Module - 9

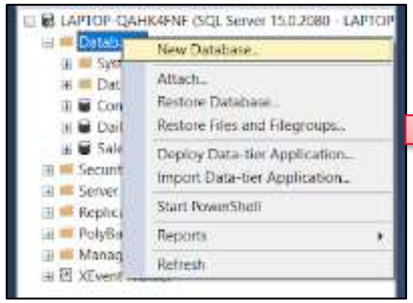
CRUD operations using EF

What is Entity Framework?

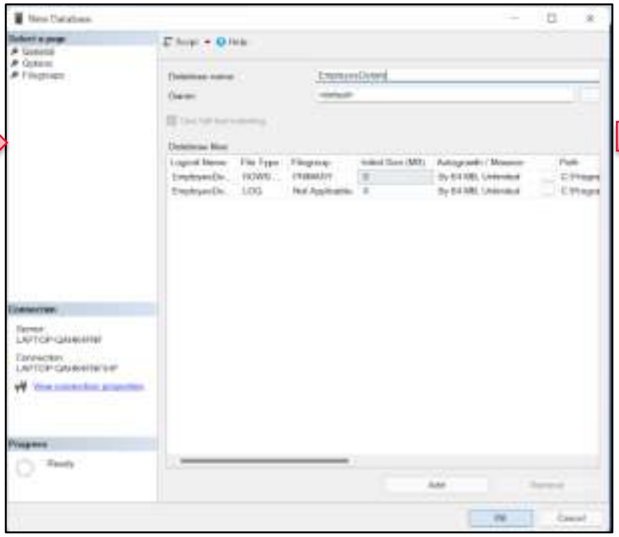
Entity Framework was first released in 2008, Microsoft's primary means of interacting between .NET applications and relational databases. Entity Framework is an Object Relational Mapper (ORM) which is a type of tool that simplifies mapping between objects in your software to the tables and columns of a relational database.

- Entity Framework (EF) is an open source ORM framework for ADO.NET which is a part of .NET Framework.
- An ORM takes care of creating database connections and executing commands, as well as taking query results and automatically materializing those results as your application objects.
- An ORM also helps to keep track of changes to those objects, and when instructed, it will also persist those changes back to the database for you.
- The Entity Framework provides three approaches to create an entity model and each one has their own pros and cons.
    1. Database First approach
    2. Code First approach
    3. Model First approach
- For our CRUD operation application we will use database first approach
- Database first approach –
    - It creates model codes (classes, properties, DbContext etc.) from the database in the project and those classes become the link between the database and controller.
    - The Database First Approach creates the entity framework from an existing database.

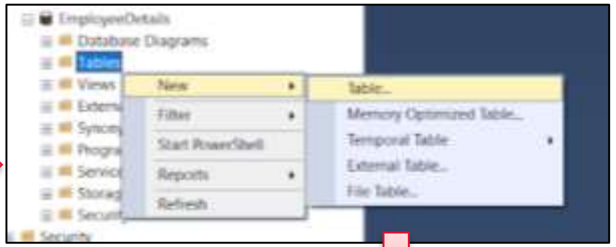
- To get started by understanding entity framework with database first approach, we need to create a database.
- Opening our Microsoft SQL Server Management Studio 18 and creating a table –
- Making EmpId as primary key and Identity specification as ‘Yes’.

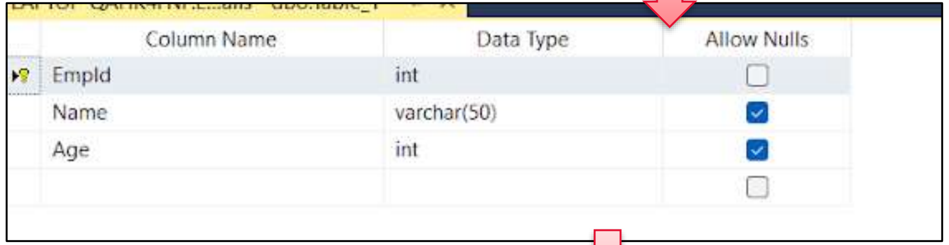

LAPTOP-QAHK4FNF (SQL Server 15.0.2080 - LAPTOP
Databa
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh

Create Database

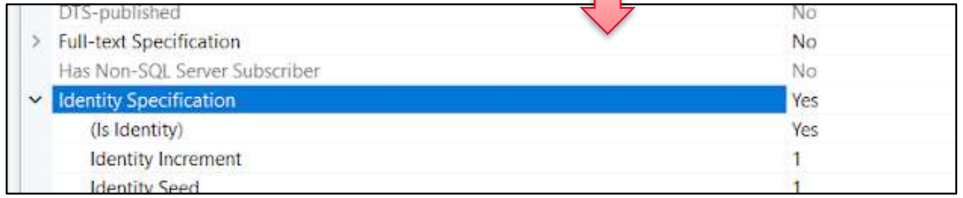


Add table to database





Column Name	Data Type	Allow Nulls
EmpId	int	☐
Name	varchar(50)	☑
Age	int	☑
		☐



DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Save Table

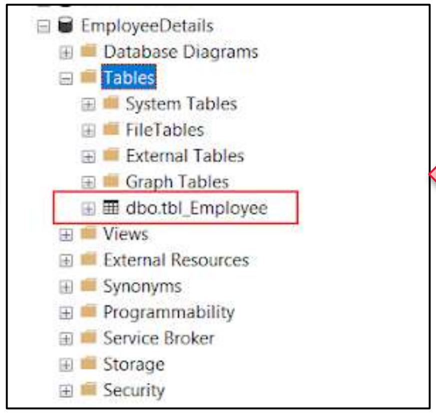
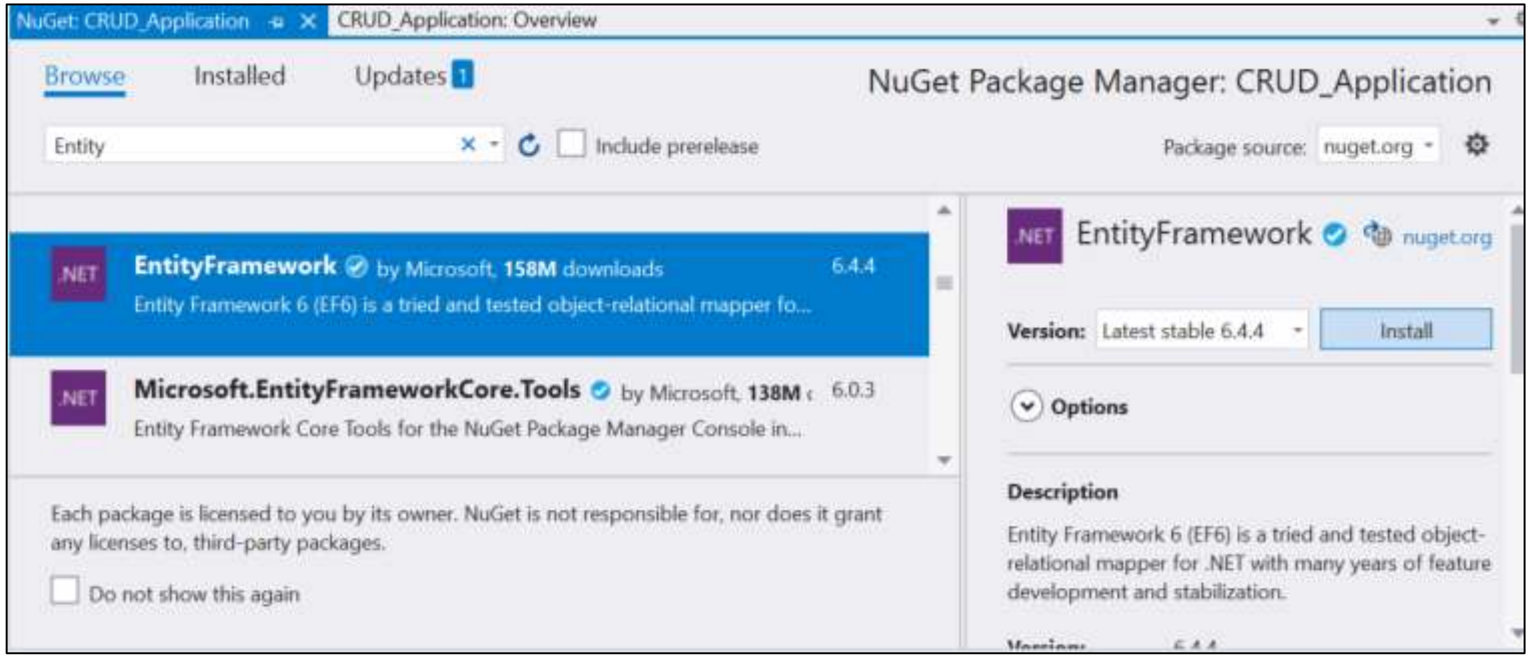


Table created

- Now to use database in our application –
  1. First right click on project > Manage NuGet Packages > EntityFramework > Install
  2. After installation right click on the model folder => Add => ADO.NET Entity Data model (*If you're not getting this command in then click on 'New item' and then search and select ADO.NET Entity Data model*)


NuGet: CRUD_Application
CRUD_Application: Overview
Browse
Installed
Updates 1
NuGet Package Manager: CRUD_Application
Entity
Include prerelease
Package source: nuget.org
EntityFramework by Microsoft, 158M downloads 6.4.4
Microsoft.EntityFrameworkCore.Tools by Microsoft, 138M 6.0.3
.NET
EntityFramework
nuget.org
Version: Latest stable 6.4.4
Install
Options
Description
Version: 6.4.4

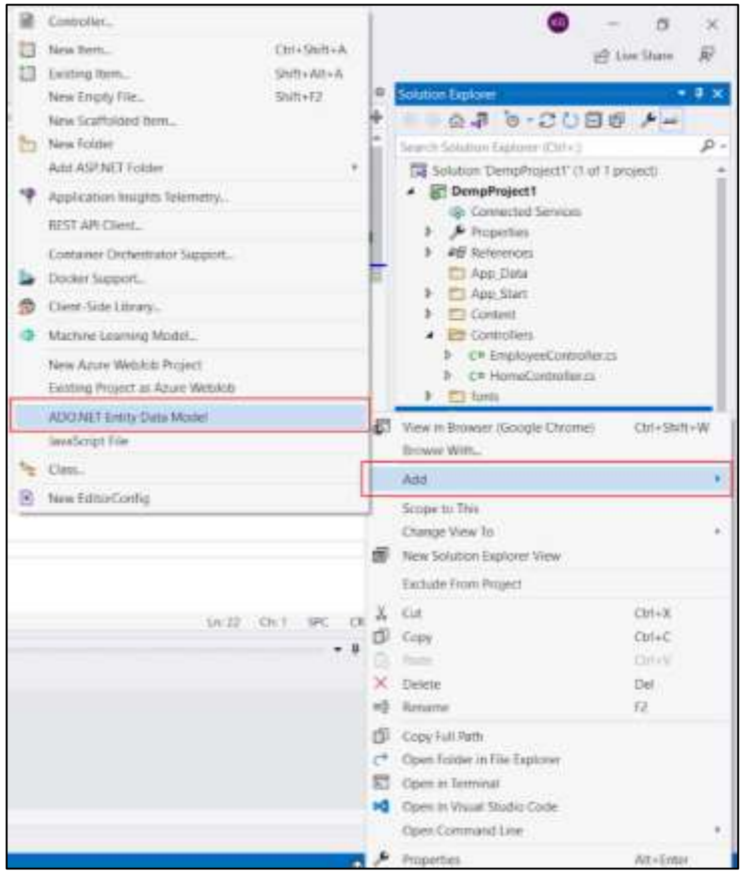
Entity Framework 6 (EF6) is a tried and tested object-relational mapper fo...

Entity Framework Core Tools for the NuGet Package Manager Console in...

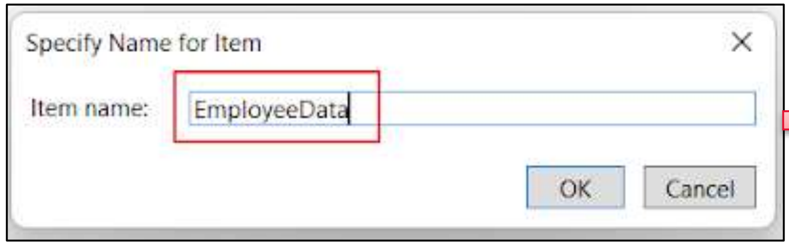
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

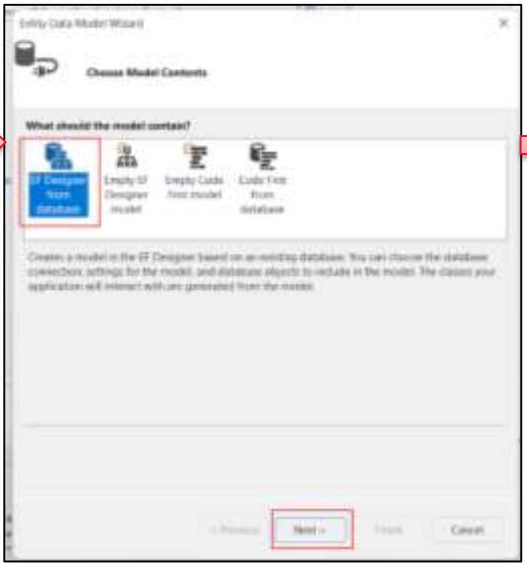
Entity Framework 6 (EF6) is a tried and tested object-relational mapper for .NET with many years of feature development and stabilization.



- Now follow following steps –


Specify Name for Item

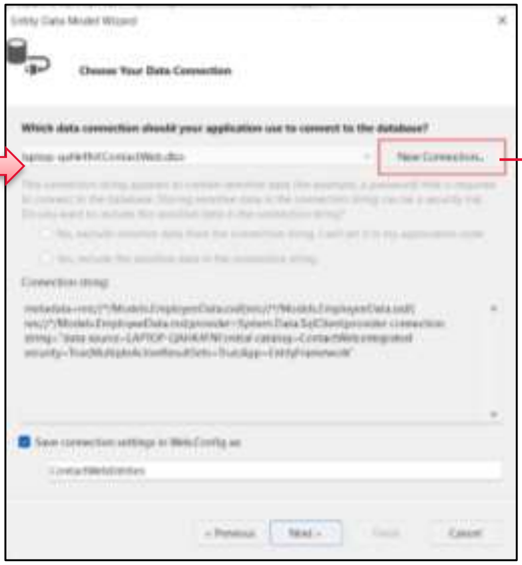
Item name: EmployeeData


Entity Data Model Wizard

Choose Model Contents

What should the model contain?

Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.


Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

laptop-qahk4fnf.ContactWeb.dbo

[illegible]

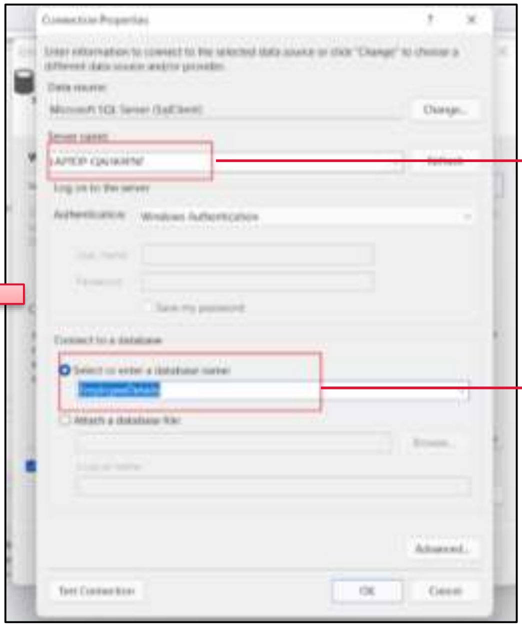
Connection string:

metadata=res://*/Models.EmployeeData.csdl|res://*/Models.EmployeeData.ssdl|res://*/Models.EmployeeData.msl;provider=System.Data.SqlClient;provider connection string="data source=LAPTOP-QAHK4FNF;initial catalog=ContactWeb;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"

☑ Save connection settings in Web.Config as:

ContactWebEntities

Creating new connection


Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Microsoft SQL Server (SqlClient)

Server name:

LAPTOP-QAHK4FNF

Log on to the server

Authentication: Windows Authentication

User name:

Password:

☐ Save my password

Connect to a database

● Select or enter a database name:

EmployeeDetails

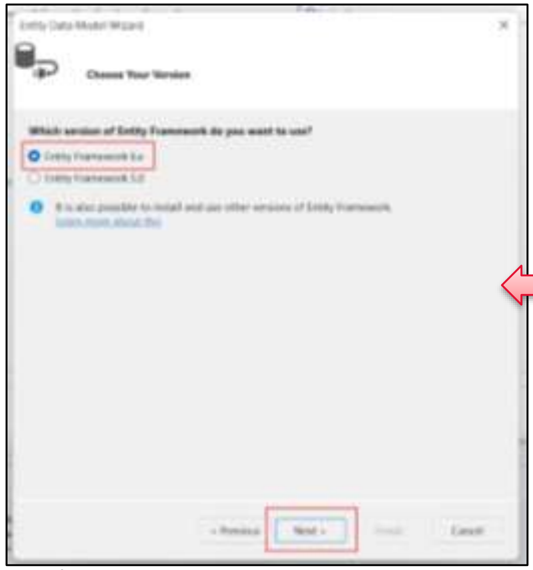
○ Attach a database file:

Logical name:

Enter server name

Select database

Select latest framework and click next =>


Entity Data Model Wizard

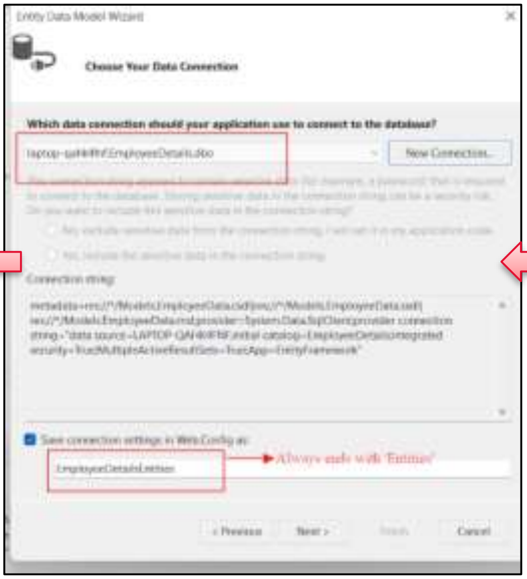
Choose Your Version

Which version of Entity Framework do you want to use?

● Entity Framework 6.x

○ Entity Framework 5.0

It is also possible to install and use other versions of Entity Framework. Learn more about this


Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

laptop-qahk4fnf.EmployeeDetails.dbo

[illegible]

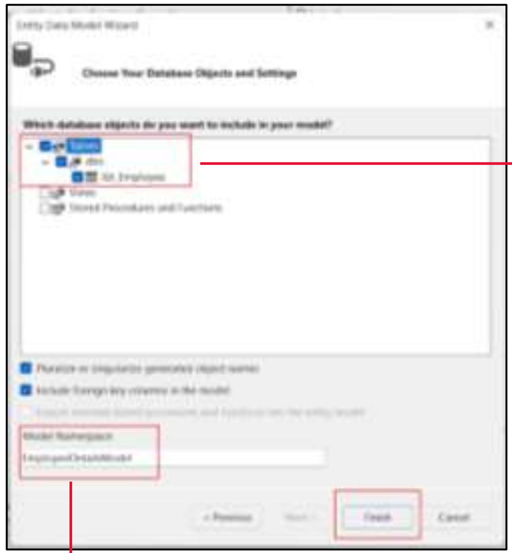
Connection string:

metadata=res://*/Models.EmployeeData.csdl|res://*/Models.EmployeeData.ssdl|res://*/Models.EmployeeData.msl;provider=System.Data.SqlClient;provider connection string="data source=LAPTOP-QAHK4FNF;initial catalog=EmployeeDetails;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"

☑ Save connection settings in Web.Config as:

EmployeeDetailsEntities

Always ends with 'Entities'


Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- Tables
  - dbo
    - tbl_Employee
- Views
- Stored Procedures and Functions

Pluralize or singularize generated object names

Include foreign key columns in the model

Import selected stored procedures and functions into the entity model

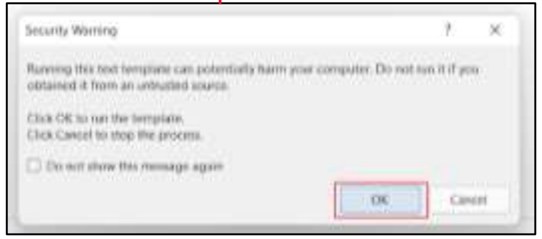
Model Namespace

EmployeeDetailsModel

Select table or view or stored procedure

This model namespace is used for creating connection strings

This warning might occur or might not occur in some cases, just click ok and wait till it stops occurring

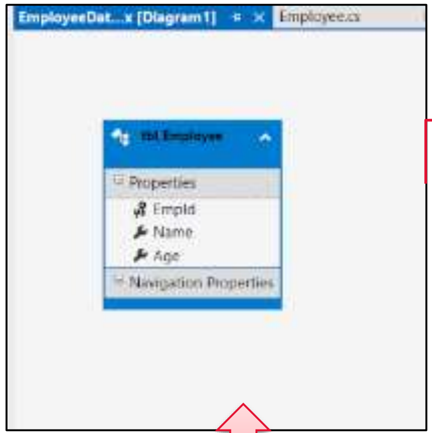

Security Warning

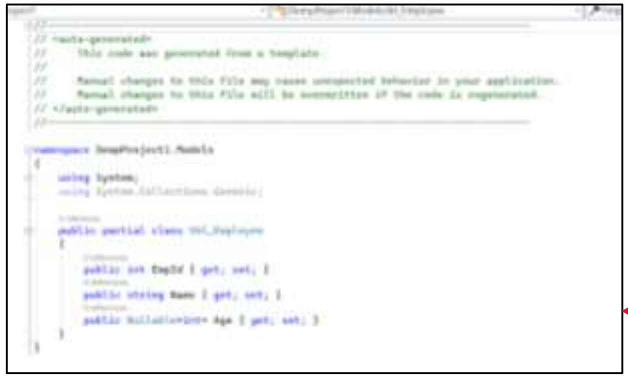
Running this text template can potentially harm your computer. Do not run it if you obtained it from an untrusted source.

Click OK to run the template.
Click Cancel to stop the process.

Do not show this message again

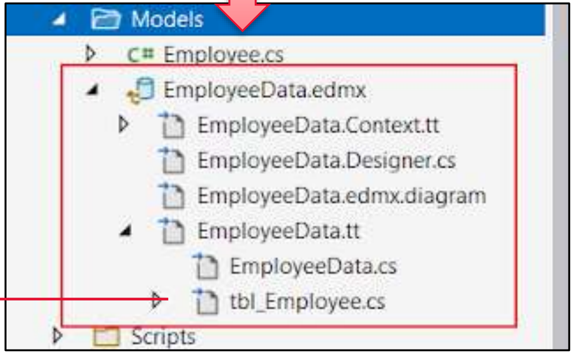
Select table or view or stored procedure





```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace DemoProject1.Models
{
    using System;
    using System.Collections.Generic;

    public partial class tbl_Employee
    {
        public int EmpId { get; set; }
        public string Name { get; set; }
        public Nullable<int> Age { get; set; }
    }
}
```

Table model

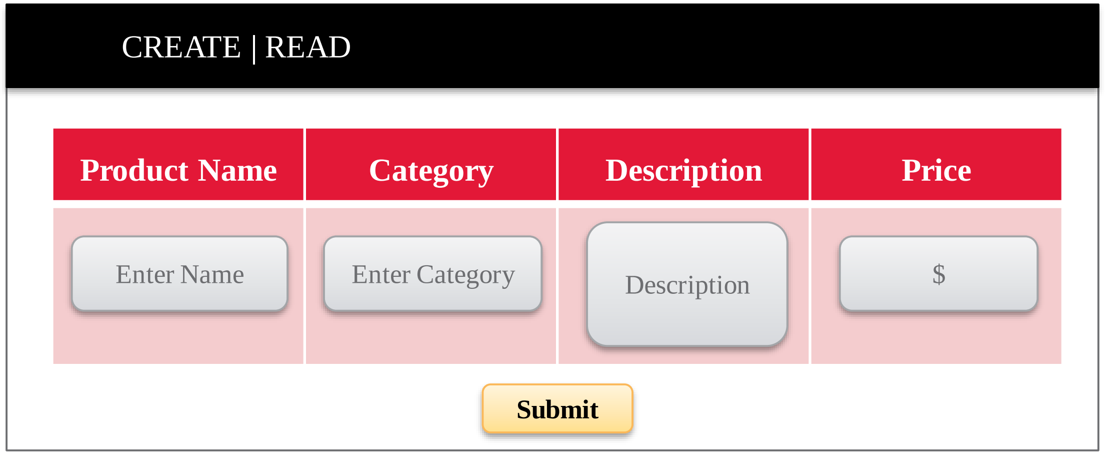


This creates Database model

What are CRUD operations?

- As discussed in module-6 CRUD operations means Create, Read, Update and Delete. CRUD Operations are the basic thing when performing database operations. We can insert a record then read, edit or delete it from the database.
- Lets create a CRUD application for products data which consists of following for input–
  - Product Name
  - Category
  - Description
  - Price
- And following data will be presented as the output–
  - Product Id
  - Product Name
  - Category
  - Description
  - Price
  - Edit/Delete option

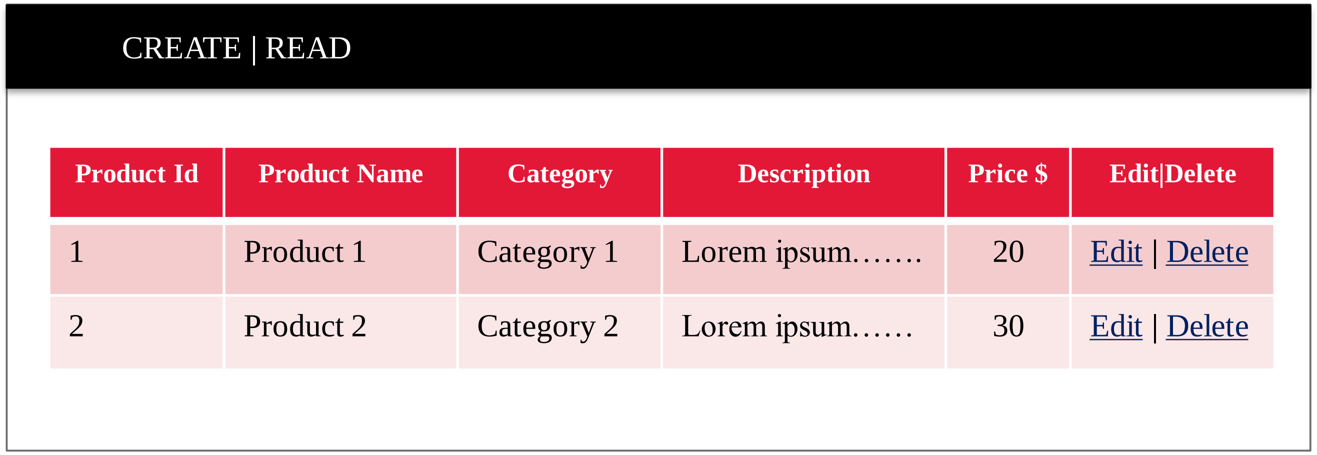
Create



CREATE | READ

Product Name	Category	Description	Price
Enter Name	Enter Category	Description	$

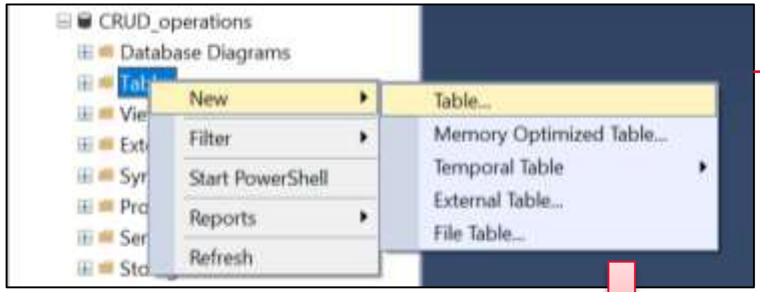
Read



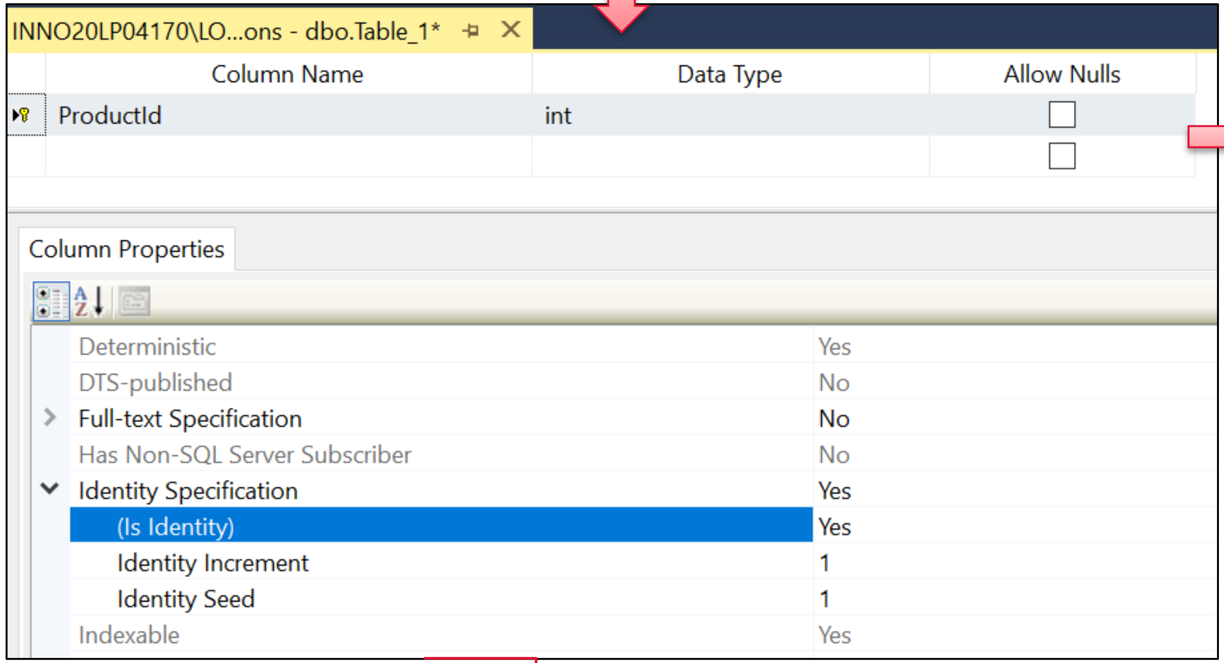
CREATE | READ

Product Id	Product Name	Category	Description	Price $	Edit|Delete
1	Product 1	Category 1	Lorem ipsum…….	20	Edit | Delete
2	Product 2	Category 2	Lorem ipsum……	30	Edit | Delete

- First we need to create a database in SQL server. Follow the below steps –



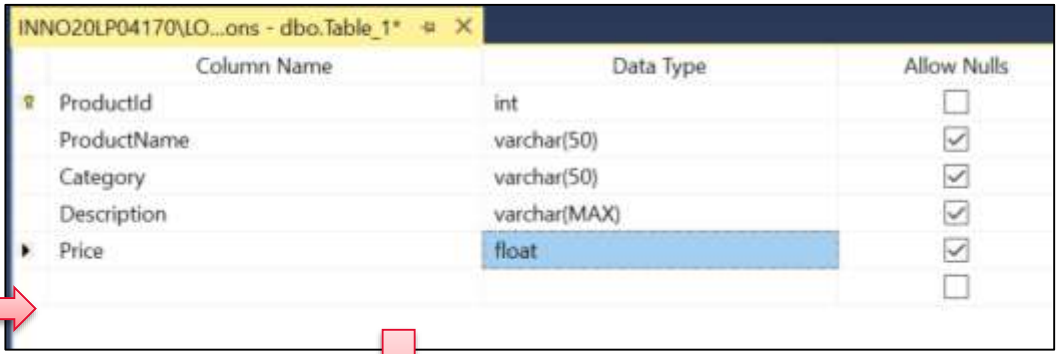
Creating new table



Column Name	Data Type	Allow Nulls
ProductId	int	☐
		☐

Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes

Make product key primary key and Identity specification as ‘Yes’



Column Name	Data Type	Allow Nulls
ProductId	int	☐
ProductName	varchar(50)	☑
Category	varchar(50)	☑
Description	varchar(MAX)	☑
Price	float	☑
		☐

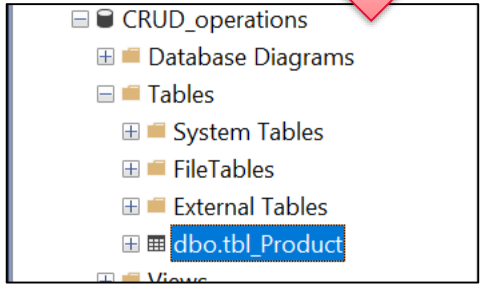


Table Created

I. Lets create a new project for CRUD application –

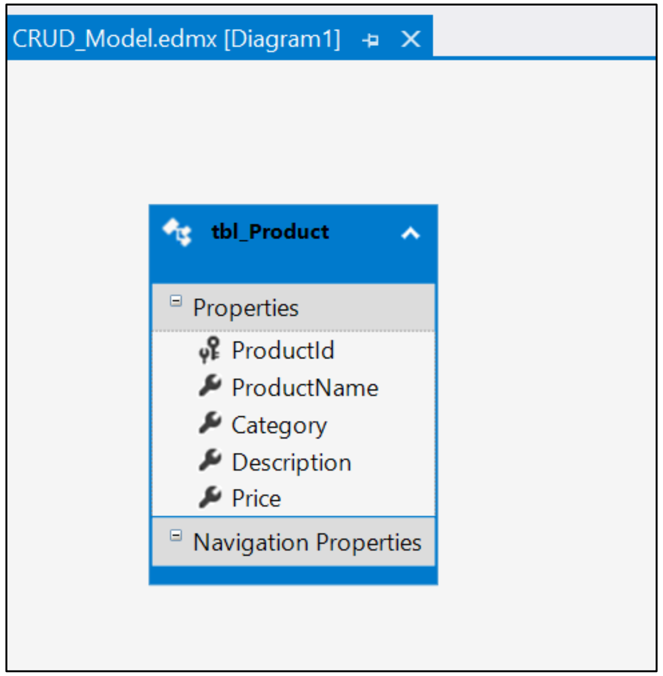
1. Open Microsoft Visual Studio.
2. Click on File > New > Project and select ASP.NET Web Application Template.
3. Enter the project name and click Ok.
4. Click on Empty, check the check-box MVC, and click on Ok. An empty MVC web application will open.

(Reference Module-2)

II. Lets create a new project for CRUD application –

1. Right click on project > Manage NuGet packages > EntityFramework > Install
2. After installation right click on models > Add > New item > ADO.NET Entity Framework.
3. Enter the project name and click Ok.
4. Click on Empty, check the check-box MVC, and click on Ok. An empty MVC web application will open.

(Reference Module-9)

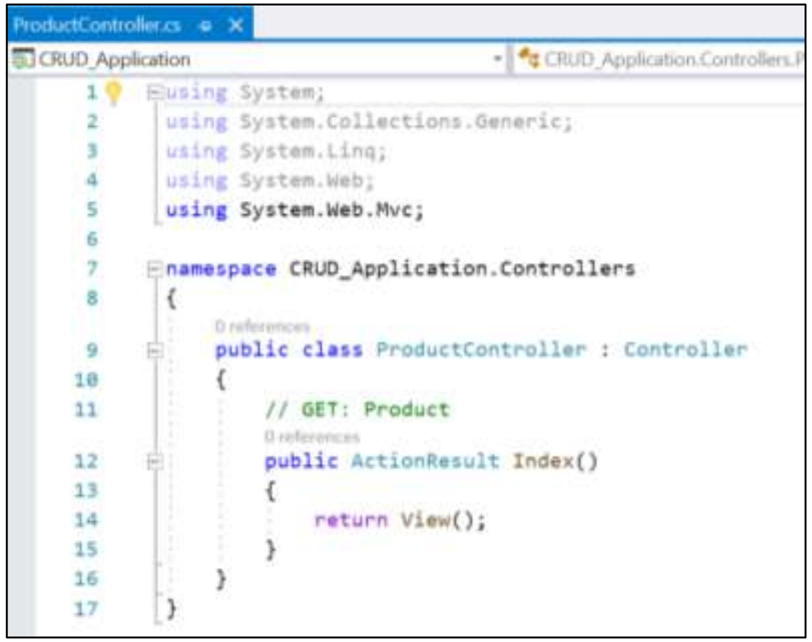


Model

III. Lets create controller –

1. Right click on controller > Add > Controller
2. Select Empty controller
3. Name it as ProductController
4. Save

(Reference Module-3)



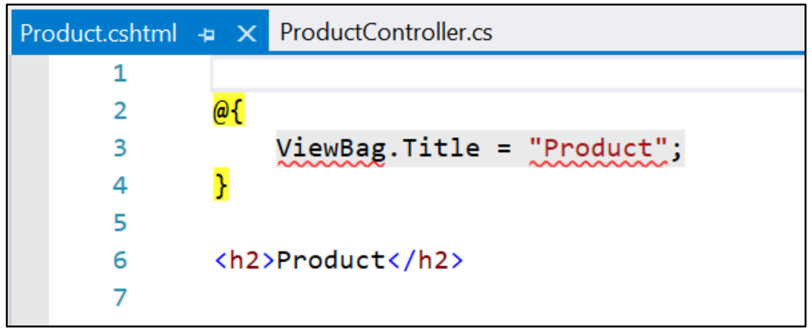
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace CRUD_Application.Controllers
{
    public class ProductController : Controller
    {
        // GET: Product
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Controller

IV. Lets create view for Index Action Method in ProductController –

1. Right click inside Index Action Method > Add View
2. Select MVC view
3. Name it as the action method name
4. Save

(Reference Module-3)



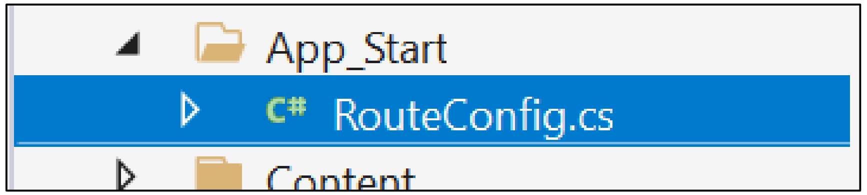
```

@{
    ViewBag.Title = "Product";
}

<h2>Product</h2>

```

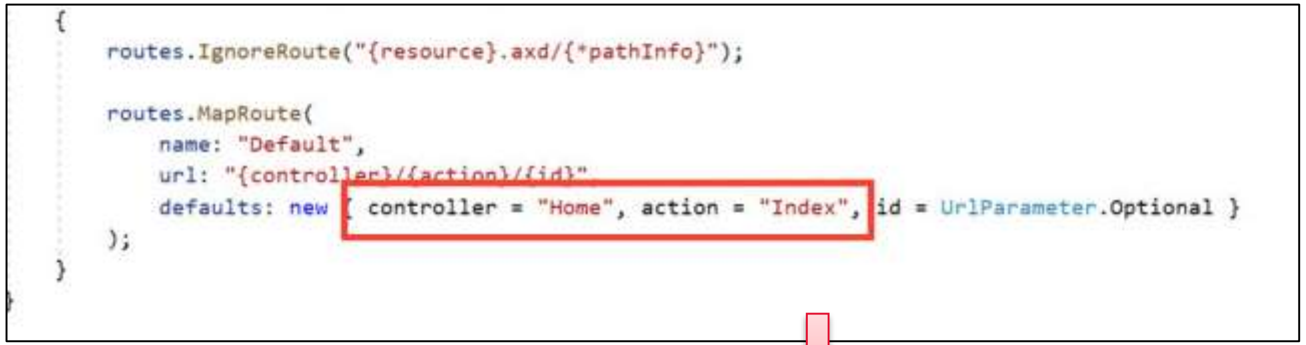
View

V. To make our web page directly open the newly created controller we need to do Routing –

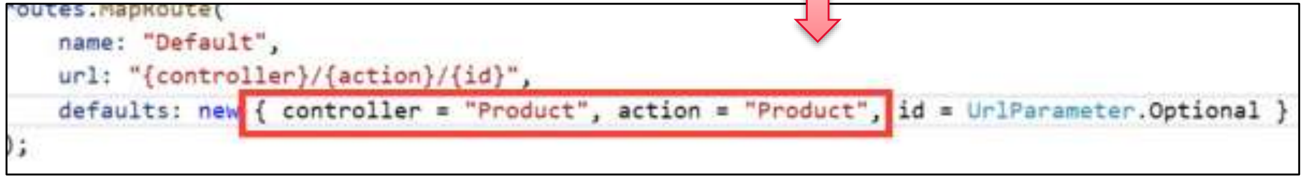
1. Go to App_Start > RouteConfig.cs
2. Change controller from Home to new Controller name and action to new view created
3. Save

In MVC, routing is a process of mapping the browser request to the controller action and return response back. Each MVC application has default routing for the default **HomeController**. We can set custom routing for newly created controller.





```
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```

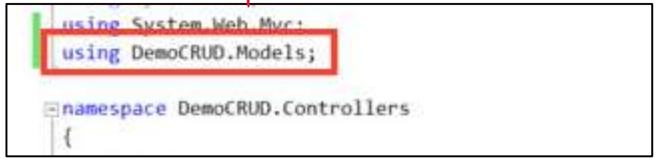


```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Product", action = "Product", id = UrlParameter.Optional }
);
```

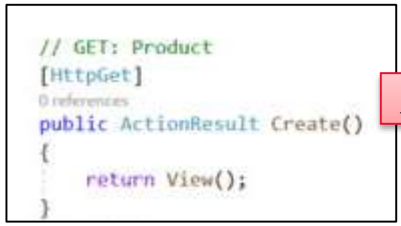
VI. Lets start with **CREATE** action method–

1. Add action method called Create in Product controller
2. Include HttpGet Add a view to Create method
3. Save

Adding Model to the controller

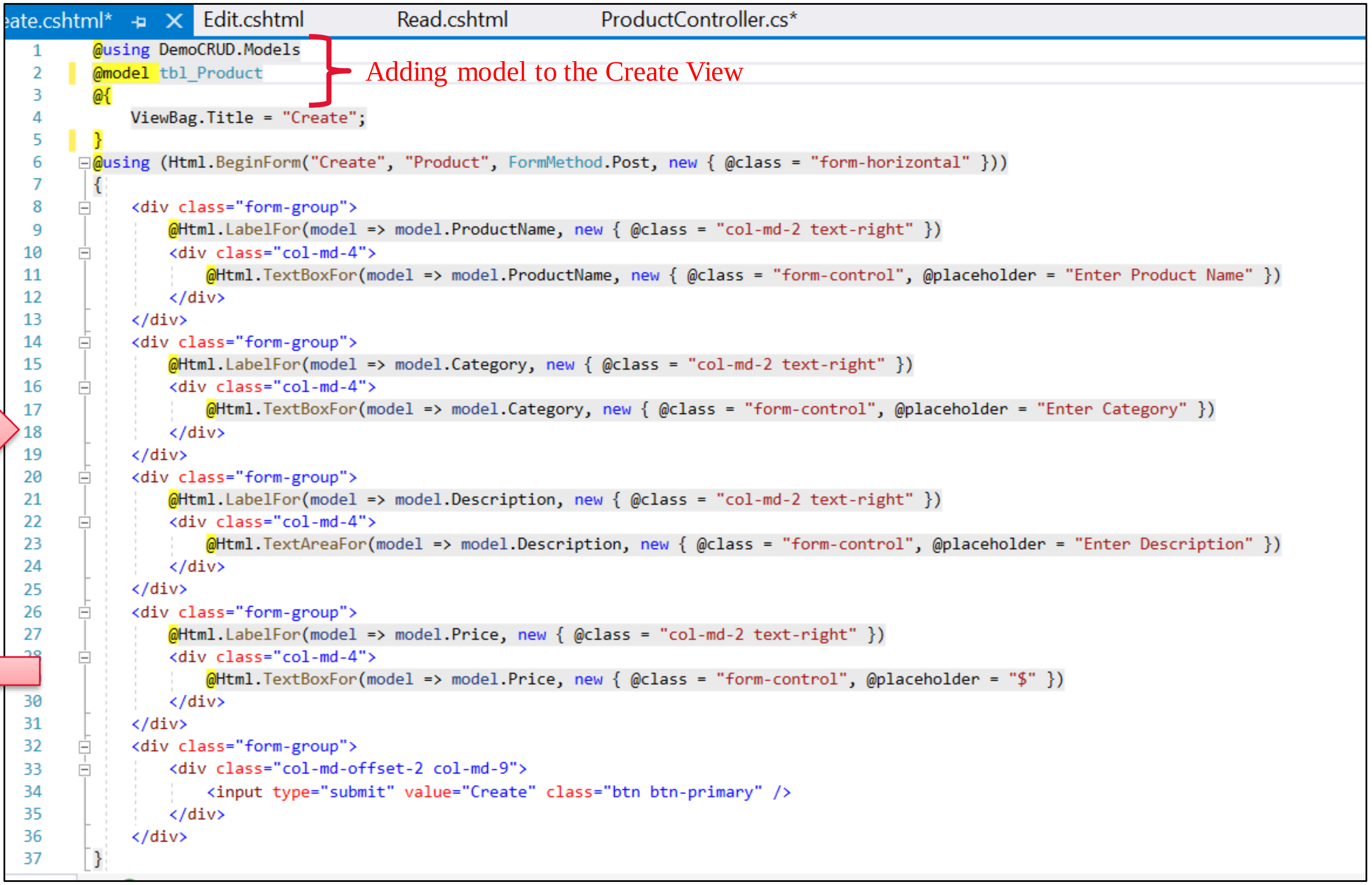


```
using System.Web.Mvc;
using DemoCRUD.Models;

namespace DemoCRUD.Controllers
{
```



```
// GET: Product
[HttpGet]
0 references
public ActionResult Create()
{
    return View();
}
```

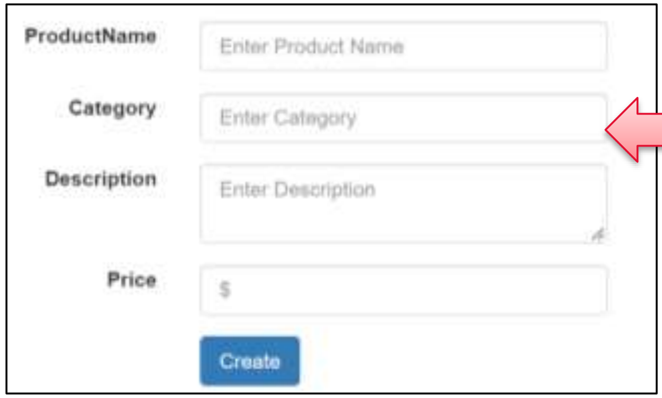
Adding View to Create



Adding model to the Create View

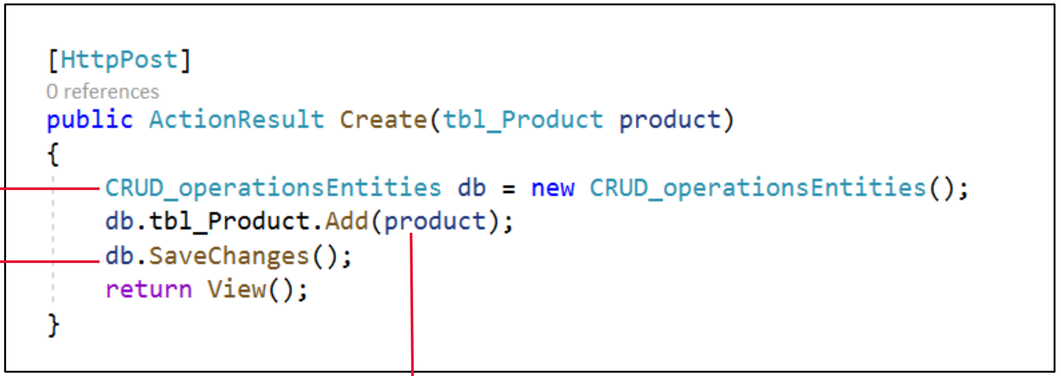
```
@using DemoCRUD.Models
@model tbl_Product
@{
    ViewBag.Title = "Create";
}
@using (Html.BeginForm("Create", "Product", FormMethod.Post, new { @class = "form-horizontal" }))
{
    <div class="form-group">
        @Html.LabelFor(model => model.ProductName, new { @class = "col-md-2 text-right" })
        <div class="col-md-4">
            @Html.TextBoxFor(model => model.ProductName, new { @class = "form-control", @placeholder = "Enter Product Name" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(model => model.Category, new { @class = "col-md-2 text-right" })
        <div class="col-md-4">
            @Html.TextBoxFor(model => model.Category, new { @class = "form-control", @placeholder = "Enter Category" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(model => model.Description, new { @class = "col-md-2 text-right" })
        <div class="col-md-4">
            @Html.TextAreaFor(model => model.Description, new { @class = "form-control", @placeholder = "Enter Description" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(model => model.Price, new { @class = "col-md-2 text-right" })
        <div class="col-md-4">
            @Html.TextBoxFor(model => model.Price, new { @class = "form-control", @placeholder = "$" })
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-9">
            <input type="submit" value="Create" class="btn btn-primary" />
        </div>
    </div>
}
```

Output



ProductName	Enter Product Name
Category	Enter Category
Description	Enter Description
Price	$
	Create

Lets now adding another create action method for HttpPost and saving data to the database –

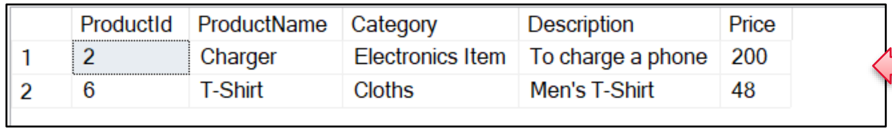


```
[HttpPost]
0 references
public ActionResult Create(tbl_Product product)
{
    CRUD_operationsEntities db = new CRUD_operationsEntities();
    db.tbl_Product.Add(product);
    db.SaveChanges();
    return View();
}
```

Instantiate new object of DbContext from ConnectionString

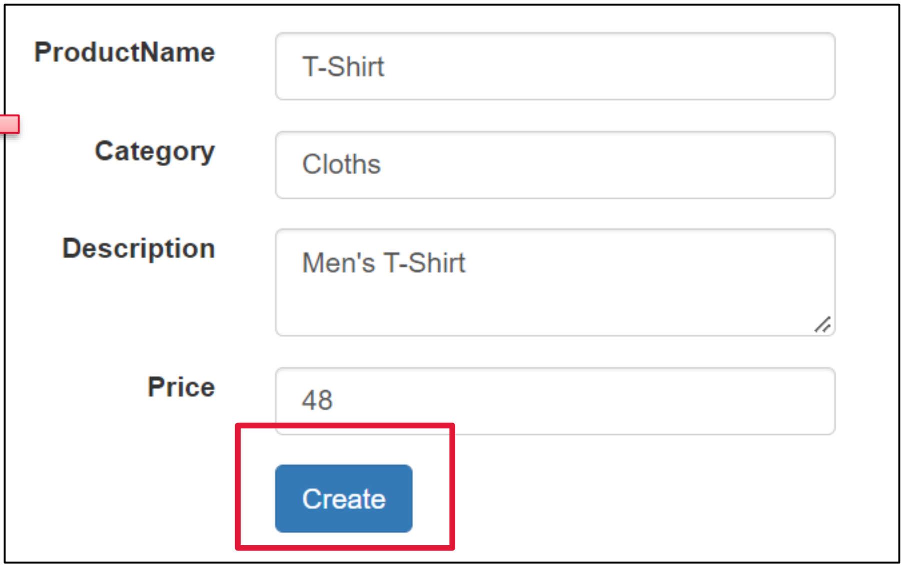
It saves the changes to database. Without this data can't be saved to the database.

Helps to add new data to our database



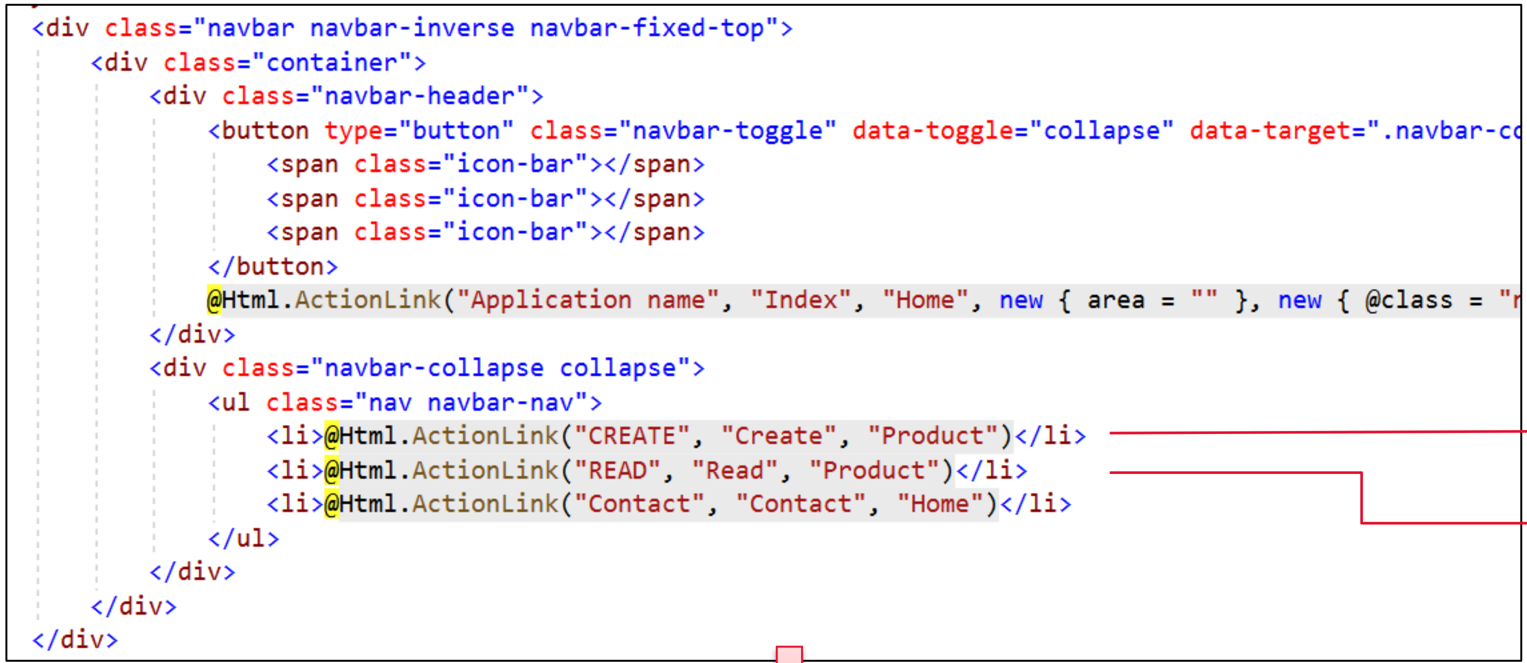
	ProductId	ProductName	Category	Description	Price
1	2	Charger	Electronics Item	To charge a phone	200
2	6	T-Shirt	Cloths	Men's T-Shirt	48

New data added to the database



ProductName	T-Shirt
Category	Cloths
Description	Men's T-Shirt
Price	48

- Before moving forward to Read action method lets make some changes to layout page for better understanding –

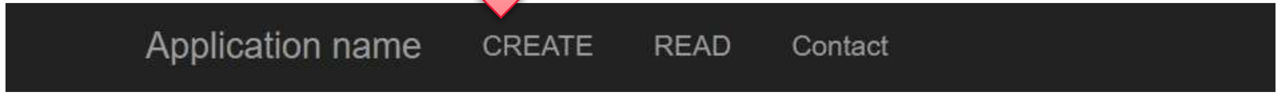


```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-co
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "r
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li>@Html.ActionLink("CREATE", "Create", "Product")</li>
                <li>@Html.ActionLink("READ", "Read", "Product")</li>
                <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
        </div>
    </div>
</div>
```

Adding a CREATE to the nav-bar

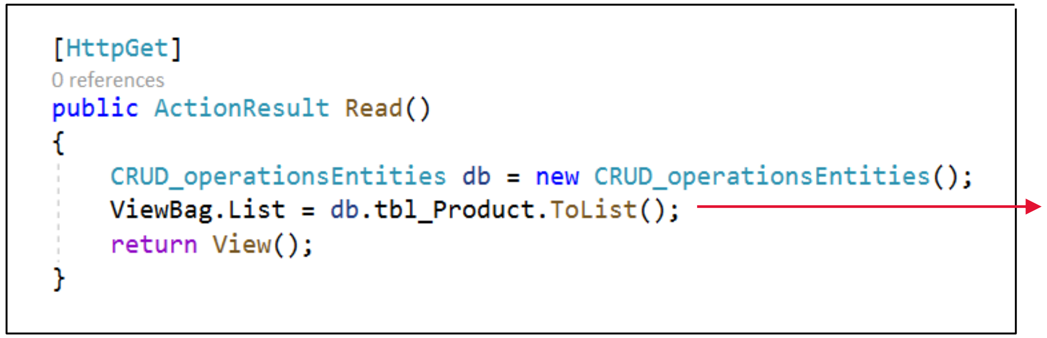
Adding a READ to the nav-bar

OUTPUT



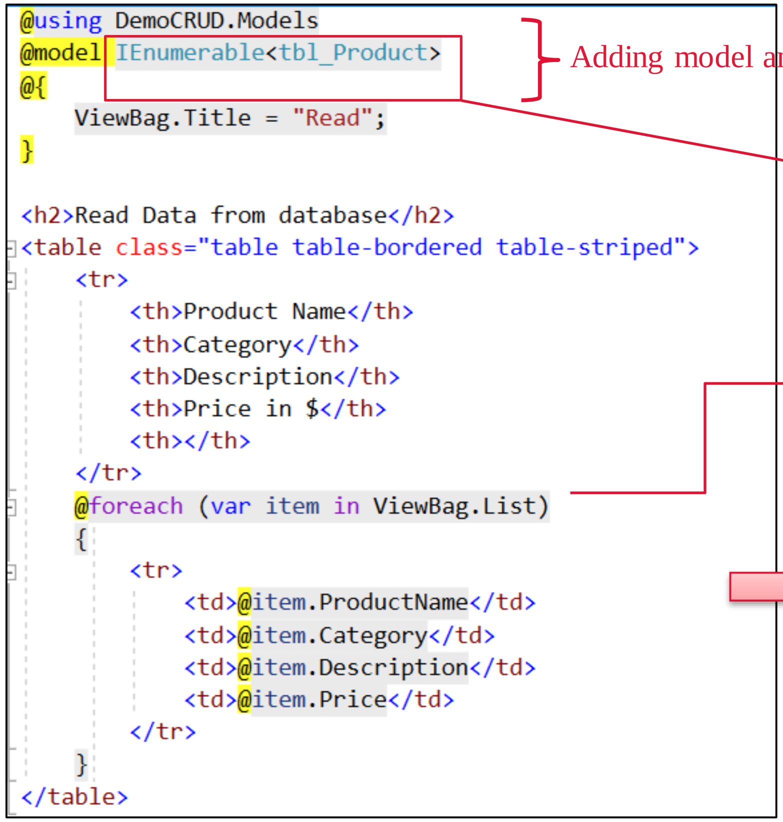
VII. Moving forward to **READ** action method–

1. Add action method called Read in Product controller
2. Include HttpGet
3. Add a view to Read method
4. Save



```
[HttpGet]
0 references
public ActionResult Read()
{
    CRUD_operationsEntities db = new CRUD_operationsEntities();
    ViewBag.List = db.tbl_Product.ToList();
    return View();
}
```

Helps to get data in form of List

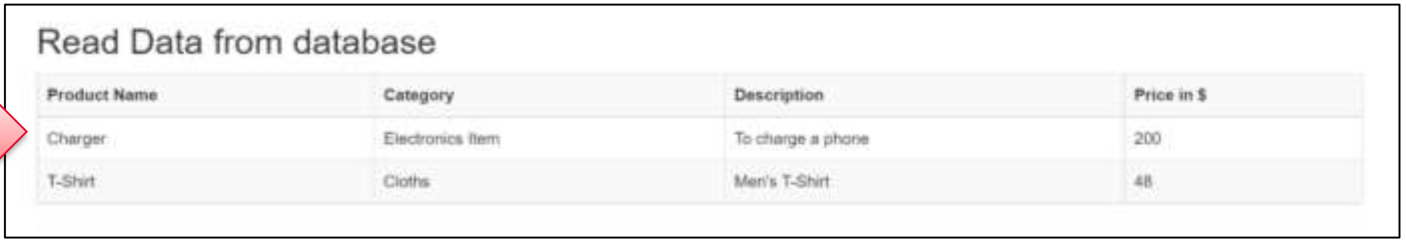


```
@using DemoCRUD.Models
@model IEnumerable<tbl_Product>
@{
    ViewBag.Title = "Read";
}

<h2>Read Data from database</h2>
<table class="table table-bordered table-striped">
    <tr>
        <th>Product Name</th>
        <th>Category</th>
        <th>Description</th>
        <th>Price in $</th>
        <th></th>
    </tr>
    @foreach (var item in ViewBag.List)
    {
        <tr>
            <td>@item.ProductName</td>
            <td>@item.Category</td>
            <td>@item.Description</td>
            <td>@item.Price</td>
        </tr>
    }
</table>
```

Adding model and table to the View

- Used when we want to iterate among our classes using a foreach loop
- We can also use List<> instead of IEnumerable
- Only difference between IEnumerable and List is that IEnumerable is read-only and List is not.

Using foreach to loop through every data on the table

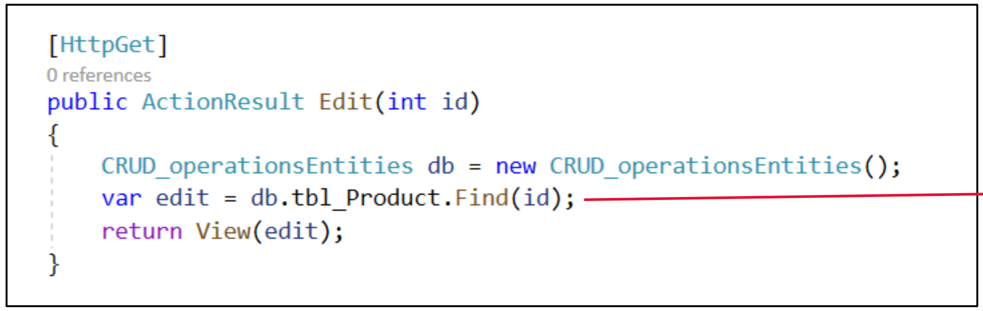


Read Data from database

Product Name	Category	Description	Price in $
Charger	Electronics Item	To charge a phone	200
T-Shirt	Cloths	Men's T-Shirt	48

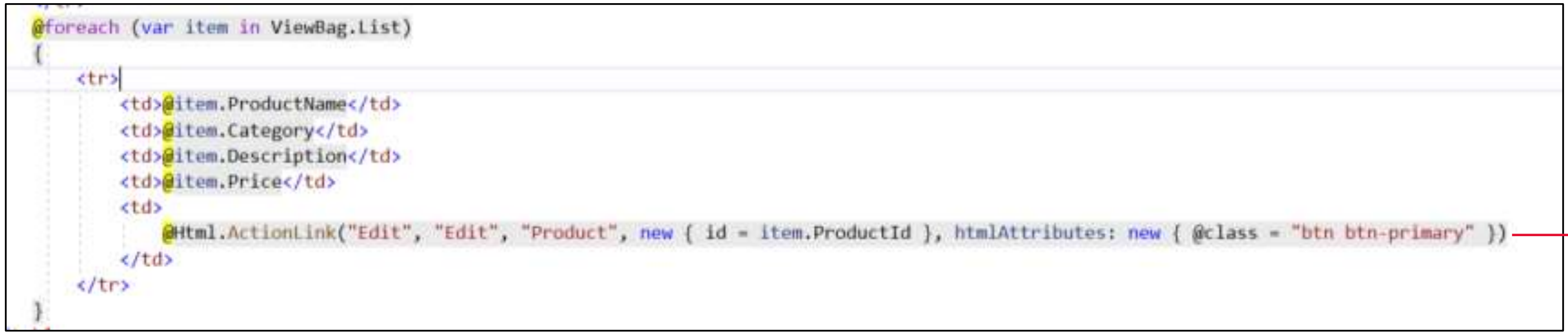
VIII. Next **EDIT** action method–

1. Add action method called Edit in Product controller and pass id as a parameter
2. Include HttpGet
3. Add an Edit button to the column of Read data table using Html.ActionLink(), type = Submit and id = ProductId



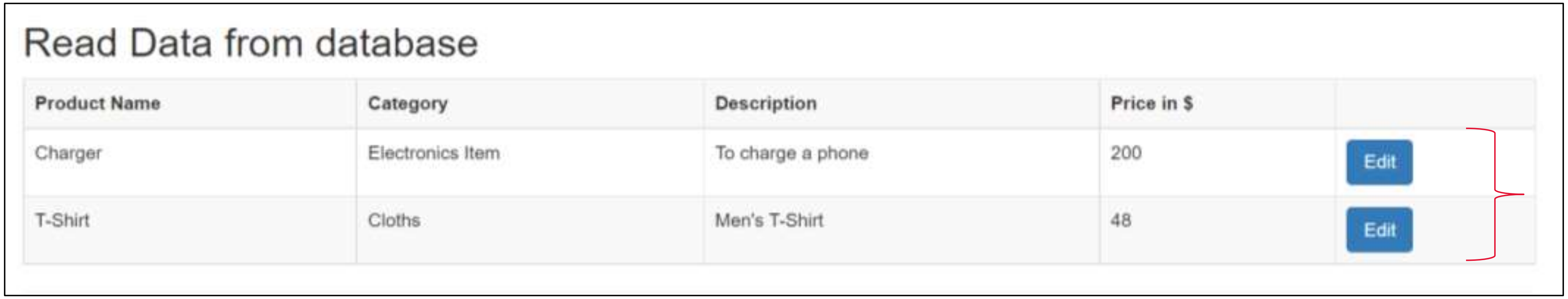
```
[HttpGet]
0 references
public ActionResult Edit(int id)
{
    CRUD_operationsEntities db = new CRUD_operationsEntities();
    var edit = db.tbl_Product.Find(id);
    return View(edit);
}
```

It help us to find a data to be edited.



```
@foreach (var item in ViewBag.List)
{
    <tr>
        <td>@item.ProductName</td>
        <td>@item.Category</td>
        <td>@item.Description</td>
        <td>@item.Price</td>
        <td>
            @Html.ActionLink("Edit", "Edit", "Product", new { id = item.ProductId }, htmlAttributes: new { @class = "btn btn-primary" })
        </td>
    </tr>
}
```

Helps to add a button for Edit

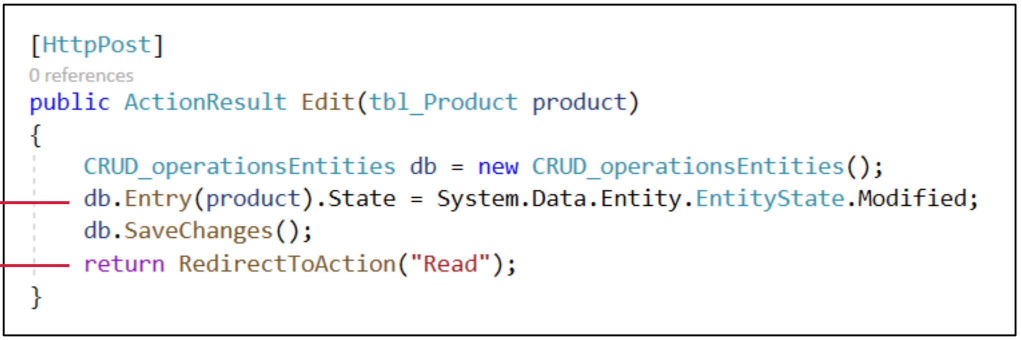


Read Data from database

Product Name	Category	Description	Price in $	
Charger	Electronics Item	To charge a phone	200	Edit
T-Shirt	Cloths	Men's T-Shirt	48	Edit

Output

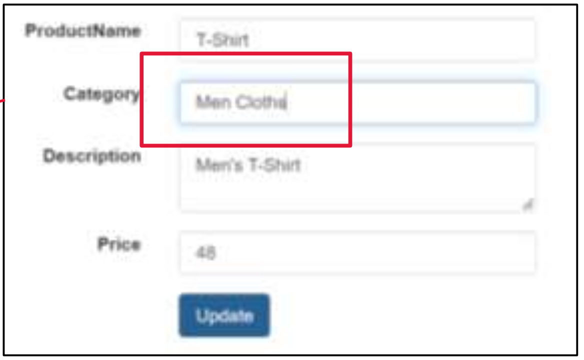
Lets now adding another edit action method for HttpPost and saving data to the database –



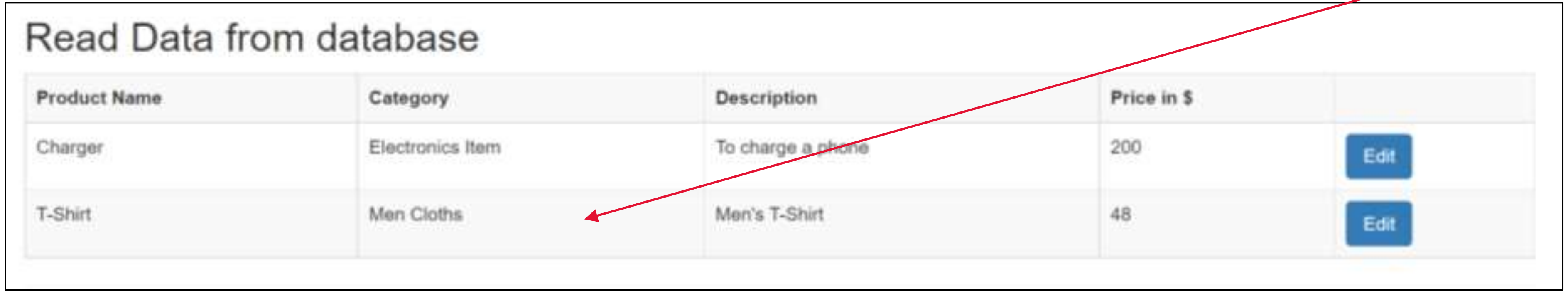
```
[HttpPost]
0 references
public ActionResult Edit(tbl_Product product)
{
    CRUD_operationsEntities db = new CRUD_operationsEntities();
    db.Entry(product).State = System.Data.Entity.EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction("Read");
}
```

Used when a data is modified in the entity that is obtained by the context.

Instead of creating a View for this we will redirect it to another action method Read



ProductName	T-Shirt
Category	Men Cloths
Description	Men's T-Shirt
Price	48



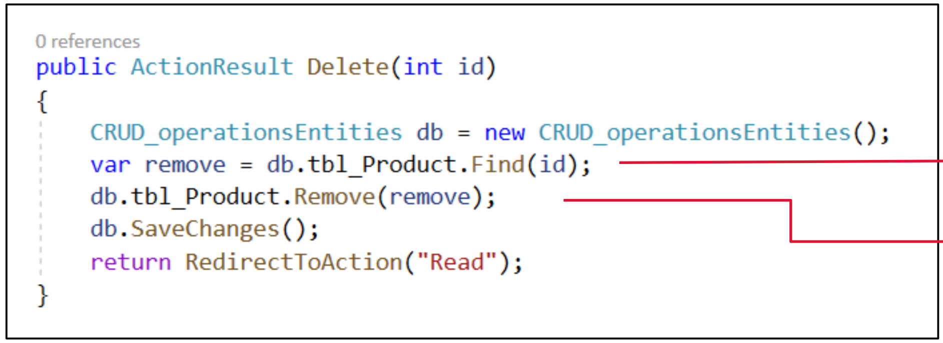
Read Data from database

Product Name	Category	Description	Price in $	
Charger	Electronics Item	To charge a phone	200	Edit
T-Shirt	Men Cloths	Men's T-Shirt	48	Edit

Value Updated

VIII. Lastly add **DELETE** action method–

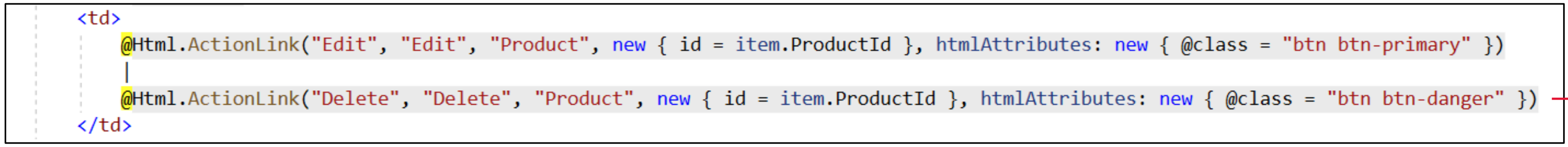
1. Add action method called Delete in Product controller and pass id as a parameter
2. Include HttpGet
3. Add a Delete button to the column of Read data table using Html.ActionLink(), type = Submit and id = ProductId



```
0 references
public ActionResult Delete(int id)
{
    CRUD_operationsEntities db = new CRUD_operationsEntities();
    var remove = db.tbl_Product.Find(id);
    db.tbl_Product.Remove(remove);
    db.SaveChanges();
    return RedirectToAction("Read");
}
```

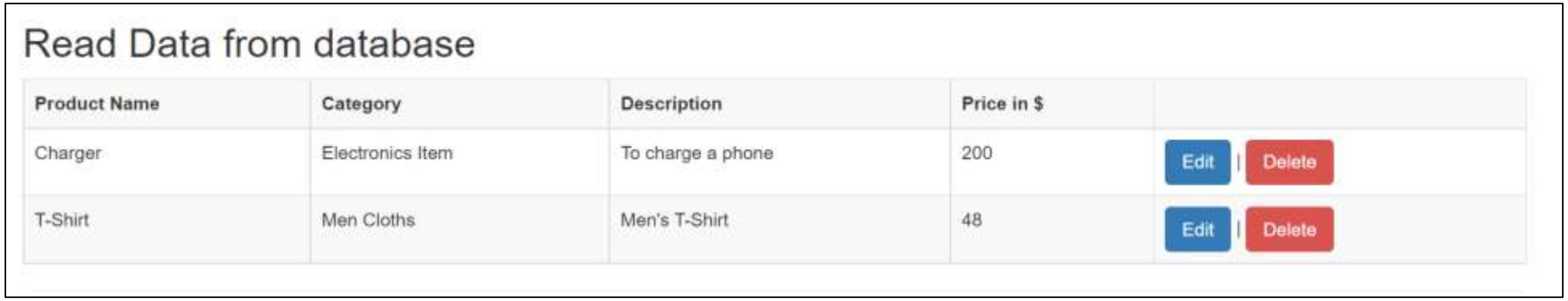
Find Row with Id

Remove data with that id and Save changes



```
<td>
    @Html.ActionLink("Edit", "Edit", "Product", new { id = item.ProductId }, htmlAttributes: new { @class = "btn btn-primary" })
    |
    @Html.ActionLink("Delete", "Delete", "Product", new { id = item.ProductId }, htmlAttributes: new { @class = "btn btn-danger" })
</td>
```

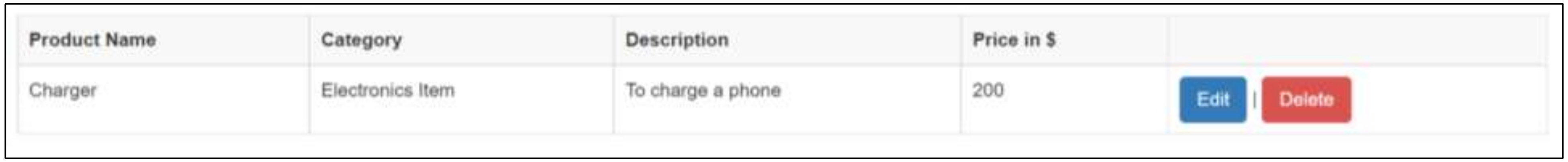
To add Delete Button



Read Data from database

Product Name	Category	Description	Price in $	
Charger	Electronics Item	To charge a phone	200	Edit | Delete
T-Shirt	Men Cloths	Men's T-Shirt	48	Edit | Delete

Delete Button



Product Name	Category	Description	Price in $	
Charger	Electronics Item	To charge a phone	200	Edit | Delete

Row deleted when delete button is clicked

THANK YOU