public ActionResult Create(Employee model)
30.         {
31.             _context.Employees.Add(model);
32.             _context.SaveChanges();
33.             ViewBag.Message = "Data Insert Successfully";
34.             return View();
35.         }
36.         [HttpGet]
37.         public ActionResult Edit(int id)
38.         {
39.             var data = _context.Employees.Where(x => x.EmployeeId == id).FirstOrDefa
40.             return View(data);
41.         }
42.         [HttpPost]
43.         public ActionResult Edit(Employee Model)
44.         {
45.             var data = _context.Employees.Where(x => x.EmployeeId == Model.EmployeeI
46.             if (data != null)
47.             {
48.                 data.EmployeeCity = Model.EmployeeCity;
49.                 data.EmployeeName = Model.EmployeeName;
50.                 data.EmployeeSalary = Model.EmployeeSalary;
51.                 _context.SaveChanges();
52.             }
53. 
54.             return RedirectToAction("index");
55.         }
56.         public ActionResult Delete(int id)
57.         {
58.             var data = _context.Employees.Where(x => x.EmployeeId == id).FirstOrDefa
59.             _context.Employees.Remove(data);
60.             _context.SaveChanges();
61.             ViewBag.Messsage = "Record Delete Successfully";
62.             return RedirectToAction("index");
63.         }
64.         public ActionResult Detail(int id)
65.         {
```

```
66.             var data = _context.Employees.Where(x => x.EmployeeId == id).FirstOrDefa
67.             return View(data);
68.         }
69.
70.     }
71. }
```

Conclusion

I hope you liked this article and got some information. If you found this article help full share with your friends. Thank You.