

# Introduction to Entity Framework

## Entity Framework

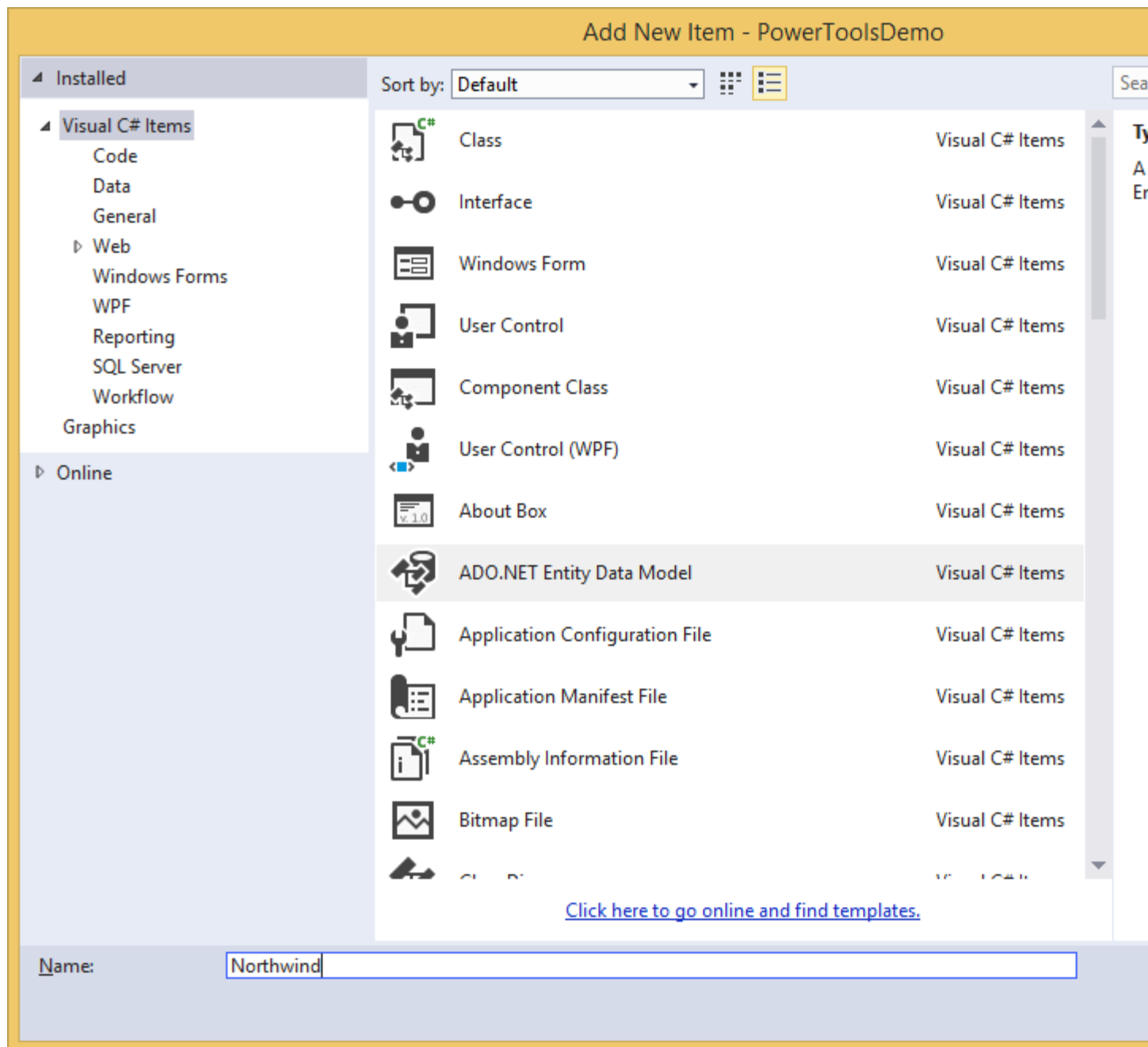
**Entity Framework** (commonly shortened to EF) is an **object-relational mapper** that allows developers to work with relational databases represented by object-oriented classes. In short, it allows you to model your database in your object-oriented design so that you don't need to write as much data-access code. EF **now fully open-sourced**, with the most recent version (V7). For this note, we'll be using EF6.

Entity Framework (EF) is the preferred data access solution and Object-Relational Mapper (ORM) from Microsoft. There are three methodologies you can pick from when building an EF model: Database-First, Model-First, and Code-First.

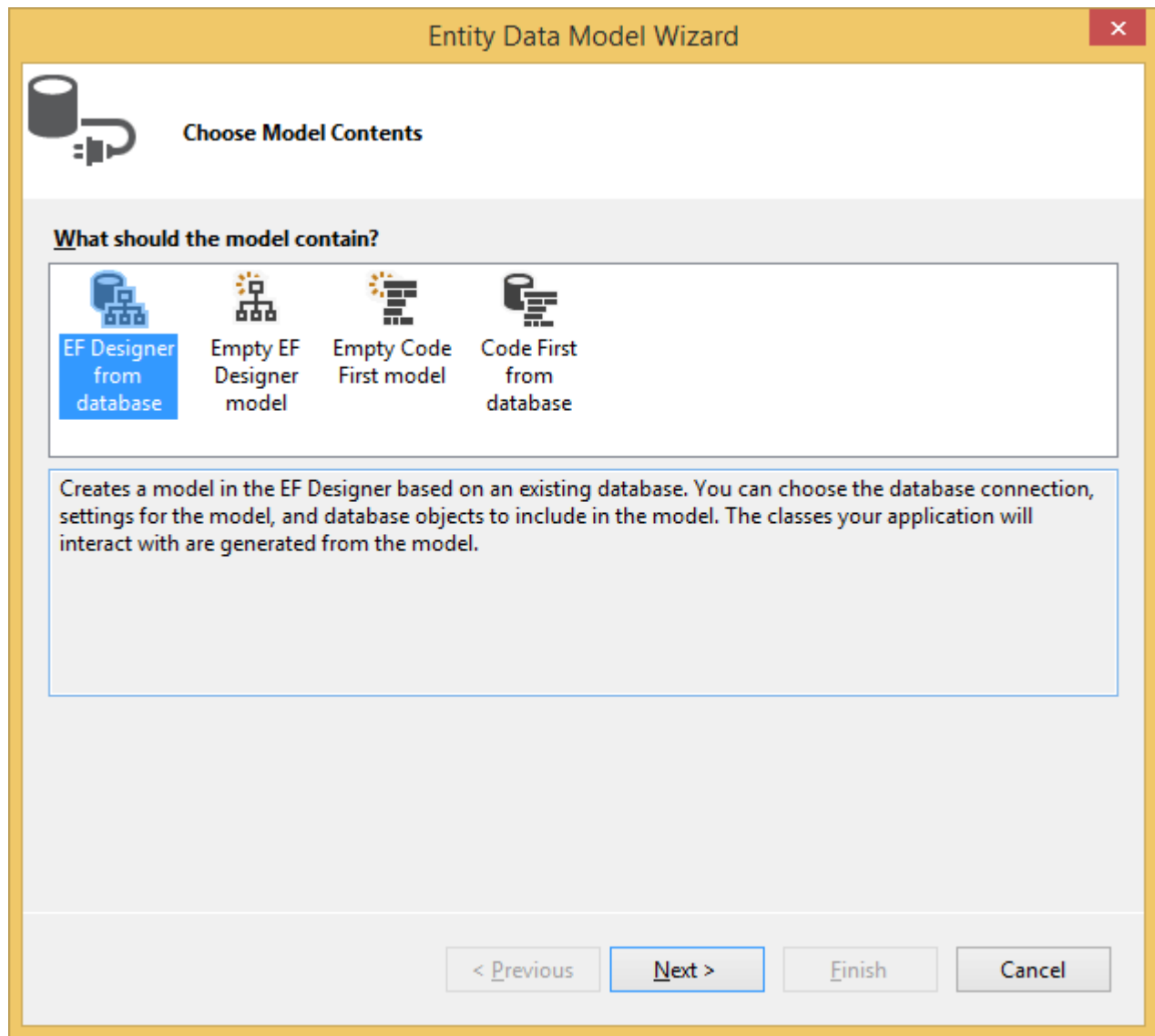
**The primary reason you'd want to use EF is because Visual Studio makes it really easy to design a model (or use an already-existing database to make a model) in EF.**

## Implementing an entity model

This demo assumes that you already have a database that exists that you would like to use. Create a project in Visual Studio Once you've got the project created, right-click on the folder where you want your model to exist and select Add a New Item.




On this dialog, select ADO.NET Entity Data Model, rename it to something appropriate for your project (I'm using the [Northwind](#) database, so my model will be called Northwind) and click *Add*. This will bring up the Choose Model Contents dialog (unless your project doesn't have Entity Framework installed, in which case you'll be prompted to select what version of EF you want. Get Version 6 if it is an option).



We're going to build an EF Designer model from a database. Click *Next*, which will open the Data Connection dialog.

Entity Data Model Wizard

 Choose Your Data Connection

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

NorthwindEntities.mdf New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Connection string:

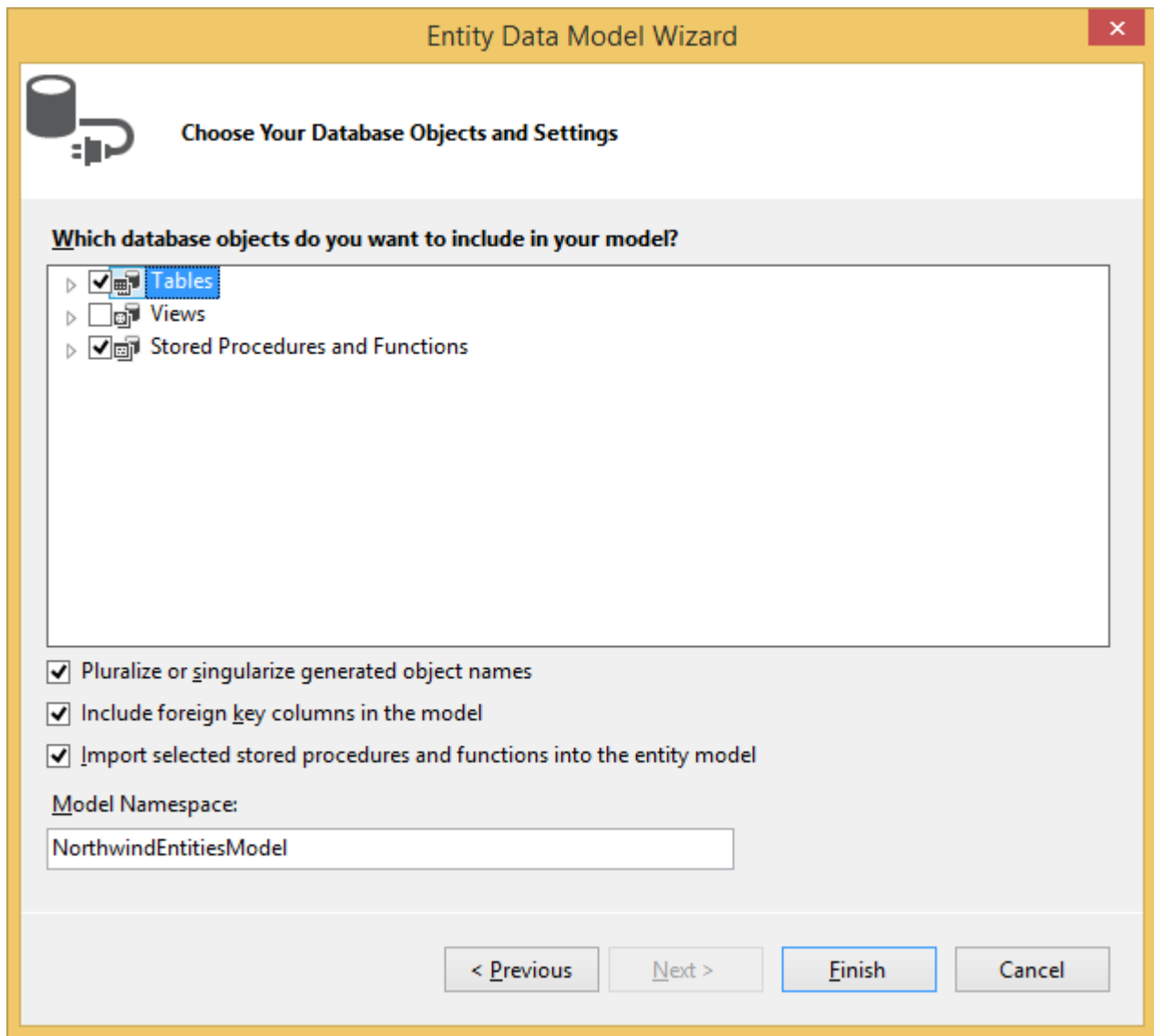
```
metadata=res://*/ModelFirst.Northwind.csdl|res://*/ModelFirst.Northwind.ssdl|
res://*/ModelFirst.Northwind.msl;provider=System.Data.SqlClient;provider connection string="data
source=(LocalDB)\v11.0;attachdbfilename=|DataDirectory|\Model\NorthwindEntities.mdf;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save connection settings in App.Config as:

NorthwindEntities

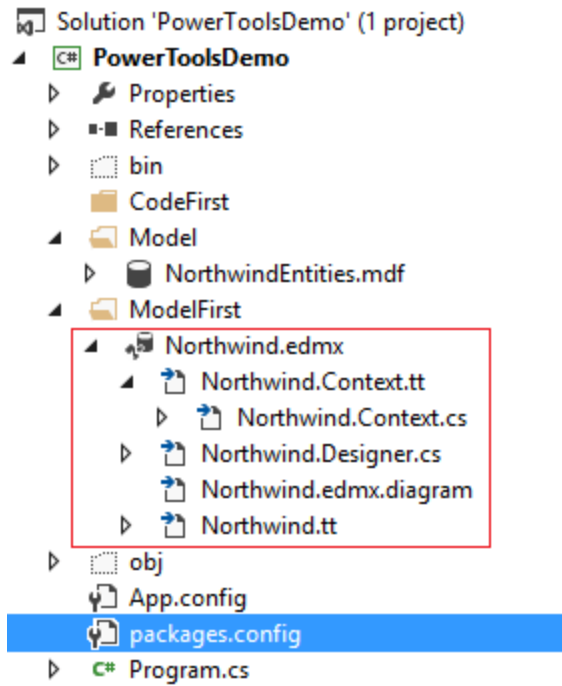
< Previous Next > Finish Cancel

Rename your connection string to whatever you like (I'm calling mine "Northwind Entities") and hit Next, which opens the Database Objects and Settings dialog.

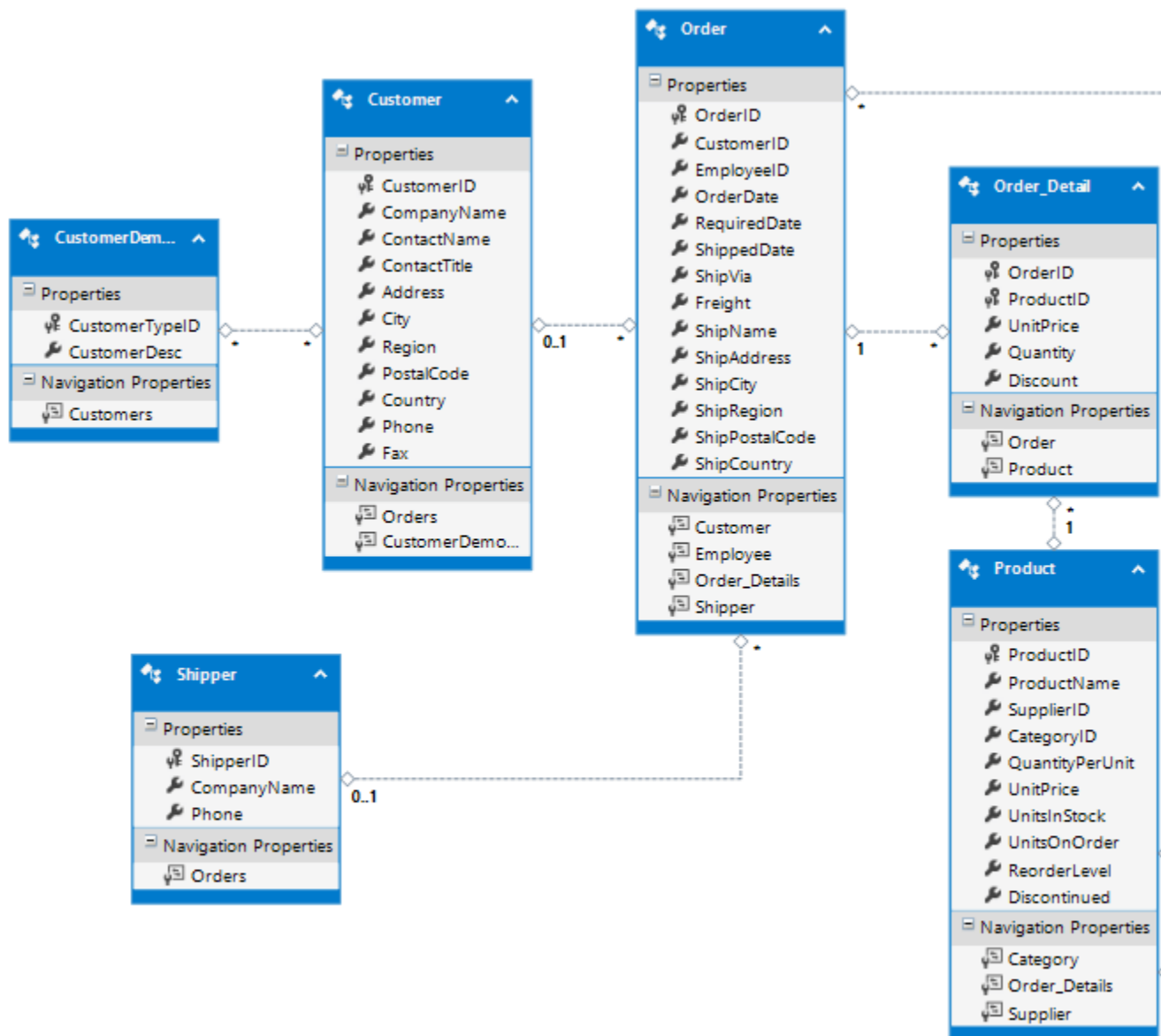


On this dialog you can select all items from the database that you want to be modeled. For this demo, we'll be using all Tables and Stored Procedures in the Northwind database.

Click *Finish* to generate your model. Once the code generation is complete, you'll see a new EDMX file in your project:



Open that file to see a database diagram. Mine looks like this:



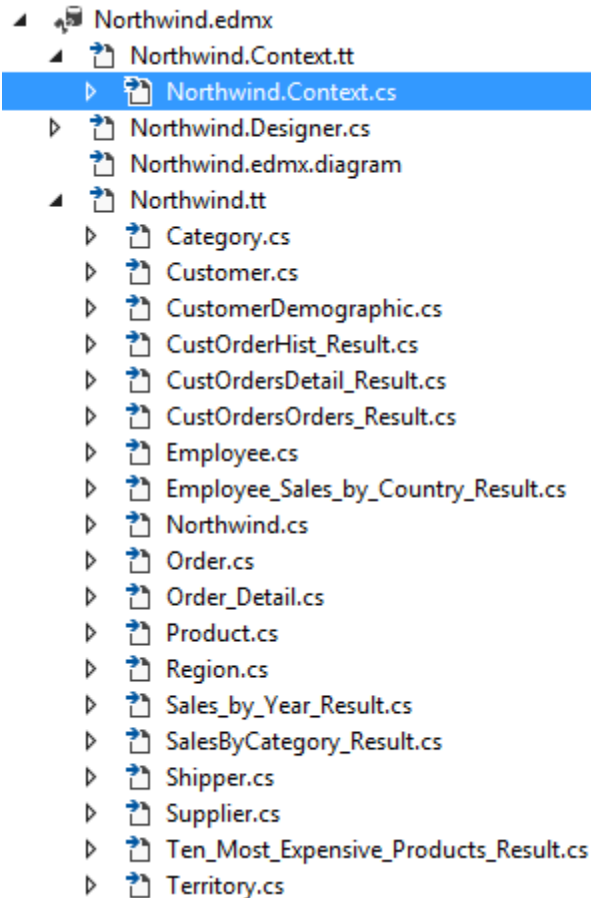
Well now, that looks like a well-formed database. But, what are those other files besides the EDMX?

Let's break them down.

- The Northwind.Context.tt file is a [T4 code-generation file](#) that is responsible for creating the class that contains the Context that we will be working with in code.

- The Northwind.Context.cs file is the class that was generated by the T4 template, and acts as the context class for this model.
- Northwind.Designer.cs and Northwind.edmx.diagram are files used in creating and changing the diagram we saw earlier.
- Northwind.tt is another T4 code generation template that is responsible for generating the classes for all objects modeled by the database.

The objects modeled by my Northwind.tt file look like this:



But what does one of those classes actually look like? Here's the Category class:

```
public partial class Category
{
    public Category()
    {
        this.Products = new HashSet<Product>();
    }
}
```



```

public int CategoryID { get; set; }
public string CategoryName { get; set; }
public string Description { get; set; }
public byte[] Picture { get; set; }

public virtual ICollection<Product> Products { get; set; }
}

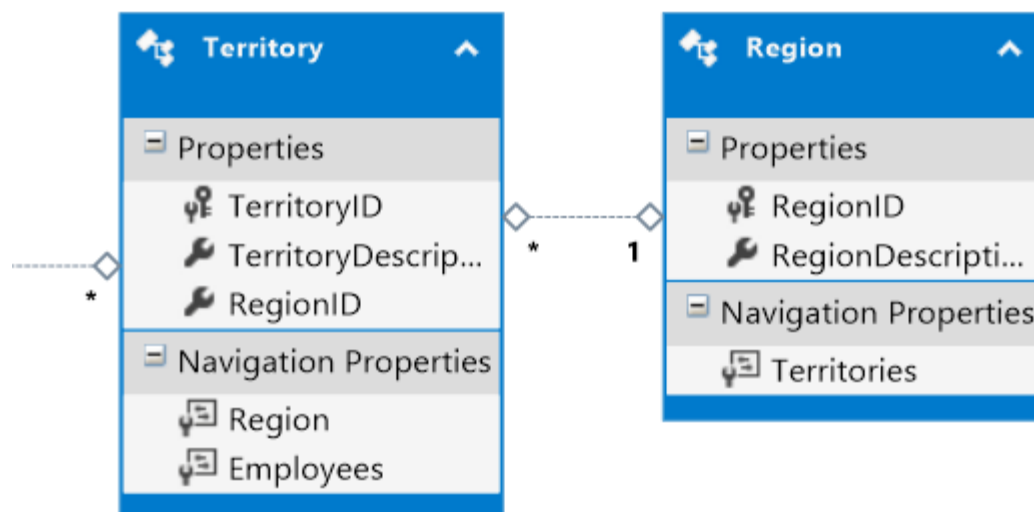
```

Well now, EF appears to have translated the SQL data types (e.g. varchar, varbinary, int) into C# datatypes (int, string, byte[]). Further, it appears to have created a virtual property for Products. But what does that mean?

## Types of Relationships

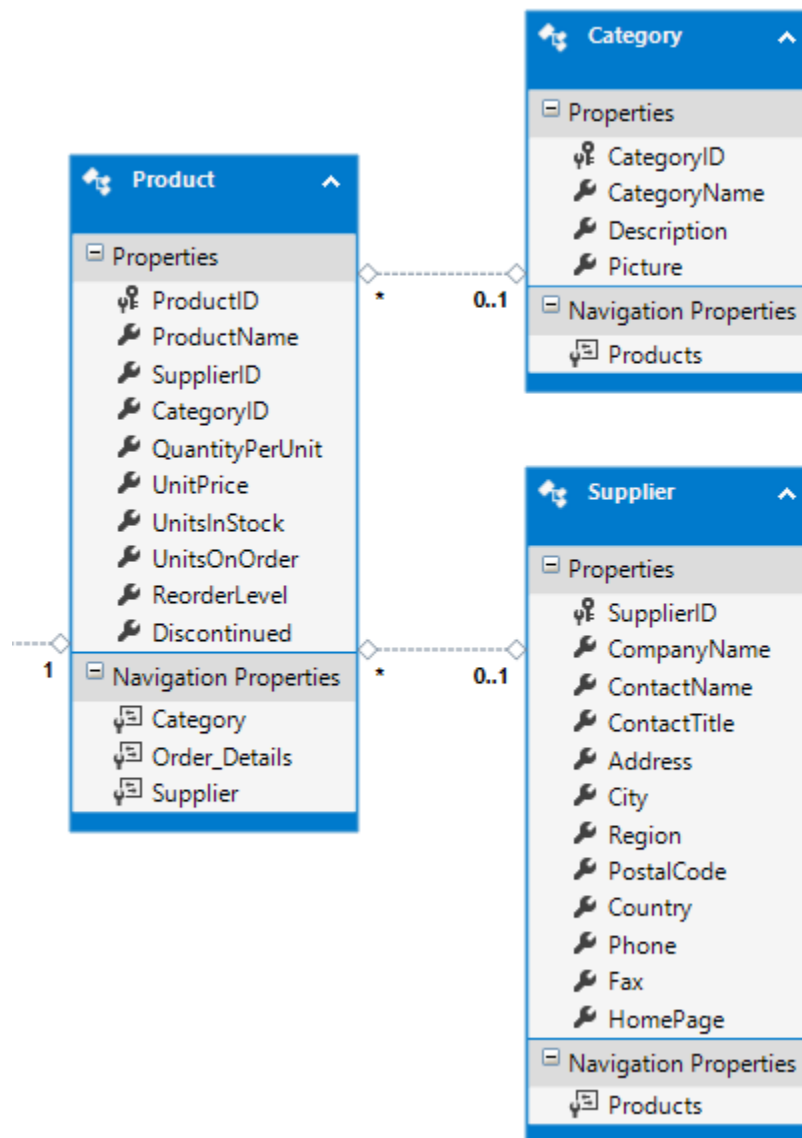
In relational databases, tables are related to each other via the use of keys. One table has a "foreign key" to another table, and the kind of relationship tells you something about how objects in each of those tables are related.

For example, in our Northwind database we have the tables **Region** and **Territory**. The Territory table contains a column called RegionID, which is a foreign key to the Region table. That relationship is shown in this snippet from the full Northwind diagram:



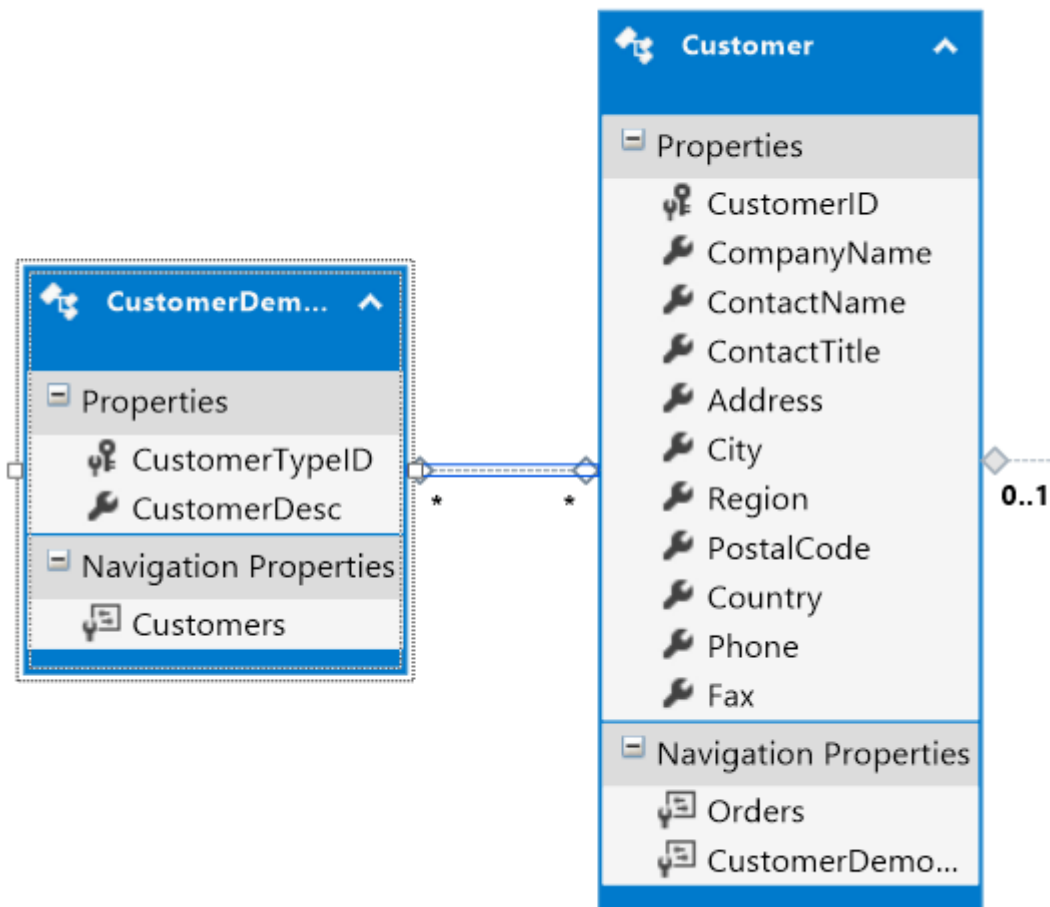
See the 1 and \* markers? Those signify the relationship between the two tables: for every 1 region, there can be many territories. This is called a **1-to-many relationship**.

We should also look at the relationship between **Category** and **Product**:



This relationship looks similar, but now we have 0..1 to \*. This is called a **zero-or-one-to-many relationship**, and is merely a construct of having the foreign key column in the related table be nullable. In other words, a Product *can* have a related Category, but it doesn't *have* to, and any given Category may have multiple Products associated to it.

Finally let's take a look at the Customer and CustomerDemographic tables. Here's another snippet from the main diagram:



Now there's an \* on each side of the relationship. This is called a **many-to-many relationship**. It means that one Customer could have many CustomerDemographic entries, and one CustomerDemographic could have many Customers.

In a database, a many-to-many relationship [requires the existence of another table](#), called a mapping table or relationship table, that shows which items from A (the left side of the relationship) are related to which items from B (the right side). But Entity Framework is smart, and it realizes that this mapping table is actually not an entity unto itself, but instead is simply a relationship between two existing entities.

Now we've created our database-first model using Entity Framework! Despite all of my long-winded blabbering, all it really takes is connecting to the database and walking through the wizard and BOOM, you've got a model and a context so you can get started writing code.