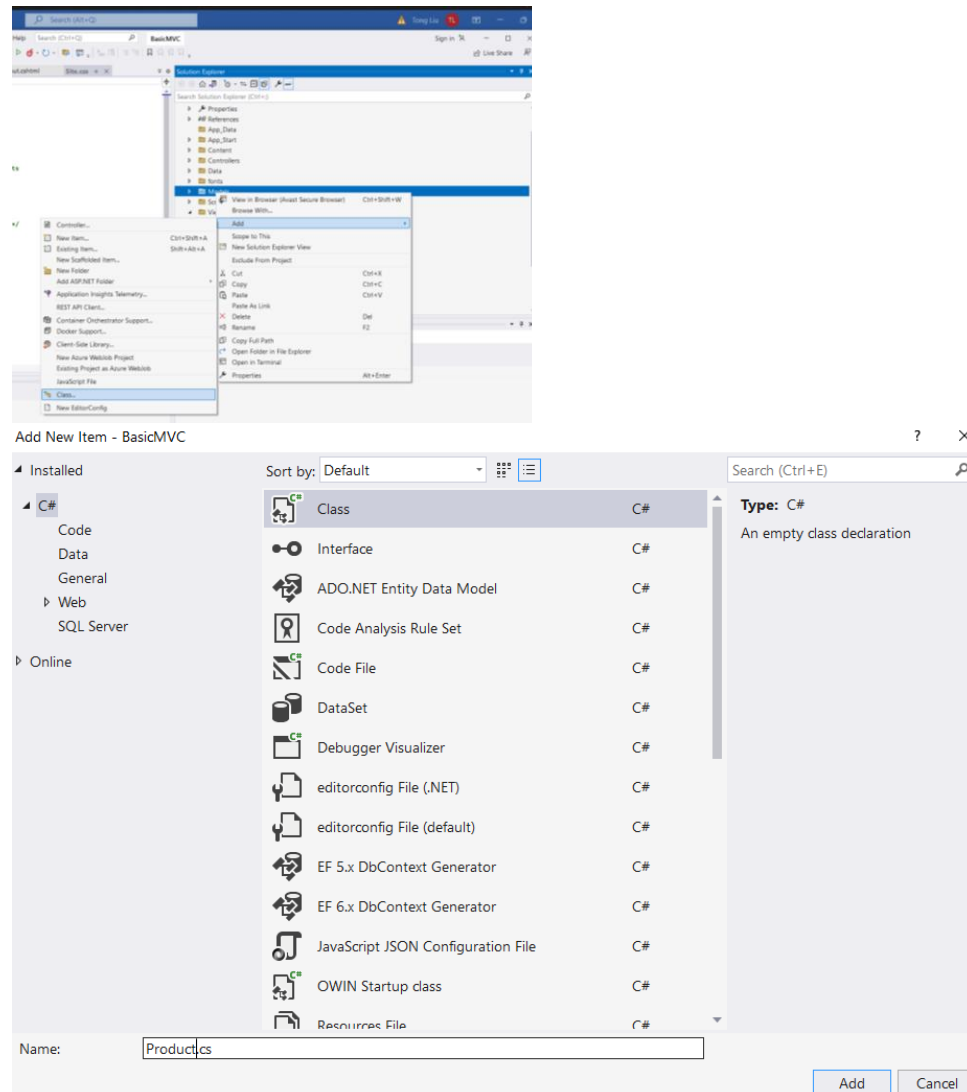


Creating a basic MVC Website Using Entity Framework

Adding Model classes

We are going to add a zero or one-to-many relationship between categories and products, where a category may have many products and a product can belong to none or one category. Right-click the Models folder and choose Add→Class from the menu. Create a new Class called Product.



And add the following code: [you can auto-generate properties in a class in Visual Studio by typing prop and then pressing Tab.] (Add primary key **[Key]** for ID)

```
public class Product
{
    [Key]
    public int ID { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public string Description { get; set; }
    public int? CategoryID { get; set; }
}
```

```

    public virtual Category Category {get; set;}
}

```


You will see a red line under Category in VS; this is because you have not yet created the category class. You can add a new class named Category to the Models folder by Right-clicking the Models folder and choose Add→Class from the menu or hover over the Category, click on the light bulb that appears, and choose the “Generate class ‘Category’ in new file.

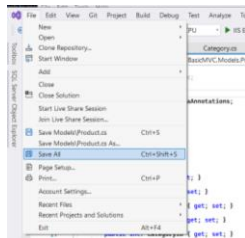
Next, complete the model classes by adding a new class named Category to the Models folder and then adding the following code to the new class (Add primary key **[Key]** for ID) :

```

public class Category
{
    [Key]
    public int ID { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Product> Products { get; set; }
}

```

Save All. Save all the files by clicking  or File -> “Save All”

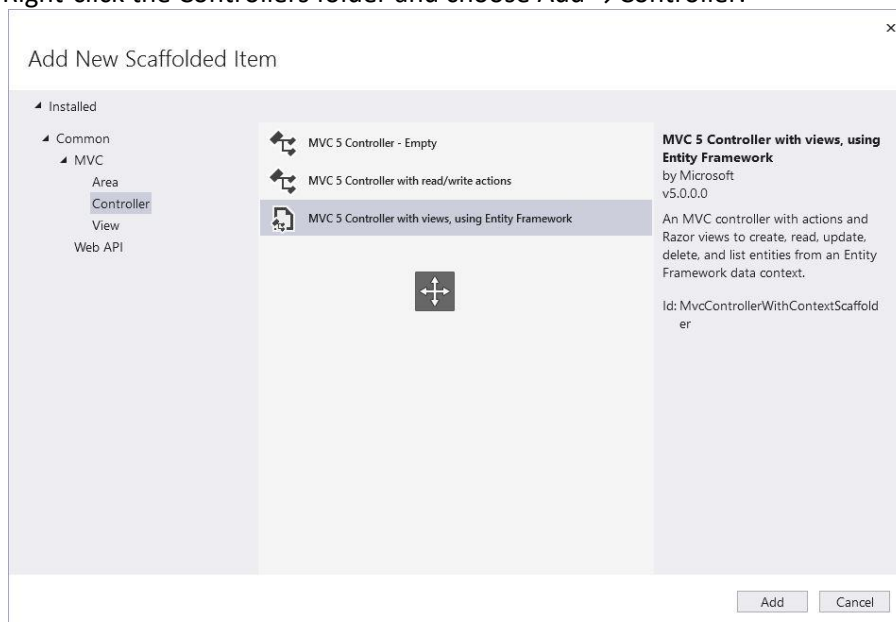


Build the solution by clicking Build → Build Solution or Build -> Rebuild Solution from the Visual Studio menu. Build solution will perform an incremental build. It will only rebuild a project if it is necessary. Rebuild solution will clean and then build the solution from scratch.

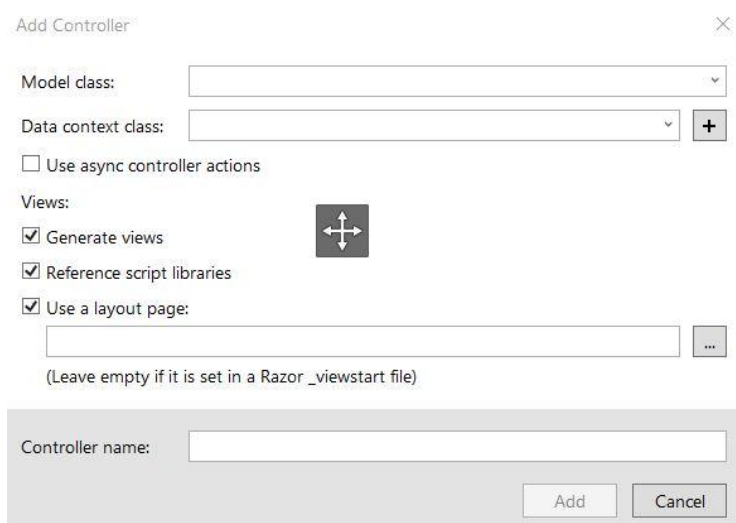
Adding Controllers and Views

Now we need to add some controllers and views in order to display and manage our product and category data.

Right-click the Controllers folder and choose Add →Controller.



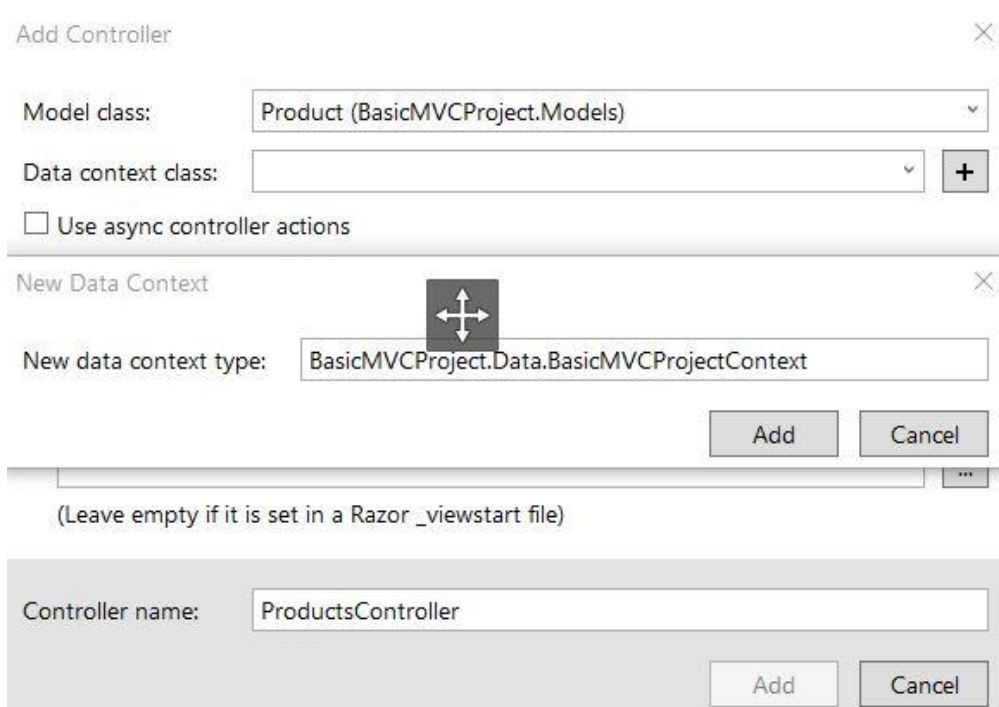
In the Add Scaffold Item window, choose the **MVC 5 Controller with views** option, using **Entity Framework**. Click Add.



Add ProductsController:

In the Add Controller window, choose the following options: (My project is called BasicMVCProject. You might have a different project name. So your Model class and Data Context class might be different.)

- Model class: Product (BasicMVCProject.Models)
- Data Context class: BasicMVCProject.Data.BasicMVCProjectContext
- Ensure that **Generate Views**, **Reference Script Libraries**, and **Use a Layout Page** are all checked
- Leave the controller name set to ProductsController.



Add Controller

Model class:

Data context class:

☐ Use async controller actions

Views:

☒ Generate views

☒ Reference script libraries

☒ Use a layout page

(Leave empty if it is set in a Razor _viewstart file)

Controller name:

Click Add and a new ProductsController class will be automatically created in the Controllers folder. Corresponding views will also be created in the Views\Products folder.

```
public class ProductsController : Controller
{
    private BasicMVContext db = new BasicMVContext();

    // GET: Products
    public ActionResult Index()
    {
        var products = db.Products.Include(p => p.Category);
        return View(products.ToList());
    }

    // GET: Products/Details/5
    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Product product = db.Products.Find(id);
        if (product == null)
        {
            return HttpNotFound();
        }
        return View(product);
    }

    // GET: Products/Create
    public ActionResult Create()
    {
    }
}
```

```

    {
        ViewBag.CategoryID = new SelectList(db.Categories, "ID", "Name");
        return View();
    }

    // POST: Products/Create
    // To protect from overposting attacks, enable the specific properties
    you want to bind to, for
    // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create([Bind(Include =
    "ID,Name,Description,Price,CategoryID")] Product product)
    {
        if (ModelState.IsValid)
        {
            db.Products.Add(product);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        ViewBag.CategoryID = new SelectList(db.Categories, "ID", "Name",
        product.CategoryID);
        return View(product);
    }

    // GET: Products/Edit/5
    public ActionResult Edit(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Product product = db.Products.Find(id);
        if (product == null)
        {
            return HttpNotFound();
        }
        ViewBag.CategoryID = new SelectList(db.Categories, "ID", "Name",
        product.CategoryID);
        return View(product);
    }

    // POST: Products/Edit/5
    // To protect from overposting attacks, enable the specific properties
    you want to bind to, for
    // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit([Bind(Include =
    "ID,Name,Description,Price,CategoryID")] Product product)
    {
        if (ModelState.IsValid)
        {
            db.Entry(product).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        ViewBag.CategoryID = new SelectList(db.Categories, "ID", "Name",
        product.CategoryID);
        return View(product);
    }

```

```

// GET: Products/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Product product = db.Products.Find(id);
    if (product == null)
    {
        return HttpNotFound();
    }
    return View(product);
}

// POST: Products/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Product product = db.Products.Find(id);
    db.Products.Remove(product);
    db.SaveChanges();
    return RedirectToAction("Index");
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
}

```

Add CategoriesController:

The beginning few steps are similar to adding the ProductsController, in the Add Controller window, choose the following options (your Model class and Data Context class might have different project names):

- Model class: Category (BasicMVCProject.Models)
- Data Context class: BasicMVCProject.Data.BasicMVCProjectContext
- Ensure that **Generate Views, Reference Script Libraries, and Use a Layout Page** are all checked
- Leave the controller name set to CategoriesController.

Add Controller
✕

Model class:
Category (BasicMVCProject.Models)

Data context class:
BasicMVCProjectContext (BasicMVCProject.Data)
+

☐ Use async controller actions

Views:

☒ Generate views

☒ Reference script libraries

☒ Use a layout page:
...

(Leave empty if it is set in a Razor _viewstart file)

Controller name:
CategoriesController

Add Cancel

Click Add and a new CategoriesController class will be automatically generated in the Controllers folder. Corresponding views will also be created in the Views\Categories folder.

```

public class CategoriesController : Controller
{
    private BasicMVCContext db = new BasicMVCContext();

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

    // GET: Categories/Details/5
    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Category category = db.Categories.Find(id);
        if (category == null)
        {
            return HttpNotFound();
        }
        return View(category);
    }

    // GET: Categories/Create
    public ActionResult Create()
    {
        return View();
    }

    // POST: Categories/Create
    // To protect from overposting attacks, enable the specific properties
    you want to bind to, for

```

```

// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "ID,Name")] Category category)
{
    if (ModelState.IsValid)
    {
        db.Categories.Add(category);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(category);
}

// GET: Categories/Edit/5
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Category category = db.Categories.Find(id);
    if (category == null)
    {
        return HttpNotFound();
    }
    return View(category);
}

// POST: Categories/Edit/5
// To protect from overposting attacks, enable the specific properties
you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "ID,Name")] Category category)
{
    if (ModelState.IsValid)
    {
        db.Entry(category).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(category);
}

// GET: Categories/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Category category = db.Categories.Find(id);
    if (category == null)
    {
        return HttpNotFound();
    }
    return View(category);
}

// POST: Categories/Delete/5

```



```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Category category = db.Categories.Find(id);
    db.Categories.Remove(category);
    db.SaveChanges();
    return RedirectToAction("Index");
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
}

```

Data->BasicMVCProjectContext will be automatically generated. It provides the Database Context: DbContext. DbContext would need using System.Data.Entity.

```

public class BasicMVContext : DbContext
{
    // You can add custom code to this file. Changes will not be overwritten.
    //
    // If you want Entity Framework to drop and regenerate your database
    // automatically whenever you change your model schema, please use data
    migrations.
    // For more information refer to the documentation:
    // http://msdn.microsoft.com/en-us/data/jj591621.aspx

    public BasicMVContext() : base("name=BasicMVContext")
    {
    }

    public System.Data.Entity.DbSet<BasicMVC.Models.Category> Categories {
get; set; }

    public System.Data.Entity.DbSet<BasicMVC.Models.Product> Products { get;
set; }
}

```



Connection String will be automatically generated.

The connectionStrings telling EF how to connect to the database will be automatically added in the main Web.Config file. The connectionStrings section of the main Web.Config file as follows: [Make sure you look at the Web.config file in the root of the project and not the one in the Views folder.]

```
<connectionStrings>
  <add name="BasicMVCContext" connectionString="Data
Source=(localdb)\MSSQLLocalDB; Initial Catalog=BasicMVCContext-20220905103550;
Integrated Security=True; MultipleActiveResultSets=True;
AttachDbFilename=|DataDirectory|BasicMVCContext-20220905103550.mdf"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```

It's also worth noting that you don't need to define a connection string in web.config. If you don't do so, then Entity Framework will use a default one based on the context class.

Using the New Product and Category Views

We can't expect users to navigate to our new views by entering the URL manually in the web browser, so we need to update the main site navigation bar as follows:

1. Open the Views\Shared_Layout.cshtml file.
2. Below the line of code `@Html.ActionLink("Contact", "Contact", "Home")`, add the following code to add links to the Categories and Products index pages:

```
<li>@Html.ActionLink("Shop by Category", "Index", "Categories")</li>
<li>@Html.ActionLink("View all our Products", "Index", "Products")</li>
```

3. Click on the Debug menu and choose Start Without Debugging. The web site will launch.

Application name

Home

About

Contact

Shop by Category

View all our Products

ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

[Learn more »](#)

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

[Learn more »](#)

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

[Learn more »](#)

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

[Learn more »](#)

© 2022 - My ASP.NET Application

Click on the Shop by Category link, When you click the link, The Categories Index view appears. It does not contain any data since we have no data in the database.

[Application name](#) [Home](#) [About](#) [Contact](#) [Shop by Category](#) [View all our Products](#)

Index

[Create New](#)

Name

© 2022 - My ASP.NET Application

After you enter few entries in after clicking shop by category, your web page may look like:

[Application name](#) [Home](#) [About](#) [Contact](#) [Shop by Category](#) [View all our Products](#)

Index

[Create New](#)

Name	
Accessories	Edit Details Delete
Audio Visual	Edit Details Delete
Computer Systems	Edit Details Delete
Components	Edit Details Delete

© 2022 - My ASP.NET Application

You can also view your Products create page which should have a dropdown list pointing to the categories:

[Application name](#) [Home](#) [About](#) [Contact](#) [Shop by Category](#) [View all our Products](#)

Create

Product

Name	<input type="text"/>
Price	<input type="text"/>
Description	<input type="text"/>
CategoryID	<div>Accessories Accessories Audio Visual Computer Systems Components</div>

[Back to List](#)

Add some data, then click Create button.

Index

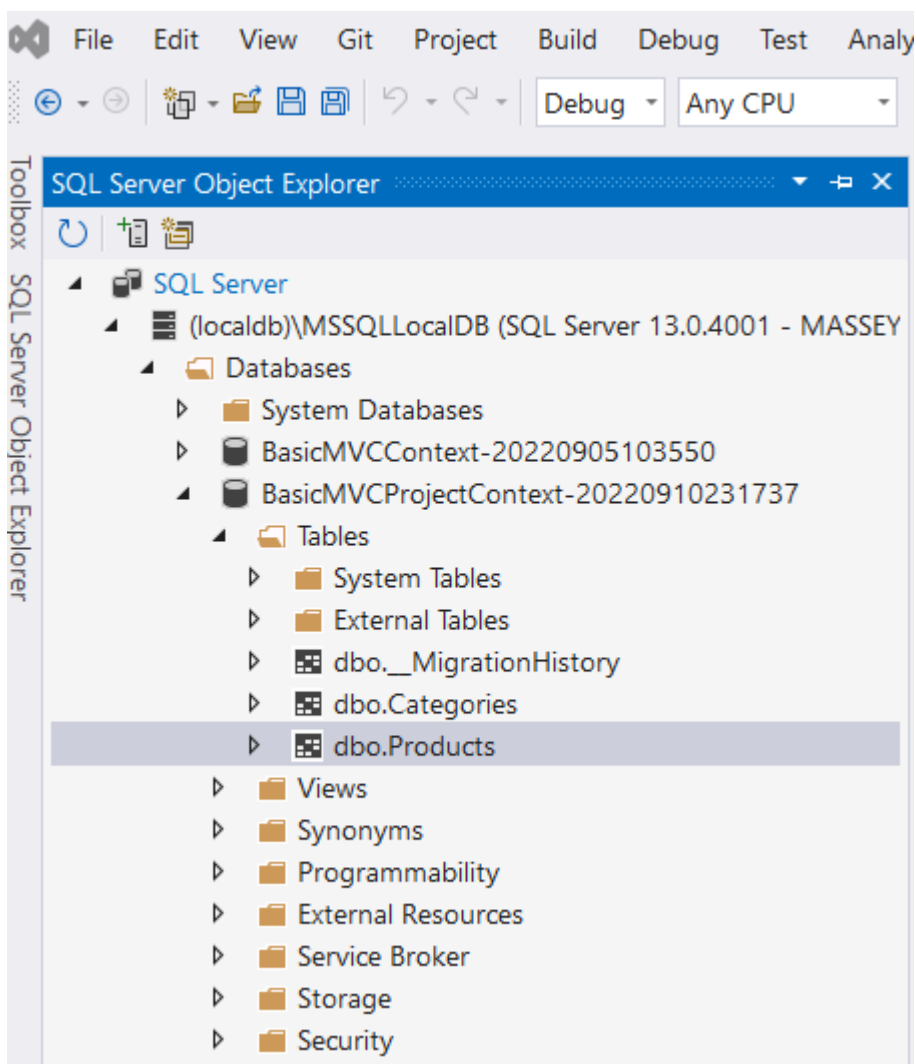
[Create New](#)

Name	Name	Price	Description	
Computer Systems	Windows 10	200.00	Windows 10 OS	Edit Details Delete
Accessories	mouse	15.00	mouse	Edit Details Delete

You can Edit, View the details or Delete the data.

Database

Click View->SQL Server Object Explorer, select your database.



To view the Products table in the database, double click the dbo.Products table. You will see the design view of the table and the SQL code to create the table.

CategoriesController.cs **dbo.Products [Design]** x dbo.Categories [Design] Category.cs _Layout.cshtml

Update Script File: dbo.Products.sql

Name	Data Type	Allow Nulls	Default
ID	int	<input type="checkbox"/>	
Name	nvarchar(MAX)	<input checked="" type="checkbox"/>	
Price	decimal(18,2)	<input type="checkbox"/>	
Description	nvarchar(MAX)	<input checked="" type="checkbox"/>	
CategoryID	int	<input checked="" type="checkbox"/>	

Keys (1)
PK_dbo.Products (Primary Key, Clustered: ID)

Check Constraints (0)

Indexes (1)
IX_CategoryID (CategoryID)

Foreign Keys (1)
FK_dbo.Products_dbo.Categories_CategoryID (ID)

Triggers (0)

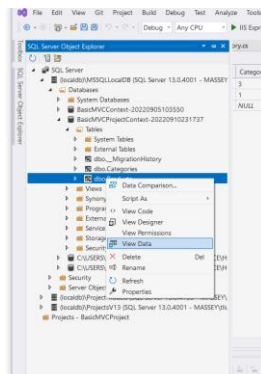
Design T-SQL

```

1 CREATE TABLE [dbo].[Products] (
2     [ID] INT IDENTITY (1, 1) NOT NULL,
3     [Name] NVARCHAR (MAX) NULL,
4     [Price] DECIMAL (18, 2) NOT NULL,
5     [Description] NVARCHAR (MAX) NULL,
6     [CategoryID] INT NULL,
7     CONSTRAINT [PK_dbo.Products] PRIMARY KEY CLUSTERED ([ID] ASC),
8     CONSTRAINT [FK_dbo.Products_dbo.Categories_CategoryID] FOREIGN KEY ([CategoryID]) REFERENCES [dbo].[Categories] ([ID])
9 )
10

```

Viewing the data in the Products table in the database, right click, then select View Data.



You will see the products data we added through the Website.

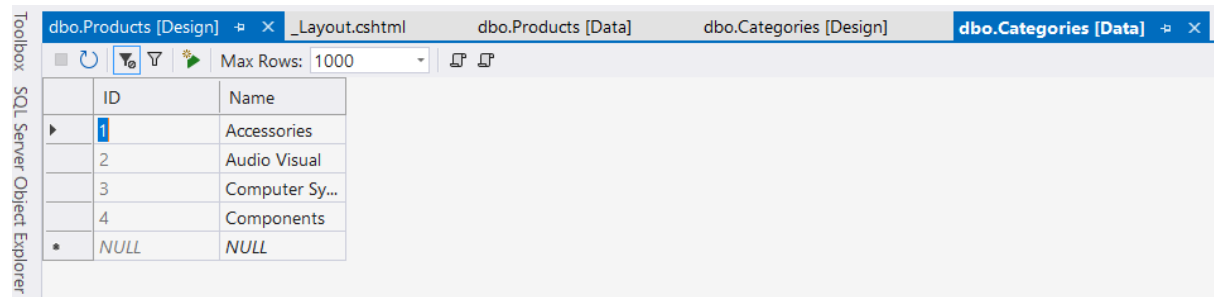
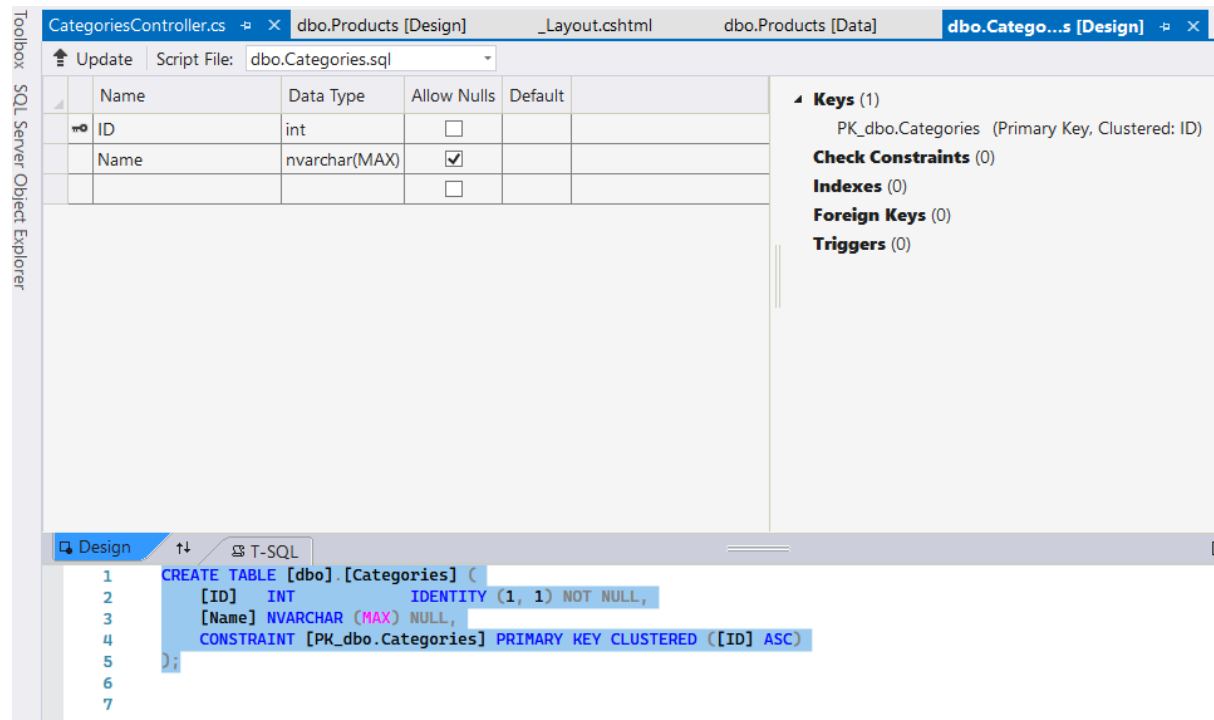
CategoriesController.cs dbo.Products [Design] Category.cs _Layout.cshtml **dbo.Products [Data]** x

Max Rows: 1000

ID	Name	Price	Description	CategoryID
2	Windows 10	200.00	Windows 10 ...	3
3	mouse	15.00	mouse	1
NULL	NULL	NULL	NULL	NULL

SQL Server Object Explorer

You can view the Categories table in the database and the data in the Categories table in the database.



Remove all the unnecessary Usings option.

To remove the unnecessary using statements, hover over the using statements, click on the light bulb that appears, and choose the Remove unnecessary Usings option.

