Universidad Tecnológica Nacional. Buenos Aires.
Ingeniería de Sistemas
Diseño de Sistemas.

ORM's - ENTITY FRAMEWORK

Lic. Adriana Maulini Buño

Entity Framework

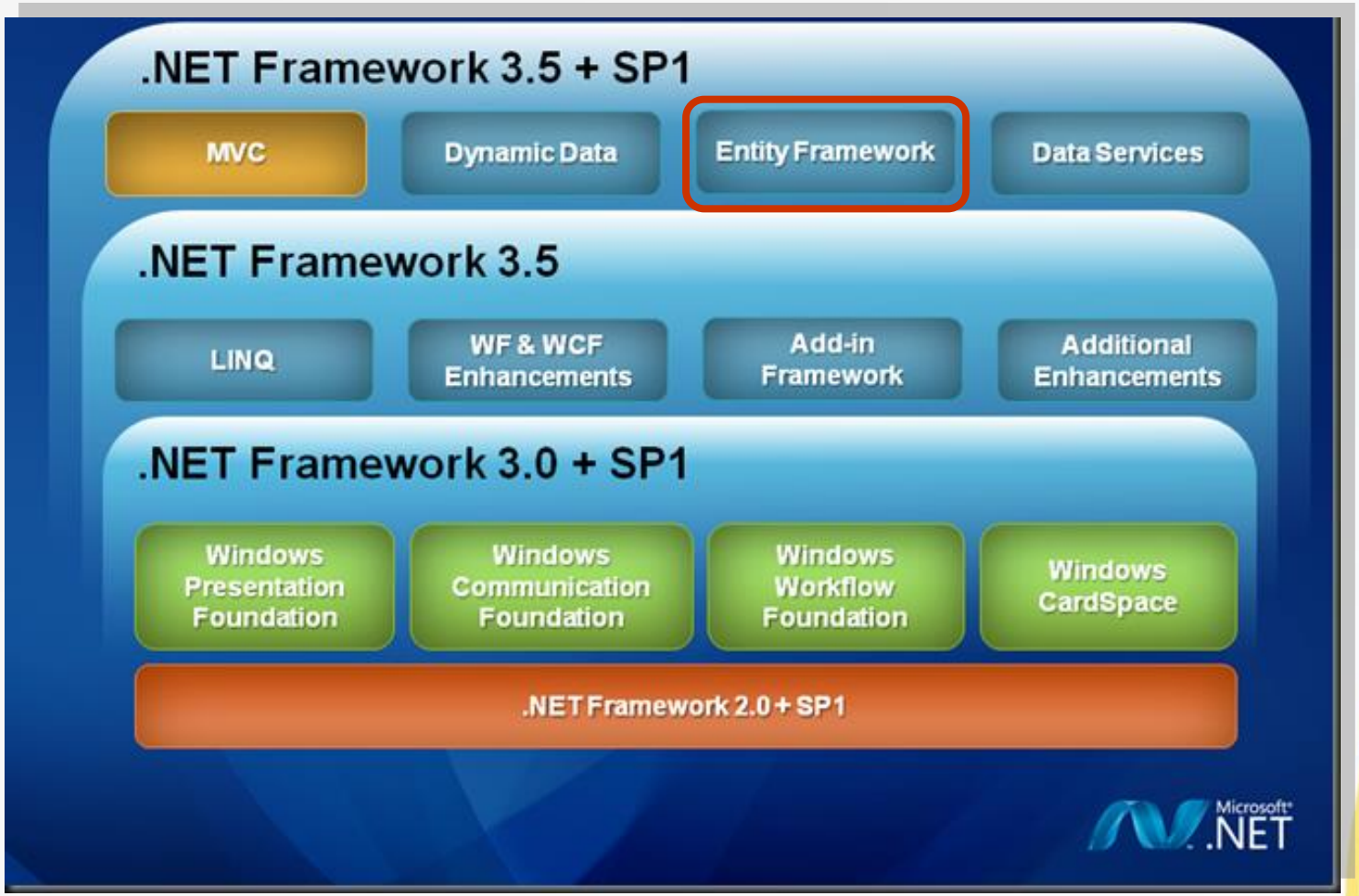
- Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos.
- Es un ORM (Object-relational mapping).
- ADO.NET son un conjunto de componentes de software que permiten la interacción con bases de datos, a través de tecnologías .NET

Entity Framework (ORM)

Menos código, mayor facilidad para interactuar con una B.D. Más tiempo para dedicarle a otras tareas.

Requisitos

- .NET framework 3.5 SP1 o superior, herramientas de desarrollo de microsoft. (La versión del VS depende de la version de EF, ejemplo: Para EF 6, necesitas VS 2012 o superior)



Novedades Entity Framework 6

- Código Abierto (Aunque el dueño sigue siendo Microsoft)

  http://entityframework.codeplex.com/
  https://git01.codeplex.com/entityframework

- Soporte de Async tanto para hacer consultar como guardar datos, se añade soporte para los patrones asincrónicos basados en tareas (task), que se introdujeron en el .NET 4.5

- Configuración por codigo, que generalmente esto se realiza por un archivo de configuración, o por código. Este se le conoce como configuración basada en código.

- Mejoras en el rendimiento en consultas Enumerable.Contains con LINQ.

- Mejoras en el tiempo de arranque, especialmente para grandes modelos.

https://csharpcomopuntodepartida.wordpress.com/2013/10/23/novedades-de-entity-framework-6/

```
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        string connectionString =
            "Data Source=(local);Initial Catalog=Northwind;"
            + "Integrated Security=true";

        // Provide the query string with a parameter placeholder.
        string queryString =
            "SELECT ProductID, UnitPrice, ProductName from dbo.products "
                + "WHERE UnitPrice > @pricePoint "
                + "ORDER BY UnitPrice DESC;";

        // Specify the parameter value.
        int paramValue = 5;

        // Create and open the connection in a using block. This
        // ensures that all resources will be closed and disposed
        // when the code exits.
        using (SqlConnection connection =
            new SqlConnection(connectionString))
        {
            // Create the Command and Parameter objects.
            SqlCommand command = new SqlCommand(queryString, connection);
            command.Parameters.AddWithValue("@pricePoint", paramValue);

            // Open the connection in a try/catch block.
            // Create and execute the DataReader, writing the result
            // set to the console window.
            try
            {
                connection.Open();
                SqlDataReader reader = command.ExecuteReader();
                while (reader.Read())
                {
                    Console.WriteLine("\t{0}\t{1}\t{2}",
                        reader[0], reader[1], reader[2]);
                }
                reader.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            Console.ReadLine();
        }
    }
```

¿Cómo se manejaban las BD antes de E.F. u otro ORM?

ADO.NET

https://msdn.microsoft.com/en-us/library/dw70f090(v=vs.110).aspx

¿Cómo se manejaban las BD antes de E.F. u otro ORM?

Ej: Insert

LINQ TO SQL

C# | VB

```
// Northwnd inherits from System.Data.Linq.DataContext.
Northwnd nw = new Northwnd(@"northwnd.mdf");

Customer cust = new Customer();
cust.CompanyName = "SomeCompany";
cust.City = "London";
cust.CustomerID = "98128";
cust.PostalCode = "55555";
cust.Phone = "555-555-5555";
nw.Customers.InsertOnSubmit(cust);

// At this point, the new Customer object is added in the object model.
// In LINQ to SQL, the change is not sent to the database until
// SubmitChanges is called.
nw.SubmitChanges();
```

https://msdn.microsoft.com/en-us/library/dw70f090(v=vs.110).aspx

E.F. WorkFlows

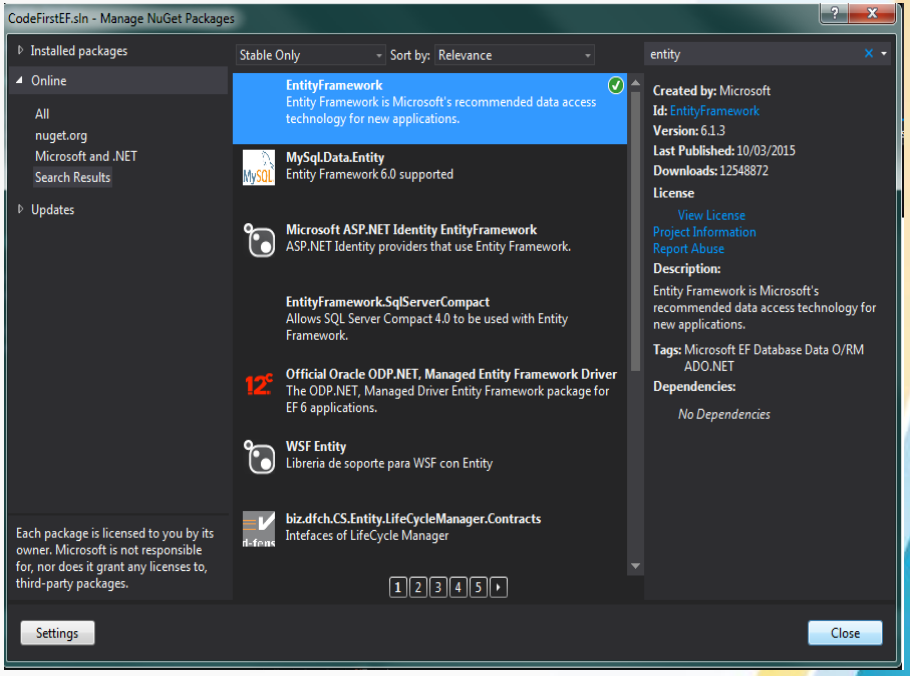
- Code First
- Model First
- DataBase First

E.F. Code First

1 - Definir clases.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EFCodeFirstBanco
{
    public class Cliente
    {
        public int ClienteId { get; set; }
        public string Nombre { get; set; }
        public string Direccion { get; set; }

        public ICollection<Cuenta> Cuentas { get; set; }

    }
}
```

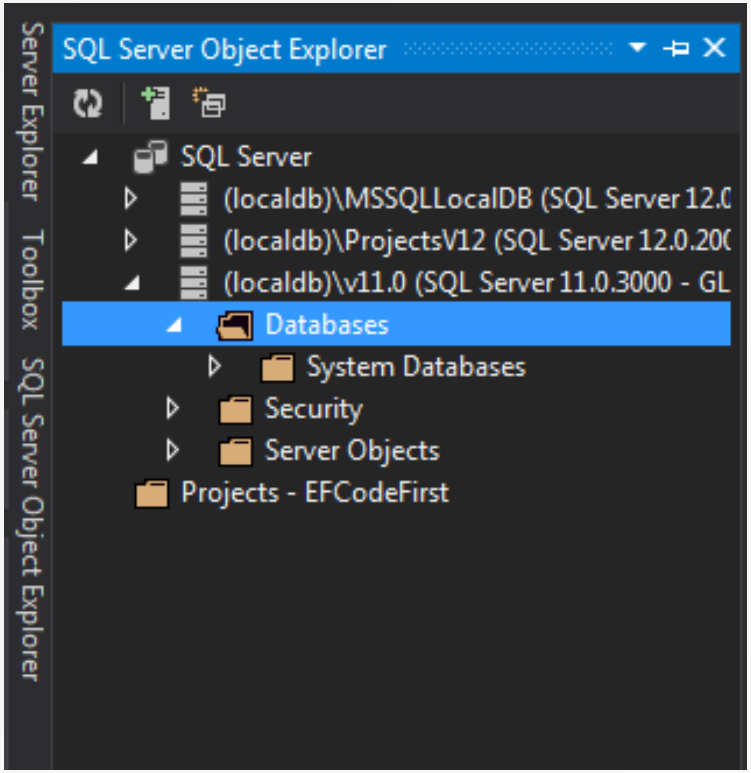
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EFCodeFirstBanco
{
    class Cuenta
    {
        public int CuentaId { get; set; }
        public string CBU { get; set; }
        public double Saldo { get; set; }

        public int ClienteID { get; set; }

        [ForeignKey("ClienteID")]
        public Cliente Cliente { get; set; }
    }
}
```



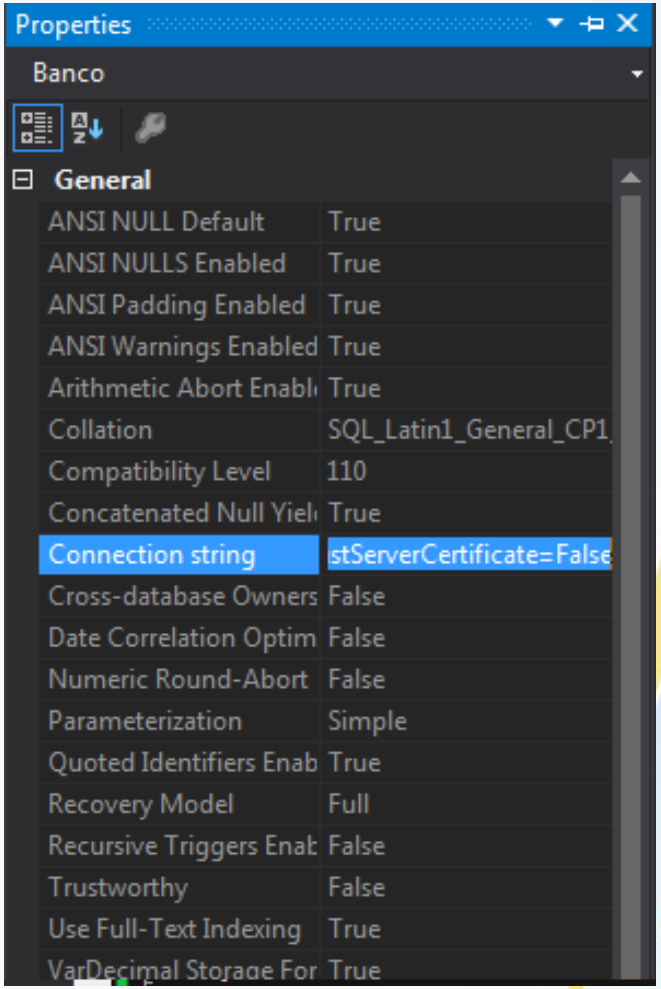
2 - Instalar Entity Framework en el Manejador de paquetes de Nuget.

E.F. Code First

3 - Crear base de datos local en el SQL Server Object Explorer.



3 - En las propiedades de la base de datos, buscar el ConnectionString y copiarlo.



E.F. Code First

3 - Referenciar System.Data.Entity y luego crear clase de contexto con el ConnectionString copiado.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace EFCodeFirstBanco
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }

    public class DBContext : DbContext
    {

        public DbSet<Cliente> Clientes { get; set; }
        public DbSet<Cuenta> Cuentas { get; set; }

        public DBContext(): base(@"Data Source=(localdb)\Pr
        {

        }

    }
}
```

E.F. Code First

4- Colocar código en el main de implementación, compilar y ejecutar.

```
file:///C:/Users/adriana.maulini/Desktop/EFCodeFirstBanco/EFCodeFirstBa
Ingrese nombre del cliente:Adriana Maulini
Ingrese direccion del cliente: palermo
Ingrese CBU de la cuenta:55669988555552
Ingrese saldo de la cuenta: 1000000000000000
Todas las cuentas en la BD:
55669988555552
Adriana MauliniPress any key to exit...
```

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new DataBaseContext())
        {
            // Crear nuevo cliente
            Console.Write("Ingrese nombre del cliente:");
            var nombre = Console.ReadLine();

            Console.Write("Ingrese direccion del cliente: ");
            var direccion = Console.ReadLine();

            var cliente = new Cliente { Nombre = nombre, Direccion = direccion};


            // Crear nueva cuenta
            Console.Write("Ingrese CBU de la cuenta:");
            var cbu = Console.ReadLine();

            Console.Write("Ingrese saldo de la cuenta: ");
            var saldo = Console.ReadLine();

            var cuenta = new Cuenta{ CBU= cbu, Saldo = Double.Parse(saldo), Cliente= client

            cliente.CuentasCliente = new Collection<Cuenta>();
            cliente.CuentasCliente.Add(cuenta);
            db.Clientes.Add(cliente);
            db.Cuentas.Add(cuenta);

            db.SaveChanges();

            // Mostrar cuentas en la BD
            var query = from b in db.Cuentas
                        orderby b.CBU
                        select b;

            Console.WriteLine("Todas las cuentas en la BD:");
            foreach (var item in query)
            {
                Console.WriteLine(item.CBU);
                Console.Write(item.Cliente.Nombre);
```

E.F. Code First

Entonces, para implementar sería (por ejemplo en el caso del cliente):

- using (var db = new BloggingContext())
- Crear Objeto cliente.
- db.Clientes.Add(cliente)
- db.SaveChanges();

¿Y para eliminar o hacer update a la data?

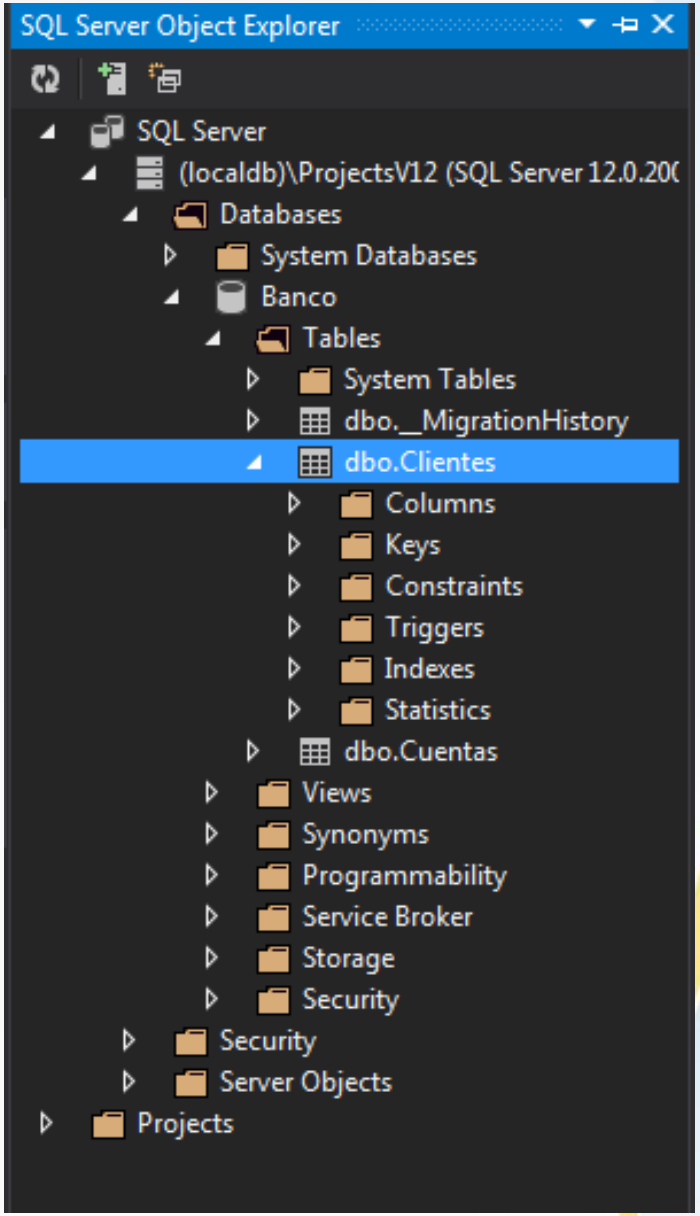
Trabajar con los métodos propios de la clase collection. (Add, Remove, etc) sobre las colecciones o listas creadas para cada tabla, y al final llamar al método SaveChanges() como en el ejemplo anterior.

E.F. Code First

Y ¿Dónde esá mi DATA?

En la base de datos local creada, lo podemos ver a través del SQL Server Object Explorer

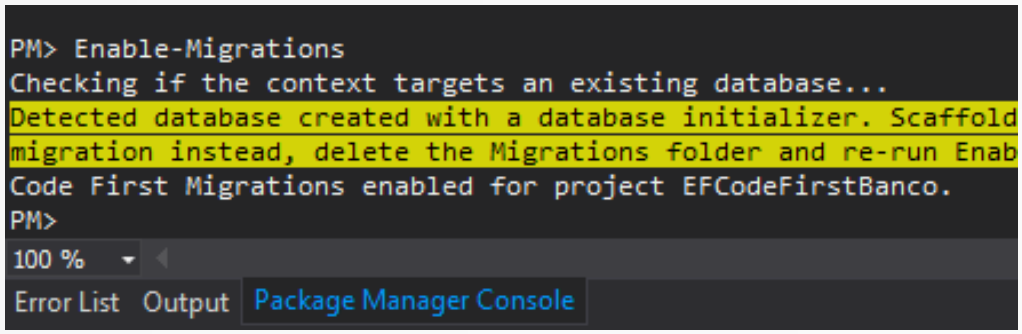
	ClienteId	Nombre	Direccion
▶	1	Adriana Maulini	palermo
	2	Juan	almagro
	3	maria	lugano
	4	lola	lugano
	5	asfgsdfg	sdfgdfgadfg
	6	dfjhdhjdghj	nuñez
*	*NULL*	*NULL*	*NULL*

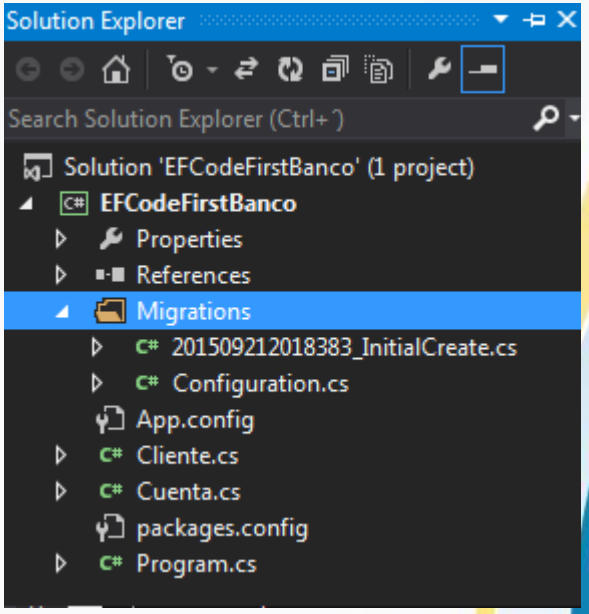


E.F. Code First

Y ¿Qué debo hacer si quiero cambiar algo en la estructura de la BD?

1- Los cambios en la estructura de la BD, se manejan mediante "Migraciones", para ello hay que habilitarlas mediante el comando "Enable-Migrations" en la consola de paquetes del VS. Esto creará una carpeta dentro del proyecto.





E.F. Code First

Y ¿Qué debo hacer si quiero cambiar algo en la estructura de la BD?

En la carpeta nueva se creó una migración inicial, en ella podemos ver cómo se forma una migración. Son básicamente clases con un método UP, y un método Down, que se ejecutarán para subir cambios o revertirlos respectivamente.

```
public partial class InitialCreate : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Clientes",
            c => new
                {
                    ClienteId = c.Int(nullable: false, identity: true),
                    Nombre = c.String(),
                    Direccion = c.String(),
                })
            .PrimaryKey(t => t.ClienteId);

        CreateTable(
            "dbo.Cuentas",
            c => new
                {
                    CuentaId = c.Int(nullable: false, identity: true),
                    CBU = c.String(),
                    Saldo = c.Double(nullable: false),
                    ClienteID = c.Int(nullable: false),
                })
            .PrimaryKey(t => t.CuentaId)
            .ForeignKey("dbo.Clientes", t => t.ClienteID, cascadeDelete: true)
            .Index(t => t.ClienteID);

    }

    public override void Down()
    {
        DropForeignKey("dbo.Cuentas", "ClienteID", "dbo.Clientes");
        DropIndex("dbo.Cuentas", new[] { "ClienteID" });
        DropTable("dbo.Cuentas");
        DropTable("dbo.Clientes");
    }
```

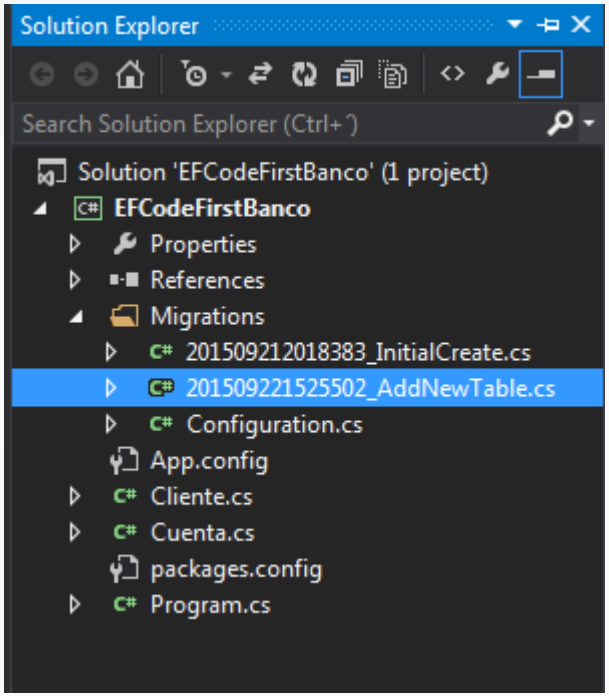
E.F. Code First

Y ¿Qué debo hacer si quiero cambiar algo en la estructura de la BD?

2- Para hacer cualquier cambio, ahora hay que ejecutar el siguiente comando: Add-Migration NOMBRE –IgnoreChanges.

```
namespace EFCodeFirstBanco.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddNewTable : DbMigration
    {
        public override void Up()
        {
        }

        public override void Down()
        {
        }
    }
}
```

```
Package Manager Console
Package source: nuget.org   Default pr
PM> Add-Migration AddNewTable -IgnoreChanges
Scaffolding migration 'AddNewTable'.
The Designer Code for this migration file includes a
scaffold the next migration. If you make additional
 AddNewTable' again.
PM>
```

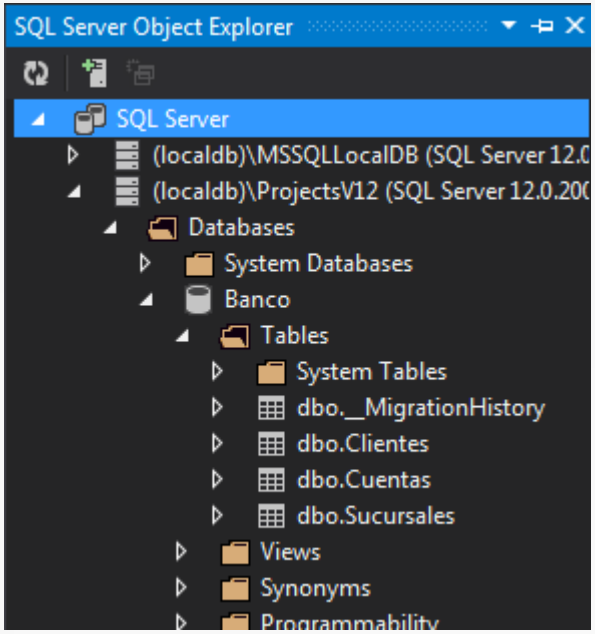


Se crea automáticamente una clase para la nueva migración.

E.F. Code First

Y ¿Qué debo hacer si quiero cambiar algo en la estructura de la BD?

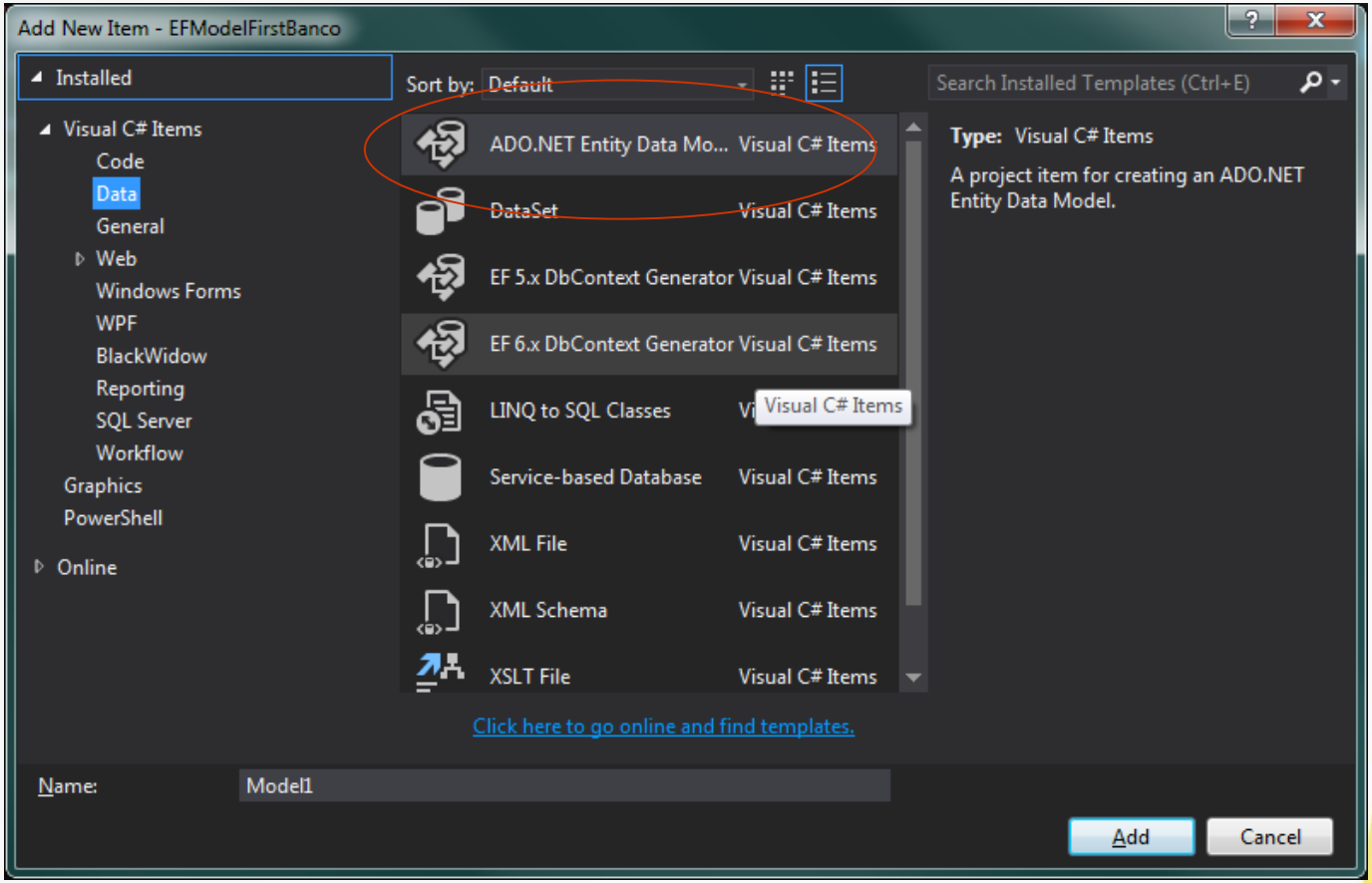
3- Ahora hay que poner el codigo necesario en la clase creada y luego ejecutar el comando "Update-Database". No hay que olvidar agregar las clases al modelo, y modificar la clase de contexto.




```
EFCodeFirstBanco
namespace EFCodeFirstBanco.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddNewTable : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                  "dbo.Sucursales",
                  c => new
                  {
                      SucursalId = c.Int(nullable: false, identity: tr
                      Nombre = c.String(),
                      Direccion = c.String(),
                  })
                  .PrimaryKey(t => t.SucursalId);
        }

        public override void Down()
        {

            DropIndex("dbo.Sucursales", new[] { "SucursalID" });
            DropTable("dbo.Sucursales");
        }
    }
}
Package source: nuget.org    Default project: EFCodeFirstBanco
argetMigration 201509212018383_InitialCreate', then delete '201509221547
M> update-database
pecify the '-Verbose' flag to view the SQL statements being applied to th
pplying explicit migrations: [201509221547045_AddNewTable].
pplying explicit migration: 201509221547045_AddNewTable.
unning Seed method.
M>
```

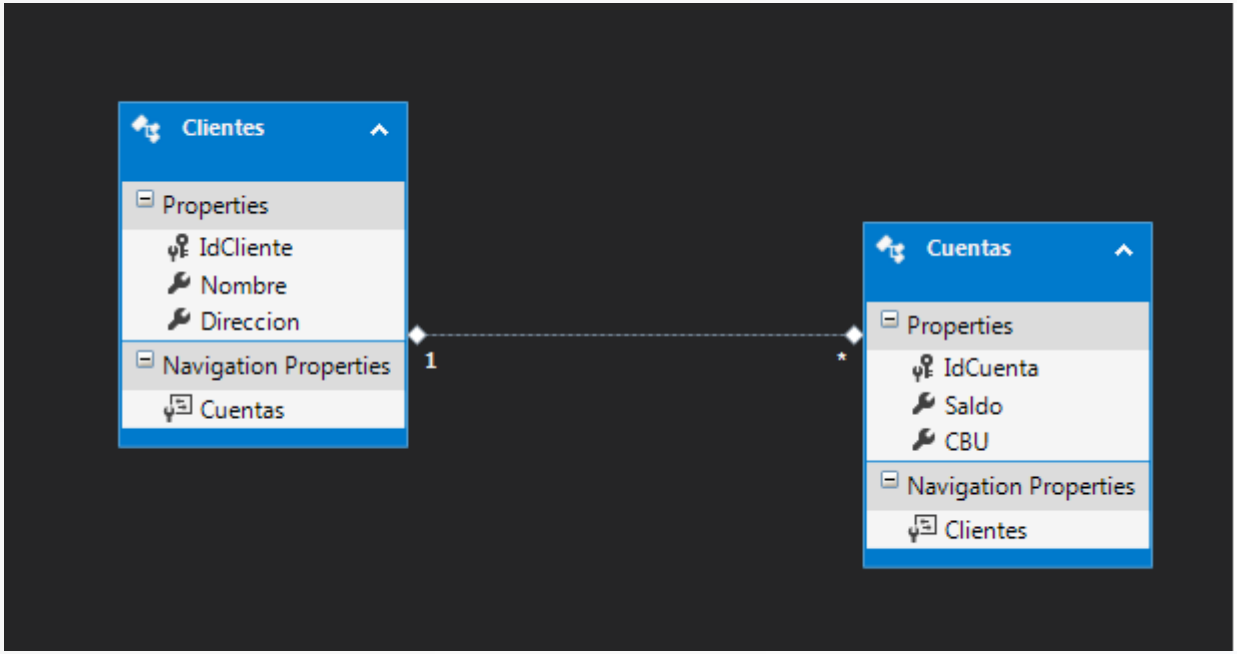
E.F. Model First

1- Agregar al proyecto un elemento nuevo ADO.NET Entity Data Model.



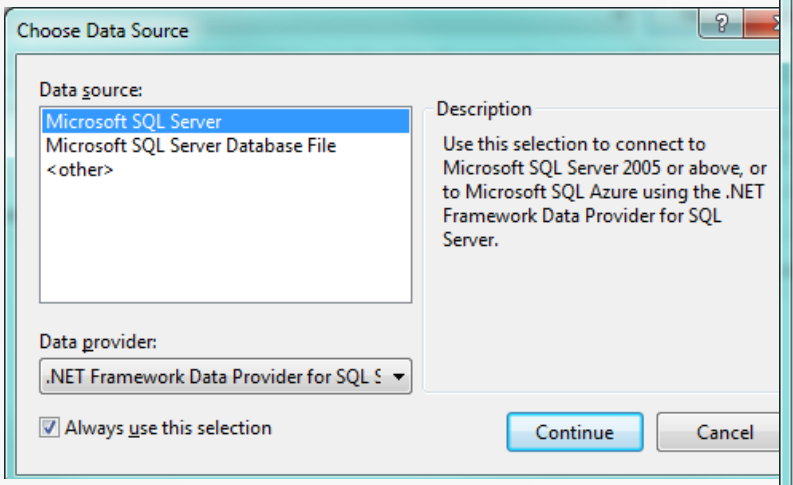
E.F. Model First

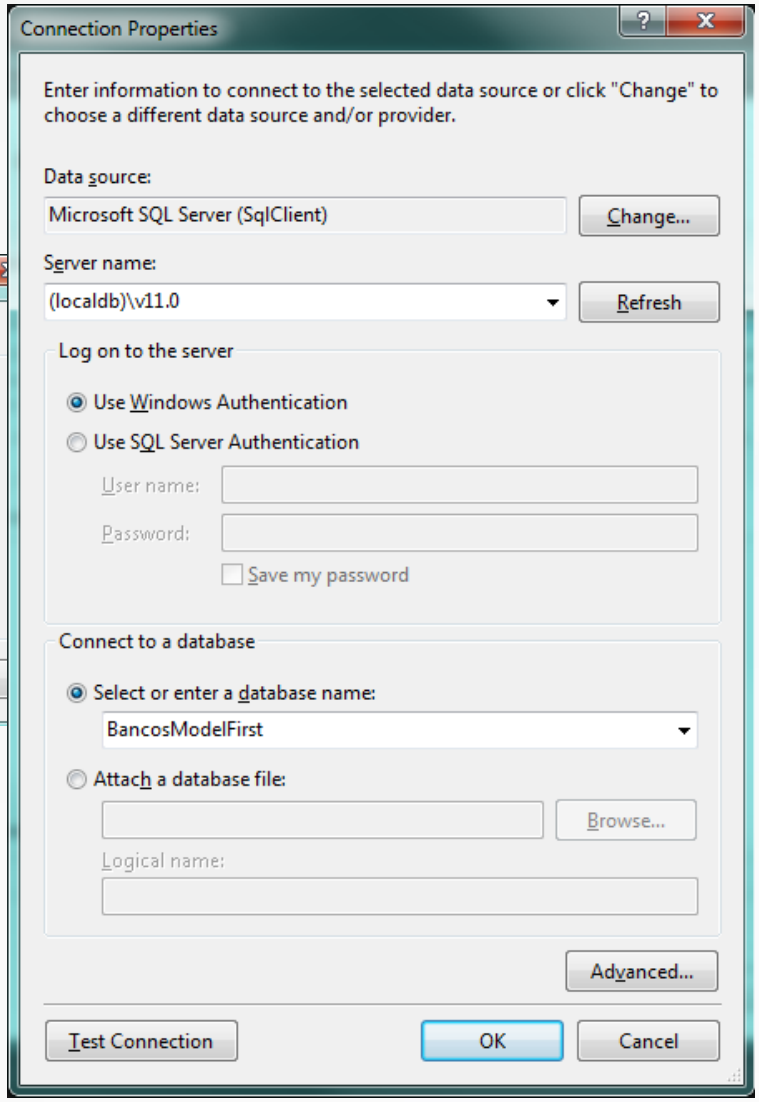
1- Seleccionar **Empty EF Designer model** en le Wizard.
2- Crear modelo utilizando las herramientas del ToolBox



E.F. Model First

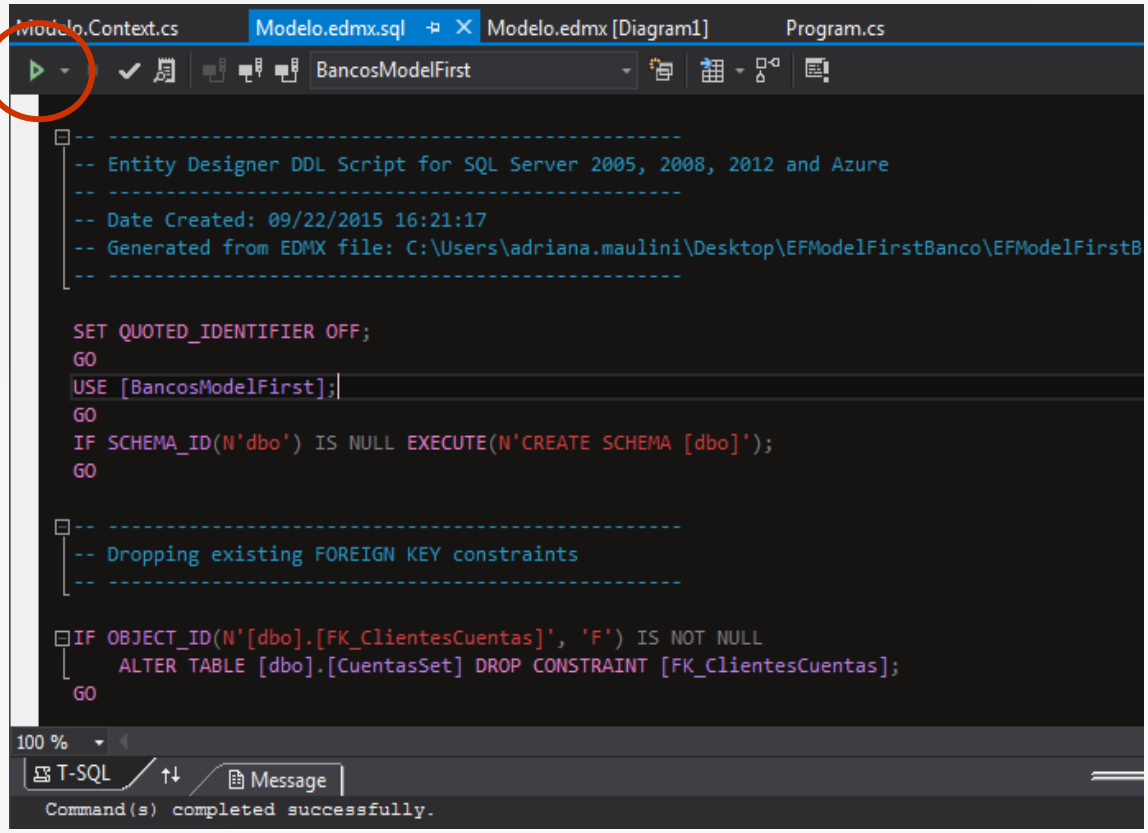
3- Botón derecho, "Generar nueva base de datos con este modelo"
4- Seguir los pasos del Wizard para establecer la conexión.





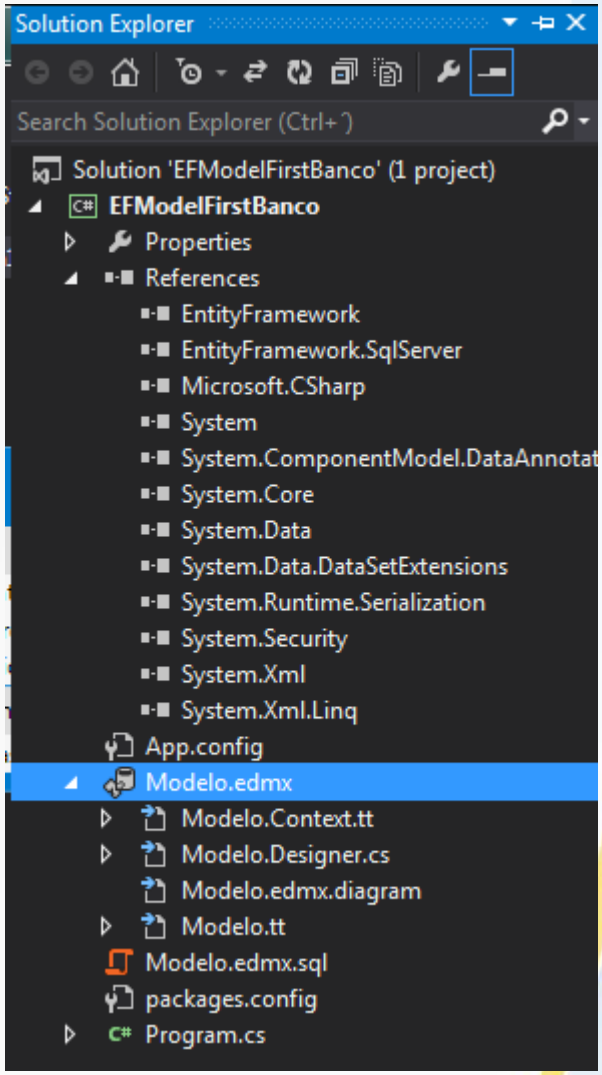
E.F. Model First

5- Abrir el archivo con extensión edmx.sql
y ejecutar el query generado.



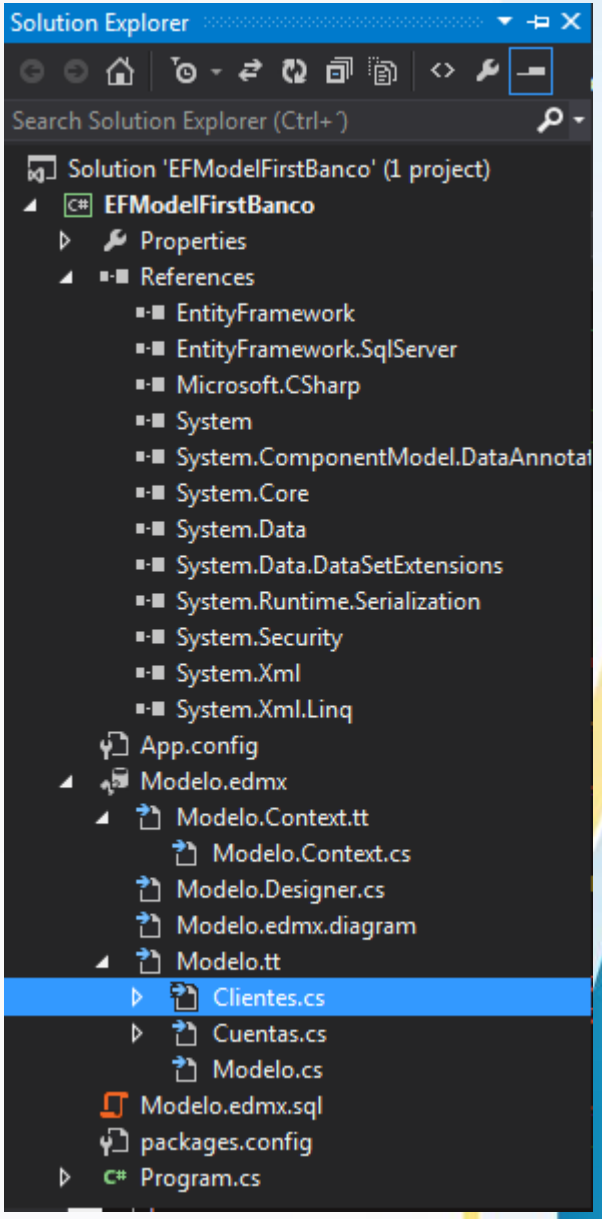
E.F. Model First

Se creó en el proyecto un archivo con extensión .edmx, ahi se debe modificar el modelo cuando sea necesario. Cada vez que se modifique hay que volver a seguir los pasos y ejecutar el query.



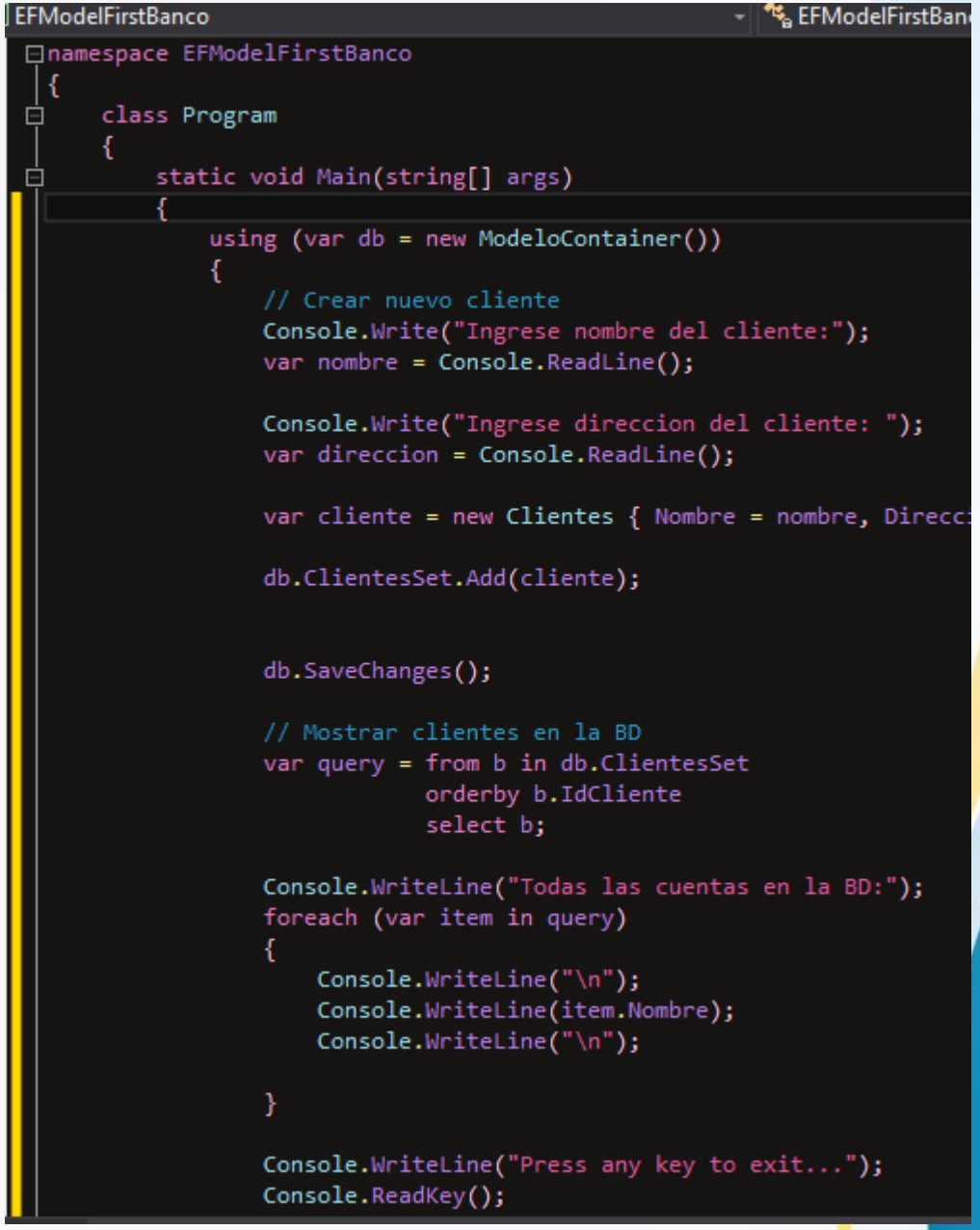
E.F. Model First

En el solution explorer, se puede ver como se crearon las clases del modelo para cada entidad.



E.F. Model First

5- Se implementa de igual forma que el "Code First". (Generando un objeto de la clase de contexto, objetos de las clases del modelo y ejecutando los métodos mencionados para hacer cambios.



E.F. DataBase First

Son los mismos pasos que Model First, con la excepción de que al agregar el elemento ADO.NET Entity Data Model. al proyecto, se selecciona "EF Designer from Database", y se selecciona una base de datos ya existente, así como los componentes de la misma que se quieren importar. Esto creará un modelo con las clases necesarias y la clase de contexto ya lista (no hace falta ejecutar el query).