# Creating and Connecting to a Database

Continuing on with our Home Store application…

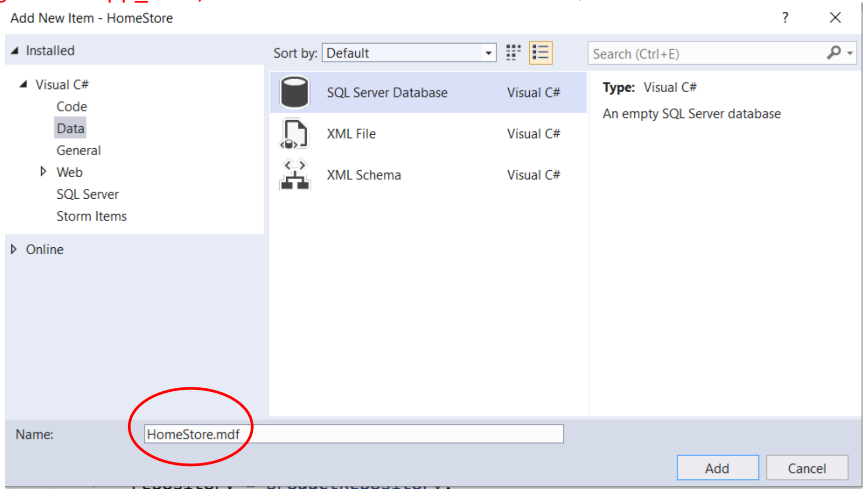
## Preparing a Database

We will use **SQL Server** as the database and will access the database using the **Entity Framework.**

## Creating the Database

We will use **SQL Server** as the database and will access the database using the **Entity Framework.**

Right-click App_Data, then choose Add > New Item > SQL Server Database



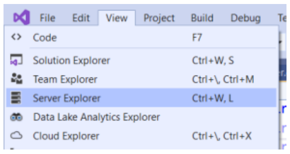
Type: Visual C#
An empty SQL Server database

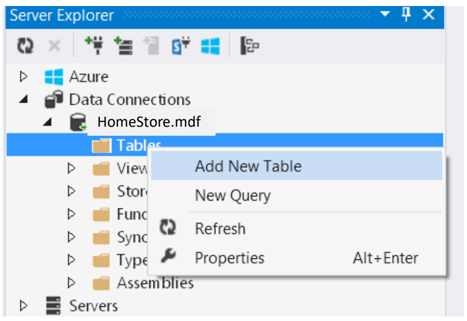
Name: HomeStore.mdf

Name it. The database will be stored inside an MDF file under App_Data in your project.

## Defining the Database Schema

Open the **Server Explorer** and pin it open.

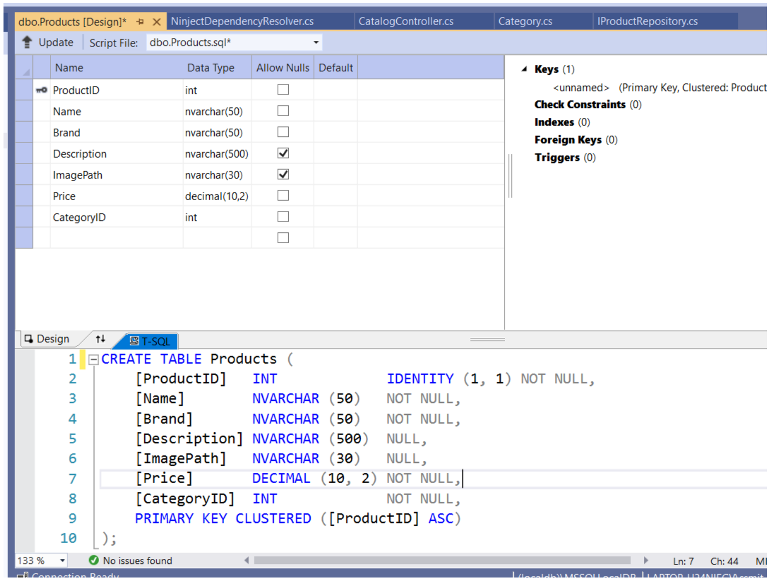

File
Edit
View
Project
Build
Debug
Code F7
Solution Explorer Ctrl+W, S
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+W, L
Data Lake Analytics Explorer
Cloud Explorer Ctrl+\, Ctrl+X



Find HomeStore.mdf under Data Connections and expand it.

Right-click on Tables and choose Add New Table.

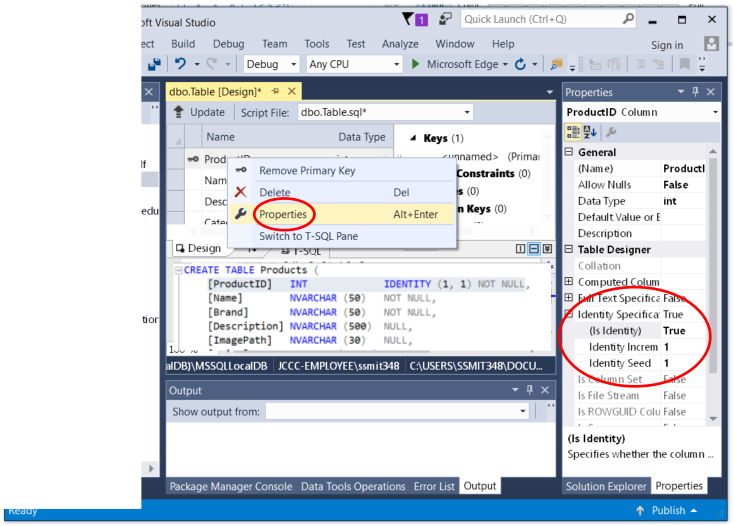
Watch the video to see how I do this!



| Name | Data Type | Allow Nulls | Default |
|---|---|---|---|
| ProductID | int | ☐ | |
| Name | nvarchar(50) | ☐ | |
| Brand | nvarchar(50) | ☐ | |
| Description | nvarchar(500) | ✓ | |
| ImagePath | nvarchar(30) | ✓ | |
| Price | decimal(10,2) | ☐ | |
| CategoryID | int | ☐ | |
| | | ☐ | |

```
CREATE TABLE Products (
    [ProductID]   INT            IDENTITY (1, 1) NOT NULL,
    [Name]        NVARCHAR (50)  NOT NULL,
    [Brand]       NVARCHAR (50)  NOT NULL,
    [Description] NVARCHAR (500) NULL,
    [ImagePath]   NVARCHAR (30)  NULL,
    [Price]       DECIMAL (10, 2) NOT NULL,
    [CategoryID]  INT            NOT NULL,
    PRIMARY KEY CLUSTERED ([ProductID] ASC)
);
```

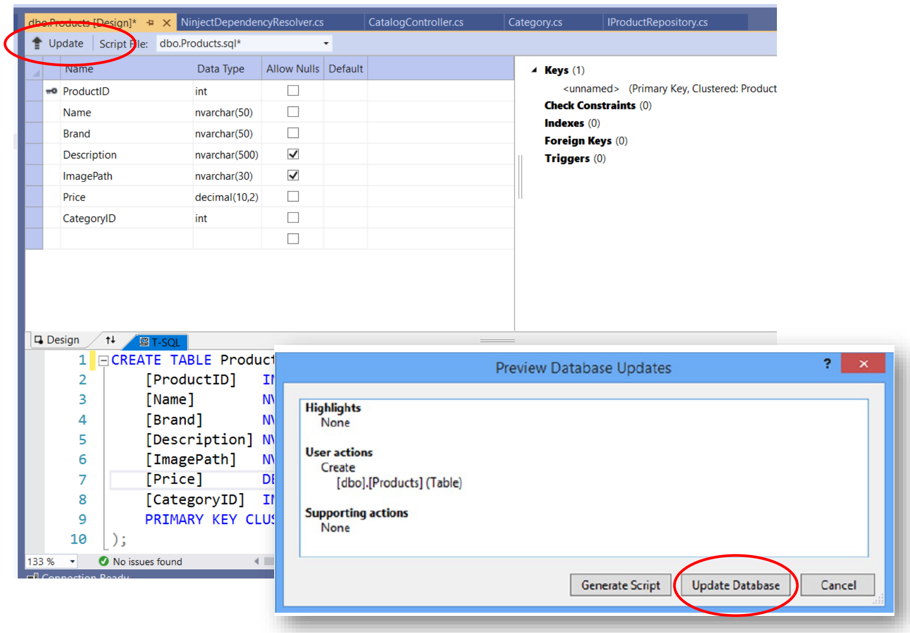
Setting the **IDENTITY** property for the ProductID column means that SQL Server will generate a unique primary key value when data is added to this table.



```
CREATE TABLE Products (
    [ProductID]   INT           IDENTITY (1, 1) NOT NULL,
    [Name]        NVARCHAR (50)  NOT NULL,
    [Brand]       NVARCHAR (50)  NOT NULL,
    [Description] NVARCHAR (500) NULL,
    [ImagePath]   NVARCHAR (30)  NULL,
```

| | |
|---|---|
| **General** | |
| (Name) | ProductI |
| Allow Nulls | False |
| Data Type | int |
| Default Value or E | |
| Description | |
| **Table Designer** | |
| Collation | |
| Computed Colum | |
| Full Text Specifica | False |
| Identity Specifica | True |
| (Is Identity) | True |
| Identity Increm | 1 |
| Identity Seed | 1 |
| Is Column Set | False |
| Is File Stream | False |
| Is ROWGUID Colu | False |

**(Is Identity)**
Specifies whether the column ...



| Name | Data Type | Allow Nulls | Default |
|---|---|---|---|
| ProductID | int | ☐ | |
| Name | nvarchar(50) | ☐ | |
| Brand | nvarchar(50) | ☐ | |
| Description | nvarchar(500) | ✓ | |
| ImagePath | nvarchar(30) | ✓ | |
| Price | decimal(10,2) | ☐ | |
| CategoryID | int | ☐ | |
| | | ☐ | |

Keys (1)
<unnamed> (Primary Key, Clustered: Product
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

```
CREATE TABLE Product
    [ProductID]   IN
    [Name]        NV
    [Brand]       NV
    [Description] NV
    [ImagePath]   NV
    [Price]       DE
    [CategoryID]  IN
    PRIMARY KEY CLUS
);
```

Preview Database Updates

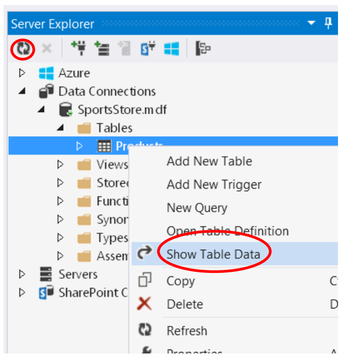
**Highlights**
None

**User actions**
Create
[dbo].[Products] (Table)

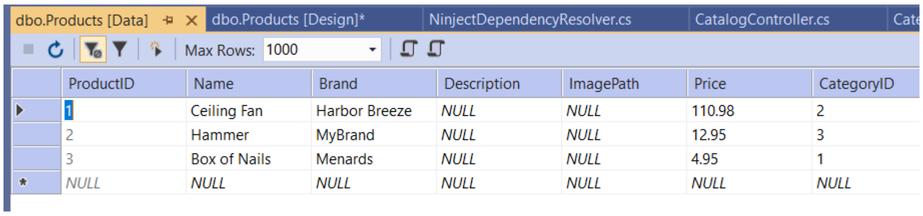
**Supporting actions**
None

## Adding Data to the Database

Refresh the Server Explorer → and then you will see the Products table.

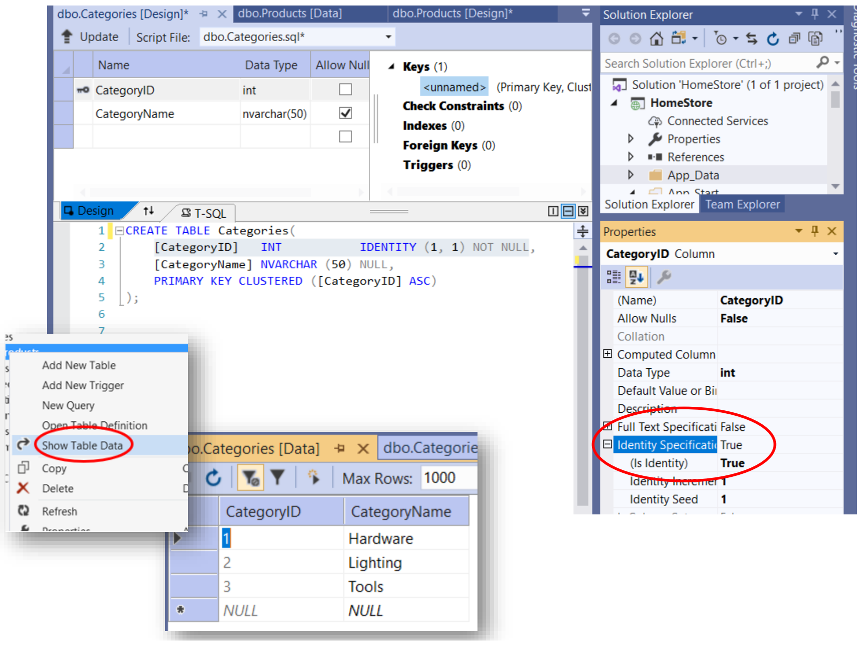


**The ProductID will be automatically assigned by SQL Server.**



| ProductID | Name | Brand | Description | ImagePath | Price | CategoryID |
|---|---|---|---|---|---|---|
| 1 | Ceiling Fan | Harbor Breeze | NULL | NULL | 110.98 | 2 |
| 2 | Hammer | MyBrand | NULL | NULL | 12.95 | 3 |
| 3 | Box of Nails | Menards | NULL | NULL | 4.95 | 1 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Now create another table called Categories.



| Name | Data Type | Allow Null |
|---|---|---|
| CategoryID | int | ☐ |
| CategoryName | nvarchar(50) | ✓ |
|  |  | ☐ |

```
CREATE TABLE Categories(
    [CategoryID]   INT            IDENTITY (1, 1) NOT NULL,
    [CategoryName] NVARCHAR (50) NULL,
    PRIMARY KEY CLUSTERED ([CategoryID] ASC)
);
```

| | |
|---|---|
| (Name) | CategoryID |
| Allow Nulls | False |
| Collation | |
| Computed Column | |
| Data Type | int |
| Default Value or Bi | |
| Description | |
| Full Text Specificati | False |
| Identity Specificati | True |
| (Is Identity) | True |
| Identity Incremen | 1 |
| Identity Seed | 1 |

| CategoryID | CategoryName |
|---|---|
| 1 | Hardware |
| 2 | Lighting |
| 3 | Tools |
| NULL | NULL |

## Creating the Entity Framework Context

The next step is to create a *context* class that will associate the model with the database. Add a new class file in Models called **EFDbContext.cs.**

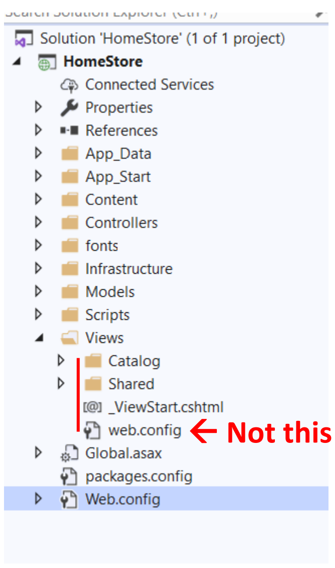
```
using System.Data.Entity;

namespace HomeStore.Models
{
    public class EFDbContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }

    }
}
```

The Entity Framework will use the **Product** model type to represent rows in the **Products** table.

We need to tell the Entity Framework how to **connect to the database** which we will do by adding a **database connection string in the Web.config file.**


```
Solution 'HomeStore' (1 of 1 project)
◢ HomeStore
    Connected Services
  ▷ Properties
  ▷ References
  ▷ App_Data
  ▷ App_Start
  ▷ Content
  ▷ Controllers
  ▷ fonts
  ▷ Infrastructure
  ▷ Models
  ▷ Scripts
  ◢ Views
    ▷ Catalog
    ▷ Shared
      _ViewStart.cshtml
      web.config
  ▷ Global.asax
    packages.config
  ▷ Web.config
```

**← Not this Web.config!**

**This Web.config →**

Put the following right under the **</configSections>** tag.

```
<connectionStrings>

    <add name="EFDbContext" connectionString="Data
Source=(localdb)\MSSQLLocalDB;Initial
Catalog=HomeStore;AttachDbFilename=|DataDirectory|\HomeStore.mdf;
Integrated Security=True" providerName="System.Data.SqlClient"/>

</connectionStrings>
```

## Creating the Product Repository

Add a class file in the Model folder called EFProductRepository.cs.

```
public class EFProductRepository : IProductRepository
{
    EFDbContext context = new EFDbContext();
    public IEnumerable<Product> Products
    {
        get { return context.Products.Include("Category"); }
    }
    public IEnumerable<Category> Categories
    {
        get { return context.Categories; }
    }

}
```

This is the repository class. It **implements the IProductRepository** interface and **uses an instance of EFDbContext** to retrieve data from the database using the Entity Framework.

To use the new repository class, we need to edit the Ninject bindings and replace the mock repository with a binding for the real one.

**NinjectDependencyResolver.cs:**

**Add this to the using directives:**

```
using HomeStore.Models;
```

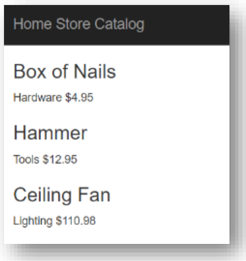
**Remove the Moq code and put this in the AddBindings method:**

```
kernel.Bind<IProductRepository>().To<EFProductRepository>();
```

Since we have two entities that the Entity Framework can link together, let's add this to our view:

```
@p.Category.CategoryName
```

Run the application and we get this:



This is the current List method:

```
public ViewResult List()
{
    return View(repository.Products);
}
```

How could we change it so that the products would be **sorted in order**?

```
public ViewResult List()
{
    return View(repository.Products.OrderBy(x => x.Price));
}
```

Notice how easy it was to use LINQ using method syntax.