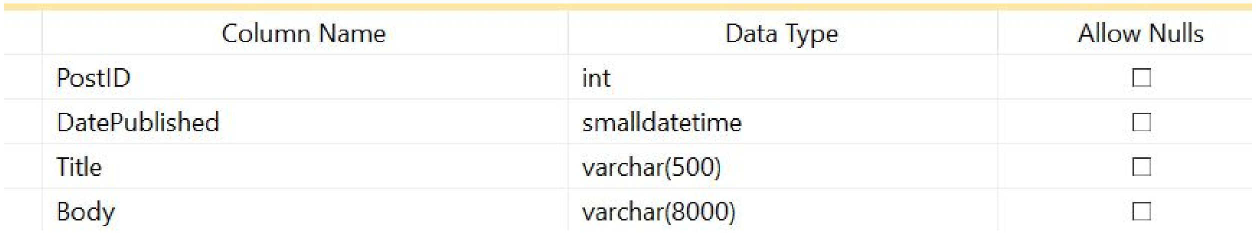
EF - Database First Workflow

1. design table with visual designer
2. EF generates domain classes

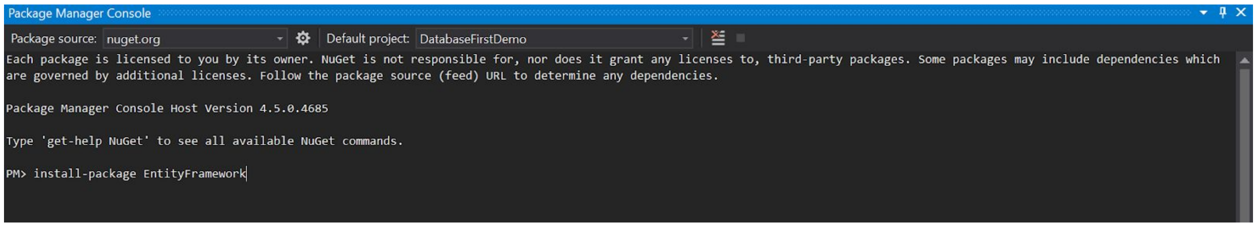
Database First
-create table in Sql Server



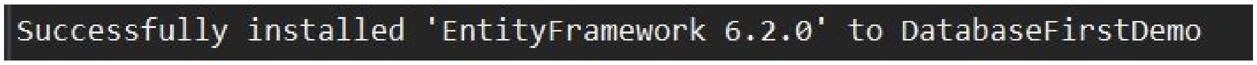
Column Name	Data Type	Allow Nulls
PostID	int	☐
DatePublished	smalldatetime	☐
Title	varchar(500)	☐
Body	varchar(8000)	☐

-create change script and save it

-go to VS create project Console Application
-install entity framework by using package management console
-tools>NuGet Package Manager> Package Manager Console

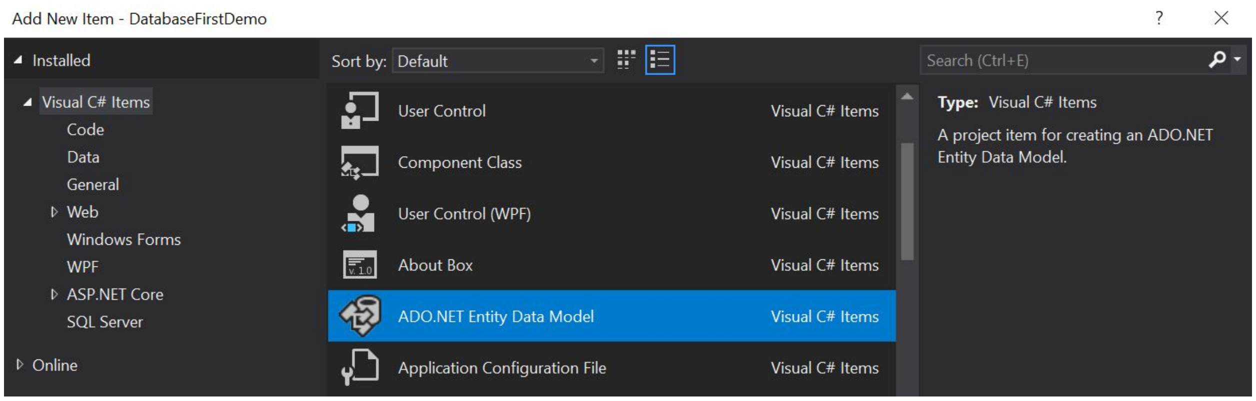


```
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which
are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 4.5.0.4685

Type 'get-help NuGet' to see all available NuGet commands.

PM> install-package EntityFramework
```



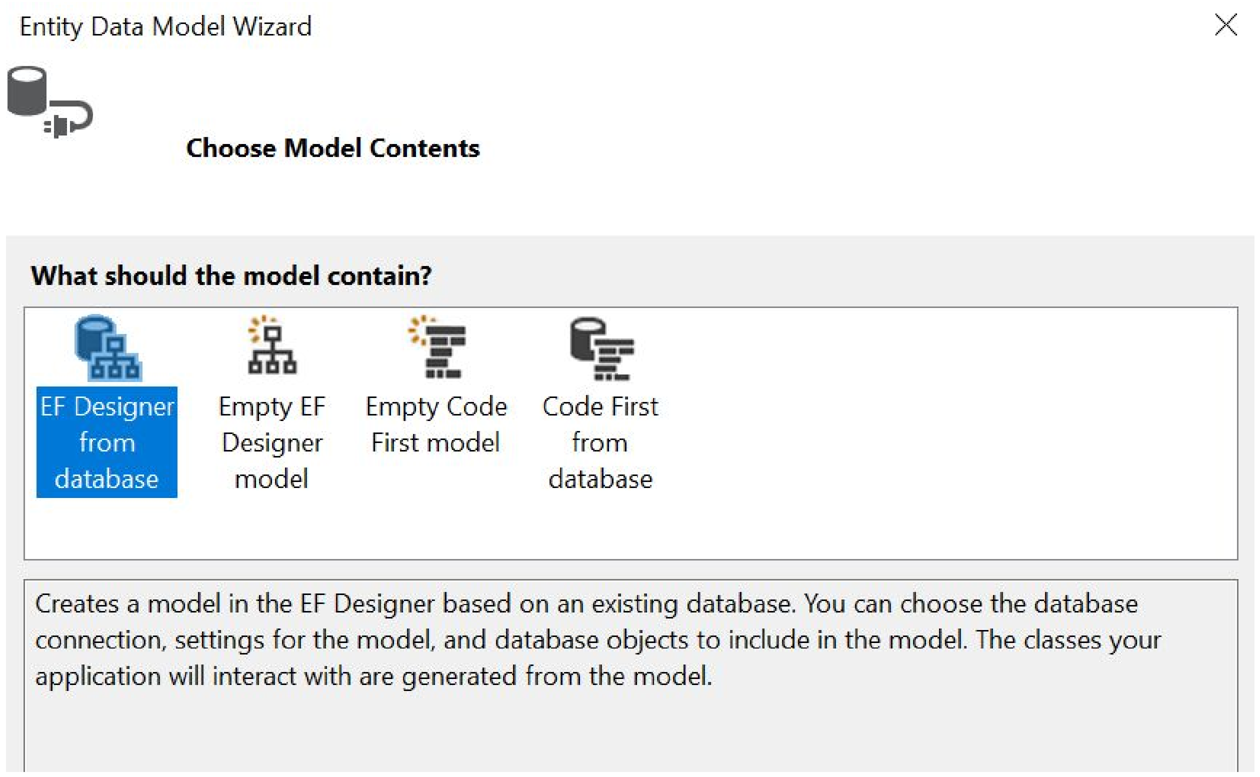
```
Successfully installed 'EntityFramework 6.2.0' to DatabaseFirstDemo
```

-go to solution explorer > right click project > Add new item > ADO.NET Entity Data Model



Add New Item - DatabaseFirstDemo
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
ASP.NET Core
SQL Server
Online
Sort by: Default
User Control — Visual C# Items
Component Class — Visual C# Items
User Control (WPF) — Visual C# Items
About Box — Visual C# Items
ADO.NET Entity Data Model — Visual C# Items
Application Configuration File — Visual C# Items
Search (Ctrl+E)
Type: Visual C# Items
A project item for creating an ADO.NET Entity Data Model.

This is going to be our conceptual model that represents the mapping between the database tables and out domain classes

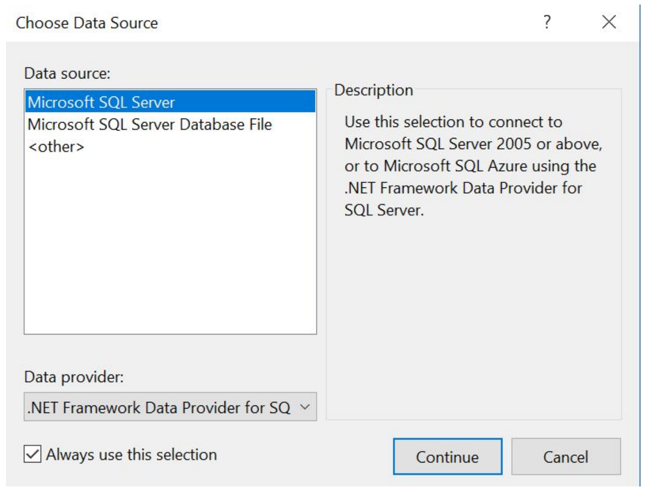

Entity Data Model Wizard

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.

- Use EF Designer from database


Choose Data Source

Data source:

Microsoft SQL Server
Microsoft SQL Server Database File
<other>

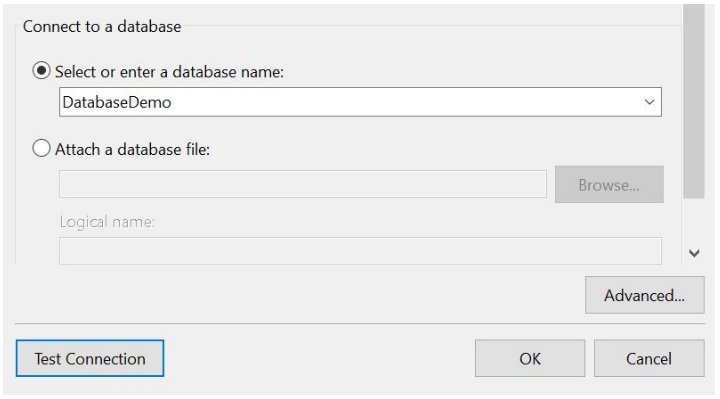
Description

Use this selection to connect to Microsoft SQL Server 2005 or above, or to Microsoft SQL Azure using the .NET Framework Data Provider for SQL Server.

Data provider: .NET Framework Data Provider for SQ

☑ Always use this selection

-add new connection to our database
-connect to database and select database name and test connection



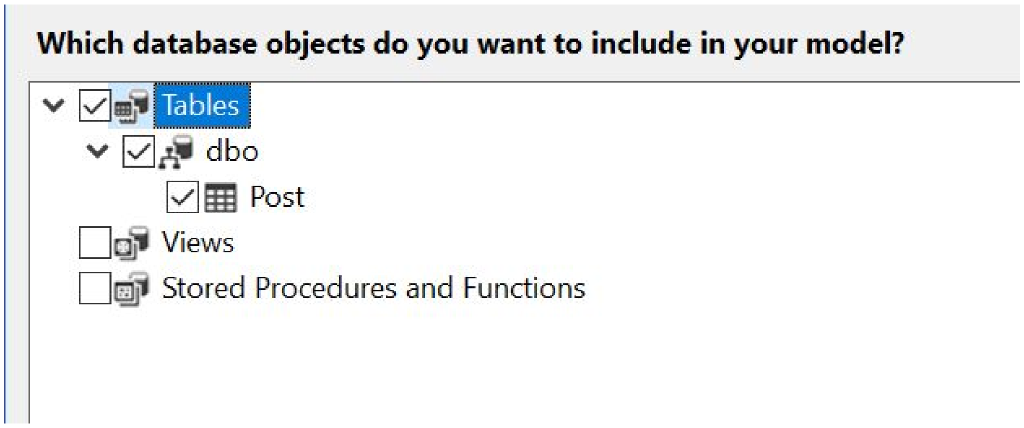
Connect to a database

● Select or enter a database name:

DatabaseDemo

○ Attach a database file:

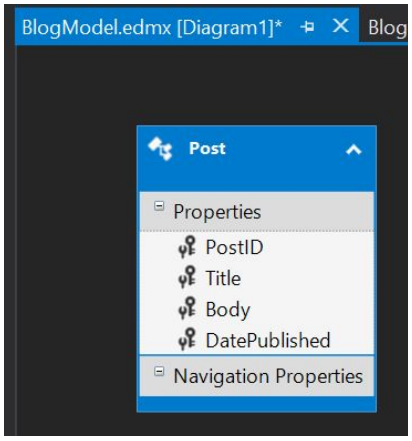
-select Tables and click finish



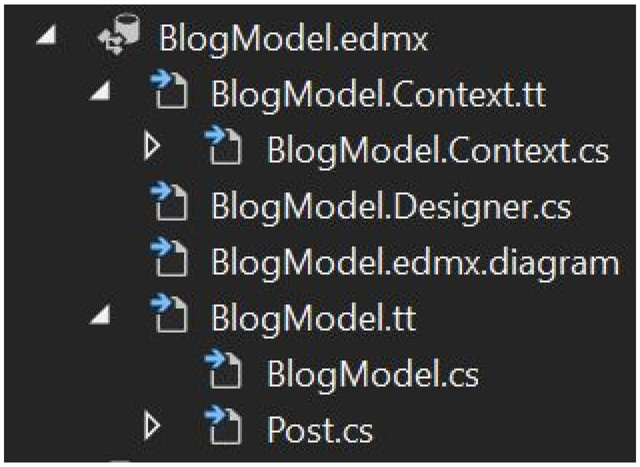
Which database objects do you want to include in your model?

- ✓ Tables
  - ✓ dbo
    - ✓ Post
- Views
- Stored Procedures and Functions

-don't worry about the security warning(choose do not show this message again)
-Here's our entity data model which is stored in a file with a edmx extension



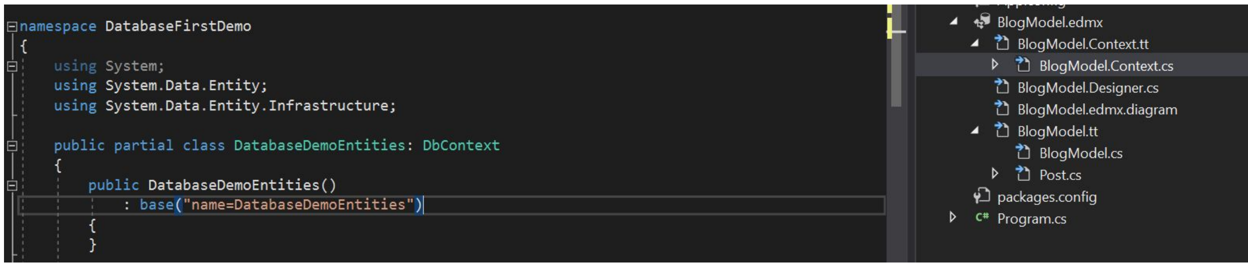
-go to solution bar
Here is all designer generated code



```
◢ BlogModel.edmx
    ◢ BlogModel.Context.tt
        ▷ BlogModel.Context.cs
      BlogModel.Designer.cs
      BlogModel.edmx.diagram
    ◢ BlogModel.tt
        BlogModel.cs
      ▷ Post.cs
```

2 parts are important here

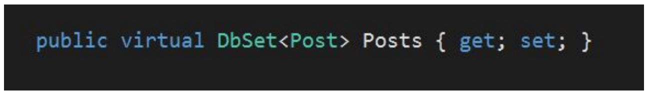
1. BlogModel.Context.tt
    a. tt is stand for template. It's a way to generate code based on a template. BlogModel.Context.cs is the actual generated code. The DatabaseDemoEntities class is derived from **DbContext**. DbContext is a class that is an abstraction over the database. It provides a simple API to load the data from or save it to the database.



```
namespace DatabaseFirstDemo
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class DatabaseDemoEntities: DbContext
    {
        public DatabaseDemoEntities()
            : base("name=DatabaseDemoEntities")
        {
        }
```

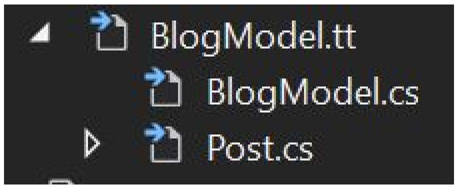
```
◢ BlogModel.edmx
    ◢ BlogModel.Context.tt
        ▷ BlogModel.Context.cs
      BlogModel.Designer.cs
      BlogModel.edmx.diagram
    ◢ BlogModel.tt
        BlogModel.cs
      ▷ Post.cs
  packages.config
▷ Program.cs
```

    b. Here's the property of type DbSet called post. A DbSet represents a table in the database. Because in our database we have a table called post, here we have a deep set of type post called posts. This is a generated code.



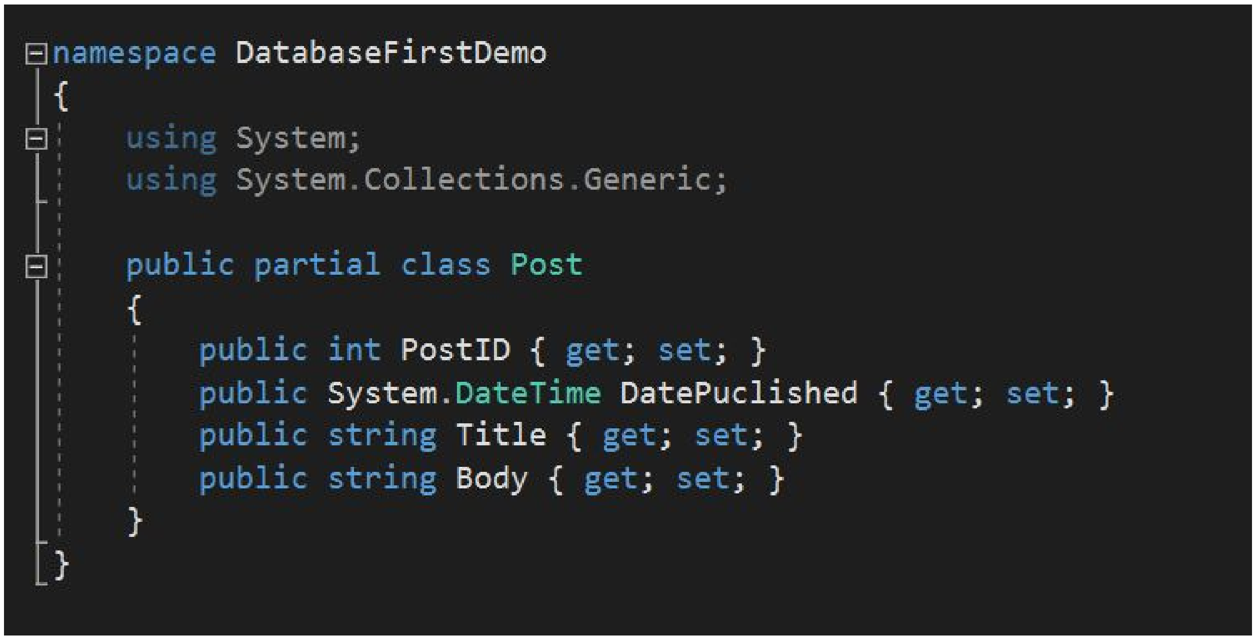
```
public virtual DbSet<Post> Posts { get; set; }
```

2.BlogModel.tt



```
◢ BlogModel.tt
    BlogModel.cs
  ▷ Post.cs
```

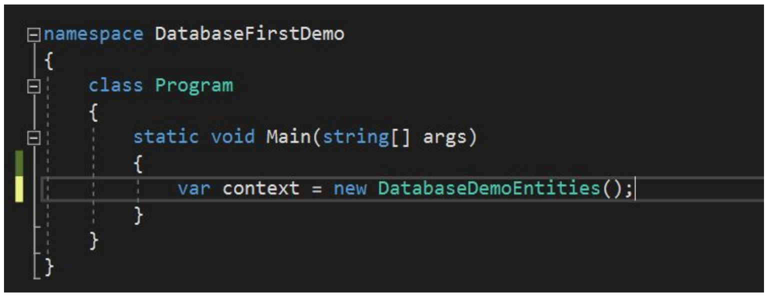
a. Post.cs. It has 4 properties based on the columns that have been created in the database.



```
namespace DatabaseFirstDemo
{
    using System;
    using System.Collections.Generic;

    public partial class Post
    {
        public int PostID { get; set; }
        public System.DateTime DatePuclished { get; set; }
        public string Title { get; set; }
        public string Body { get; set; }
    }
}
```

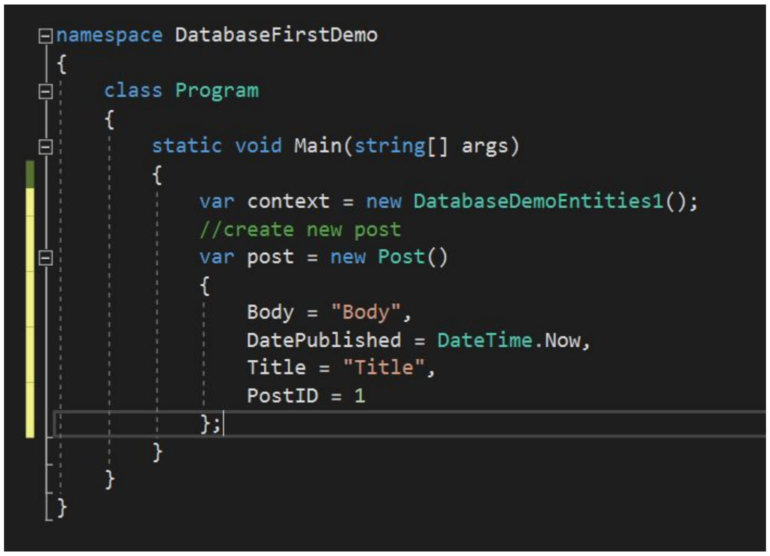
The Key thing here is started with a database then created table and then imported that into the entity data model.
Every time I wanted to make a change in my model I start with the database and then come back to edmx file and refresh it. At this point EF will update my domain classes in the Post.

<<How to use DbContext to work with the database>>
This applies to both database first and code first.
-go to program
-add var context = new DatabaseDemoEntities();



```
namespace DatabaseFirstDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            var context = new DatabaseDemoEntities();
        }
    }
}
```

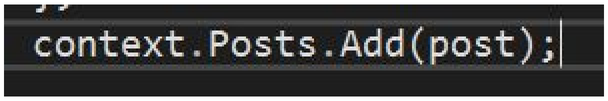
-create post



```
namespace DatabaseFirstDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            var context = new DatabaseDemoEntities1();
            //create new post
            var post = new Post()
            {
                Body = "Body",
                DatePublished = DateTime.Now,
                Title = "Title",
                PostID = 1
            };
        }
    }
}
```

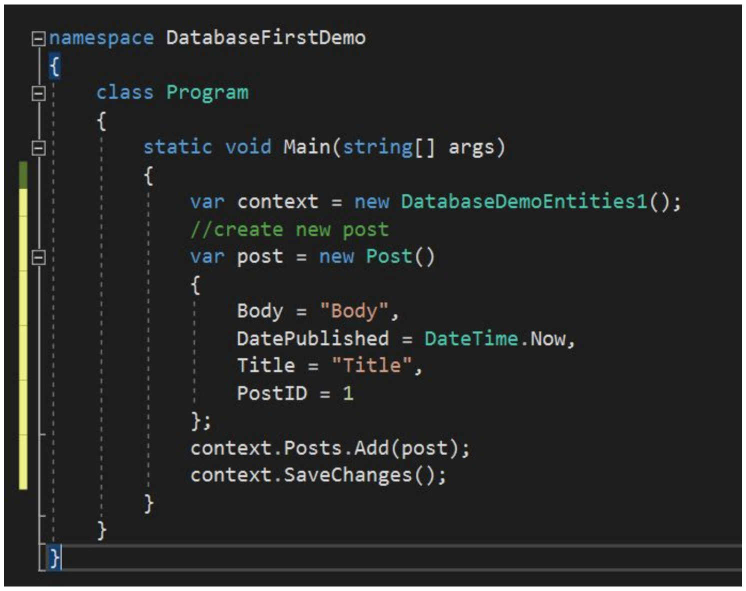
In the real world application most often will use an identity column so we don't have to specify and ID here.

-add this post to db set and save the change



```
context.Posts.Add(post);
```

At this point changes are only in the memory. Nothing is committed to the database yet.



```
namespace DatabaseFirstDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            var context = new DatabaseDemoEntities1();
            //create new post
            var post = new Post()
            {
                Body = "Body",
                DatePublished = DateTime.Now,
                Title = "Title",
                PostID = 1
            };
            context.Posts.Add(post);
            context.SaveChanges();
        }
    }
}
```

Error Message

System.Data.Entity.Infrastructure.DbUpdateException: 'Unable to update the EntitySet 'Post' because it has a DefiningQuery and no <InsertFunction> element exists in the <ModificationFunctionMapping> element to support the current operation.'

Solution

Just Add a primary key to the table. That's it. Problem solved.

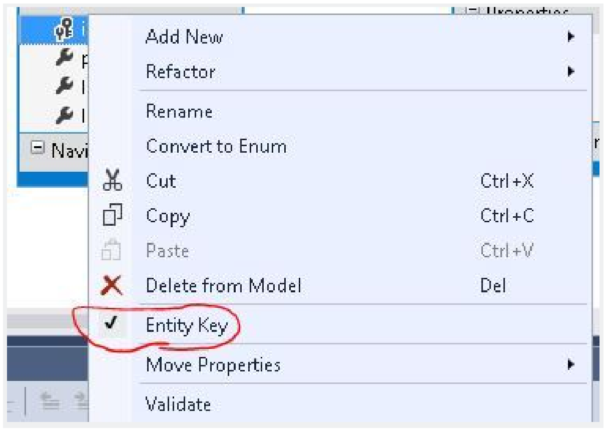
```
ALTER TABLE Post ADD PRIMARY KEY (PostID);
```

Error Message

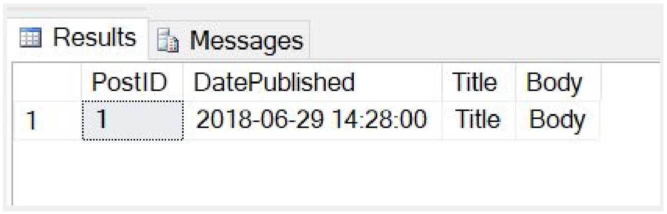
Error 3002: Problem in mapping fragments | c# linq to entities

Solution

Go to edmx and remove entity keys



-Successfully insert post to Post


Results | Messages

	PostID	DatePublished	Title	Body
1	1	2018-06-29 14:28:00	Title	Body

-no need to write a store procedure

-no need to work with ADR.Net classes like sql connection and sql command and the framework

-entity framework took care of all for developers