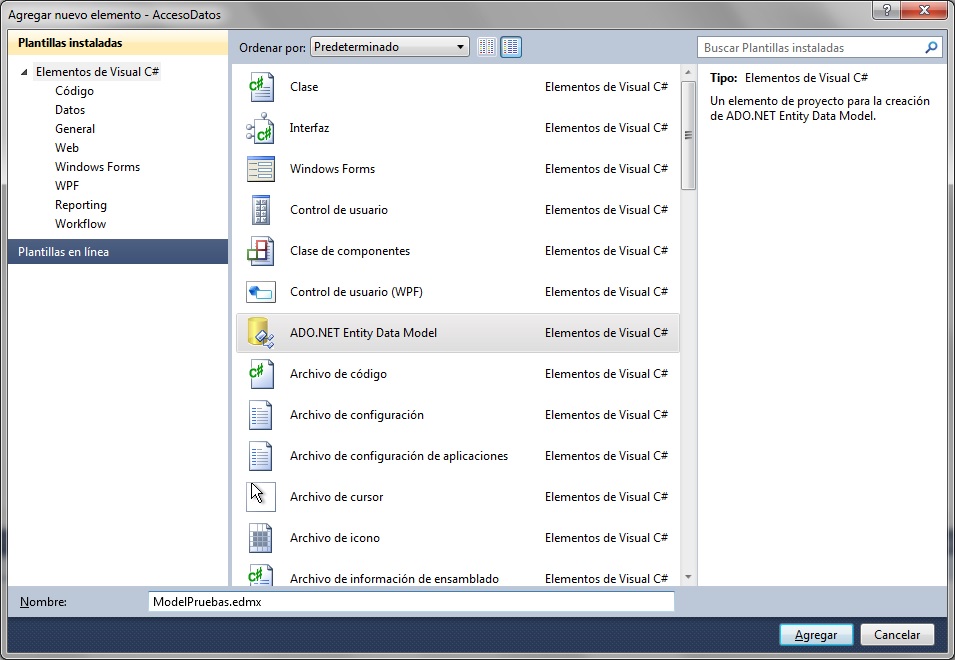
**¿Que es Entity Framework?**

Es el conjunto de herramientas y apis proporcionadas por Microsoft para el acceso a datos a través de un modelo basado en entidades, abstrayendo al programador de la Base de datos. En pocas palabras, es el [**ORM**](http://www.tuprogramacion.com/glosario/que-es-un-orm/) de Microsoft para .NET.

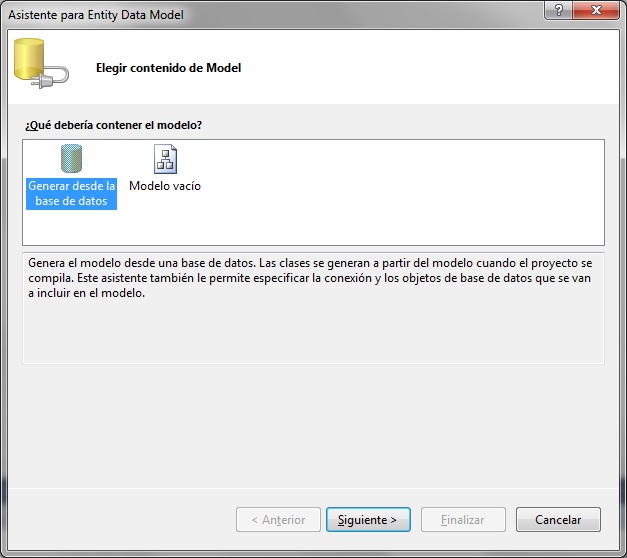
**Configuración del acceso a datos con Entity Framework**

Lo primero como siempre creamos una nuevo proyecto, ya sea web, móvil, o un simple programa de consola para poder empezar. Una vez creado, agregamos una biblioteca de clases donde colocar nuestra capa de acceso a datos. Como veis, esto es exactamente igual que si no usáramos EF.

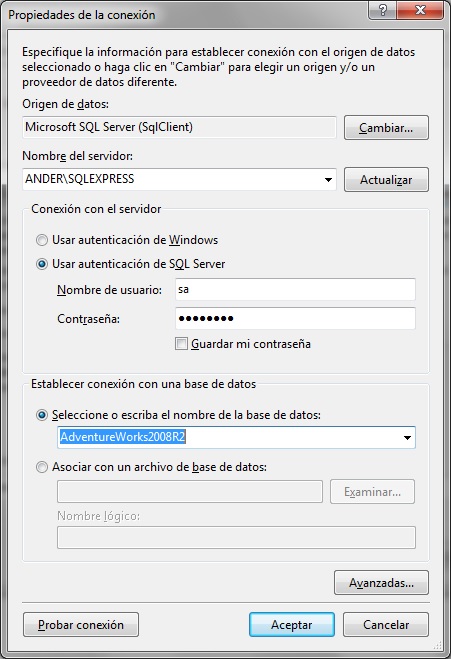
Ahora vamos a añadir nuestro contexto de conexión, para ello, creamos un nuevo archivo dentro de la capa de acceso a datos de **tipo .edmx**



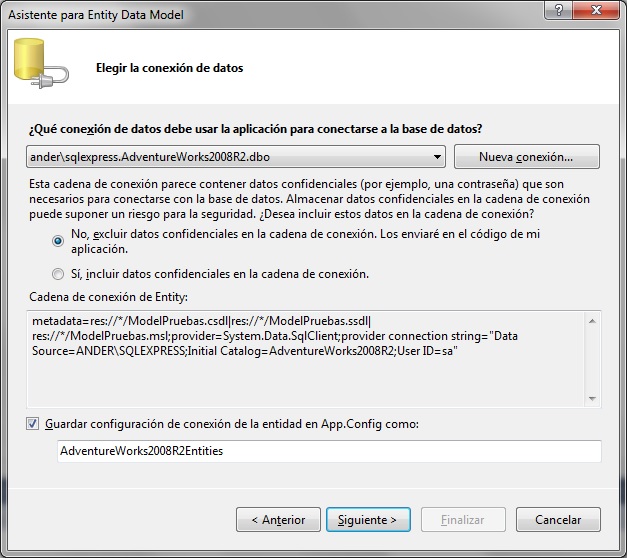
Después de asignarle un nombre, nos saldrá una ventana donde nos dará a elegir si usamos como modelo una **Base de datos ya existente (Database First) o en cambio creamos un modelo de cero (Model First)**. En Entiy Framework 4.1 surgió una nueva forma de trabajar Code First, que nos permite codificar primero el modelo. En este ejemplo para simplificar las cosas vamos a usar un BD existente:



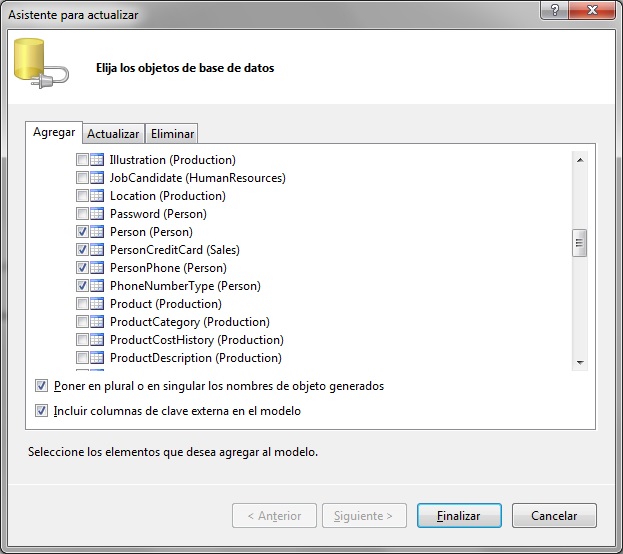
Tras esto, establecemos una conexión con la BD de datos, elegimos con la que vamos a trabajar, en nuestro caso usaremos la BD de ejemplo del SQL Server 2008 R2 "AdventureWorks"



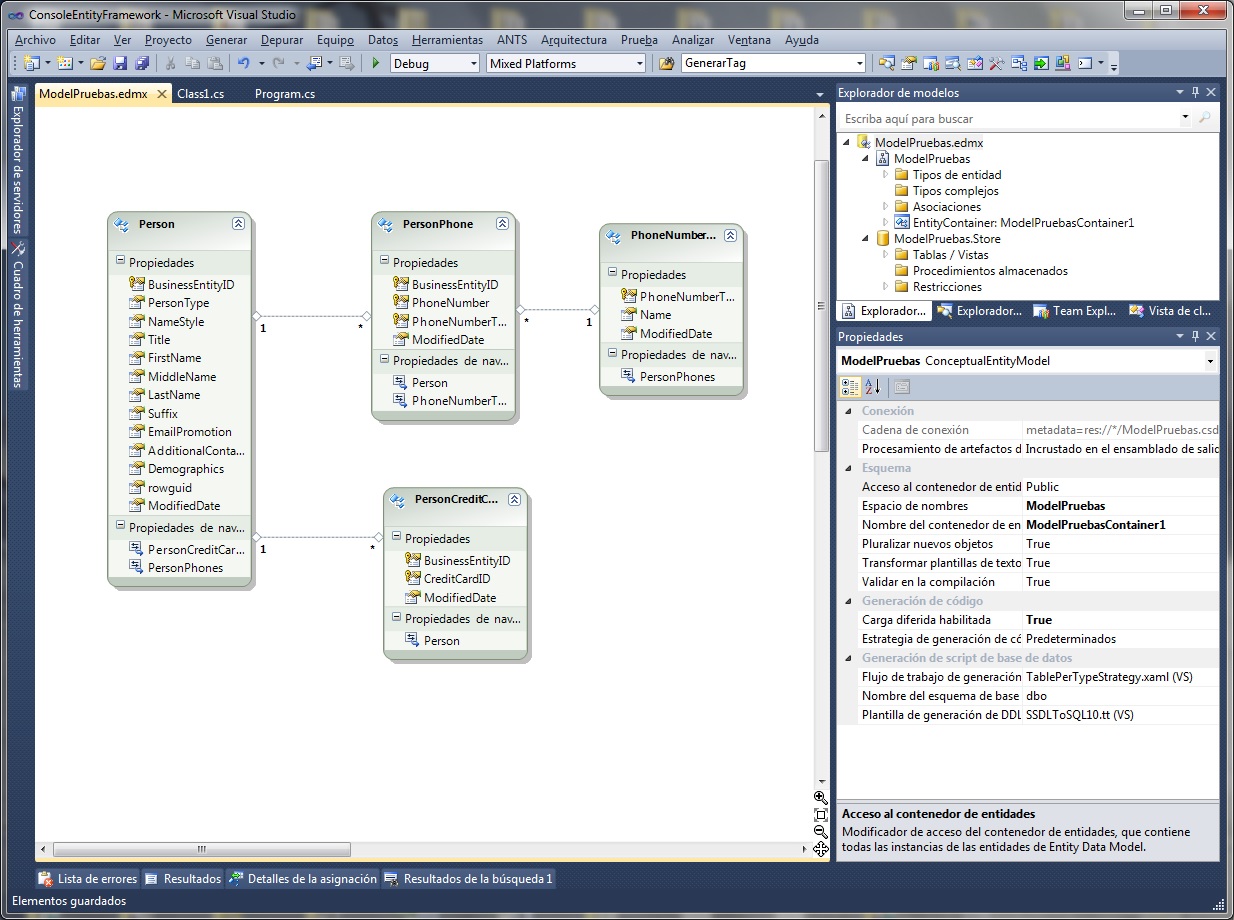
La cadena de conexión se creará automáticamente, solo tendremos que establecer un nombre y si se guardará la contraseña en el archivo de configuración o si la pasamos por código



Finalmente, deberemos elegir qué tablas, vistas, procedimientos... de la base de datos vamos a utilizar para este contexto de conexión. Para este ejemplo voy a seleccionar solamente 4 tablas:

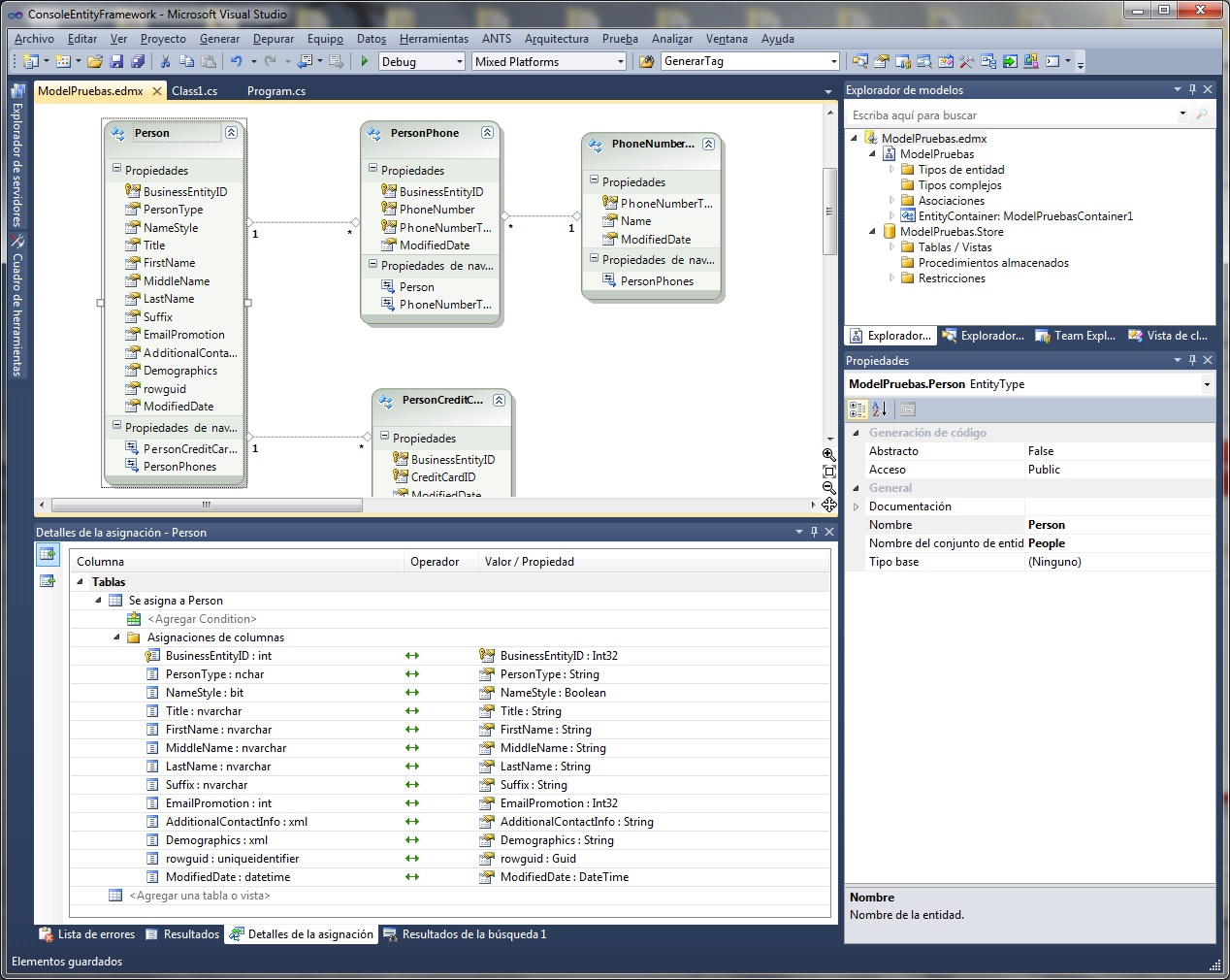


Con esto tendremos creadas nuestras entidades y relaciones. Ya tenemos todo preparado para usarse.



Si vemos las propiedades de una de las entidades, podemos ver como **Entity Framework ha asociado cada uno de los campos de las tablas de las bases de datos, a un atributo de las entidades generadas.**

Como dato curioso, podemos ver la singularización del paso de tabla a entidad, donde la tabla "people" se ha transformado en la entidad person. Esto funcionara bastante bien en ingles, pero si denominamos a nuestras tablas con nombre en castellano, como mucho lograremos que quite la ultima s.



**Uso de Entity Framework**

Ahora que tenemos creado el acceso a datos, deberemos incluir la referencia en el proyecto donde vamos a usar estos datos. Por simplificar, voy a hacerlo directamente desde un pequeño programa de consola.

Importante, crear un archivo de configuración y añadir la cadena de conexión ya que entity framework la tomará de aquí en vez del proyecto donde se creo.

Una vez configurado, podemos hacer uso de él de una manera bastante sencilla. Para ello creamos un contexto de conexión, y si queremos podemos sacar directamente un listado de los usuarios que cumplan una condición:

ContextoEntidades con = new ContextoEntidades();

List<Person> lista\_personas = con.People.Where(c=>c.cond==true).ToList();

Con esta pequeña sentencia habremos seleccionado y transformado en objetos un conjunto de datos de la base de datos.

Como conclusión, podemos observar con este último ejemplo, que hemos accedido a la BD sin necesidad de usar sql, ni realizar transformaciones posteriores para encapsular los registros en objetos. Con esto, las posibilidades son infinitas, ya que al abstraer al programador de la base de datos, podemos estar seguros de trabajar con una tecnología que aunque haya cambios de base de datos, va a seguir funcionando. Además de la rapidez a la hora de codificar, eliminando de las tareas a realizar el CRUD de las entidades.

Eso si, dedícale tiempo para entender como funciona, sino puedes puedes llegar a ser contraproducente, ejemplo: **[Lazy Load](http://www.tuprogramacion.com/programacion/lazy-load-en-entity-framework/" \o "Lazy load en Entity Framework)**

### **Programación**

# **Table Splitting y Entity Splitting en Entity Framework**

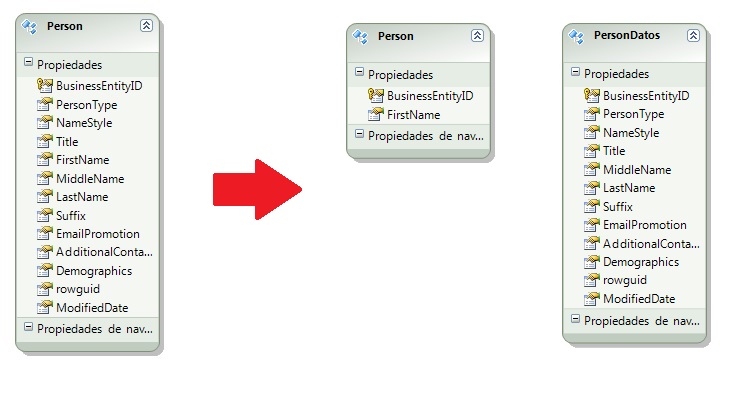
Una vez te inicias en cualquier orm, surge la necesidad de "ajustar" cada vez más tus consultas, tal como harías con el sql tradicional, para hacerlas más eficientes. Aquí entran estas dos técnicas denominadas **Table Splitting** y **Entity Splitting**.

## ¿Qué es Table Splitting y como usarlo?

Cuando haces una consulta con Entity Framework a la base de datos, éste te **mapea automáticamente los registros a entidades de tu sistema**. Ésto es algo muy cómodo y eficiente en muchas ocasiones, pero cuando las tablas empiezan a tener muchos campos o, se hacen consultas que reciben muchos registros, el rendimiento empieza haberse afectado.

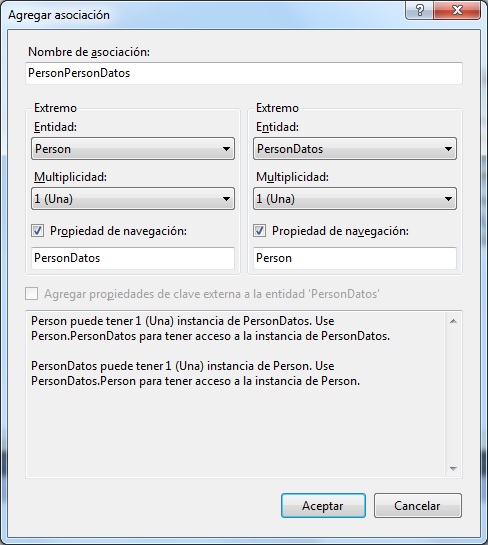
Para evitar ésto, puedes **dividir ciertas tablas que consideres que van a ser muy usadas**, pero en las que muchas veces sólo necesitas parte de sus datos.

Por ejemplo, si tienes un tabla de usuarios donde se hacen muchas consultas y sólo necesitas el ID y el nombre de usuarios. Podrías dividir esa tabla en una entidad "Person", y el resto de los campos en otra denominada personDatos. Con ésto, evitarías recoger todos los datos innecesarios como dirección, teléfono, población, etc... a no ser que los necesites explícitamente.



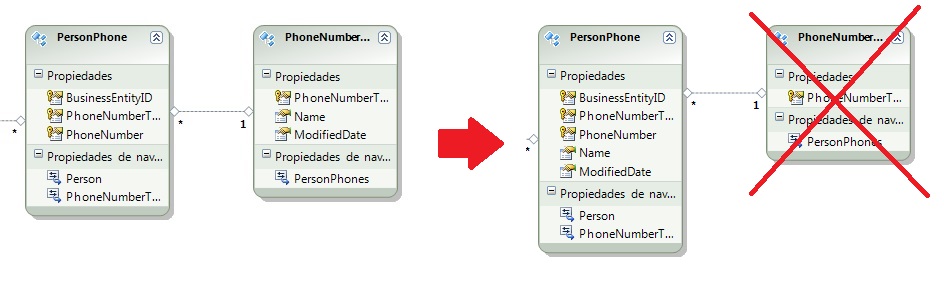
En el ejemplo vemos como hemos divido la entidad en dos, para ello basta con crear la nueva entidad, cortar y pegar los campos que deseemos.

Después, sencillamente habría que asociar estas dos entidades como si fueran dos tablas independientes:



## ¿Qué es Entity Splitting y como usarlo?

Entity Splitting sería el caso opuesto, es decir, **teniendo dos o más tablas, queremos asociarlas todas en una única entidad** para su posterior manejo. Obviamente estas tablas deben estar relacionadas.



Para ello, podemos usar directamente el interfaz gráfico de Entity Framework (.edmx), cortando y pegando los campos que queremos insertar. Después, eliminamos la entidad sobrante ya que ya no nos es de utilidad.

## Conclusión acerca del rendimiento con entity framework

Como he comentado en otros artículos, los ORMs son herramientas de trabajo muy potentes que nos facilitan en gran medida el trabajo diario. Pero hay que saber usarlos, ya que sino pueden afectar severamente el rendimiento de la aplicacion cuando la base de datos va creciendo

***Otro ejemplo:***

Ya hace bastante tiempo ([Tutorial 22](http://blogs.gamefilia.com/ollydbg/20-09-2009/26564/net-tutorial-22-base-de-datos-parte-i) y [Tutorial 23](http://blogs.gamefilia.com/ollydbg/24-09-2009/26674/net-tutorial-23-base-de-datos-parte-ii-access-sql-server-y-mysql-ejemplo-pr)) vimos como acceder a una base de datos y realizar las operaciones más comunes:

* Seleccionar
* Añadir
* Modificar
* Eliminar

En  aquellos tutoriales se realizaban las 4 acciones anteriores accediendo directamente a los campos de la base de datos y escribiendo "a mano" las sentencias SQL apropiadas para realizar cada acción.

Hoy os mostraré una forma distinta (ni peor ni mejor, distinta) de como acceder a una base de datos y realizar 4 acciones básicas.

Sin embargo, se usará lo que se conoce como ORM, o lo que es lo mismo: Nuestras tablas actúaran como "entidades", de tal forma que es posible acceder a todo el contenido de las tablas como si fuesen objetos.

La principal ventaja de esto es que no es necesario acordarse "de memoria" de las sentencias SQL, además esta metodología permite muy facilmente realizar apicaciones de N-Capas, o incluso aislar la capa de datos de la capa de negocio y crear un repositorio para comunicarnos via WCF con cualquier otra aplicación.

Antes de empezar dos cosas:

Uno de los ORMs más conocidos puede que sea [NHibernate](http://nhforge.org/" \t "_blank). Es un ORM de código abierto y es la conversión del Hibernate de Java para .NET

En ningún momento quiero dar a entender que NHibernate es mejor, peor, etc que Entity Framework 4 (hay un Entity Framework 5 disponible desde NuGet). Simplemente NHibernate existe desde antes de Entity Framework (Si la memoría no me falla, el Entity Framework se incluyó por primera vez con el SP1 del Visual Studio 2008)

Entity Framework (EF)  a priori solo sirve para trabajar contra SQL Server, mientras que NHibernate por ejemplo sirve para trabajar contra SQL Server, MySQL, Oracle, etc. Digo lo de "a priori" puesto que hay "conectores" para EF para trabajar con otros proveedores de base de datos distintos a SQL Server.

Para esta entrada se usará el Entity Framework 4 disponible en Visual Studio 2010. No será necesario descargar ningún paquete adicional ni nada por el estilo.

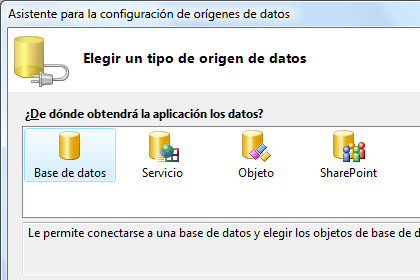
Puede que te sorprenda, pero esta entrada (y seguramente las siguientes) está escrita en C#. ¿Adios VB.NET? (bueno, ya veremos :D )

Por último, ignoro si lo que voy a explicar a continuación sirve para las versiones "Express" de Visual Studio, aunque me da que no es posible realizar esto en dichas versiones. La versión que se ha usado para esta entrada es VS 2010 Professional.

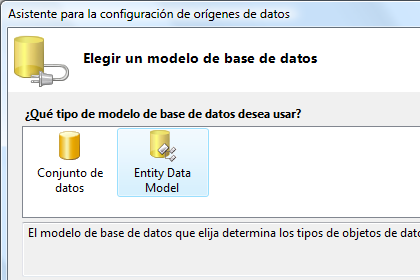
Bien, comencemos.

**Crear un nuevo modelo desde cero**

Cuando añadimos un nuevo origen de datos nos encontramos con esta ventana:

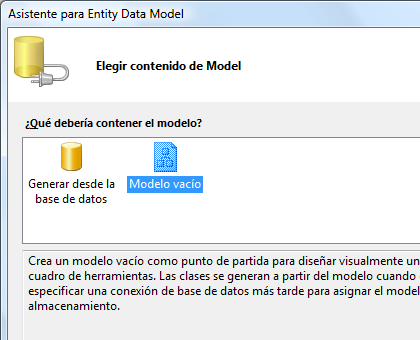


**Seleccionaremos Base de datos y se muestra esta otra ventana:**



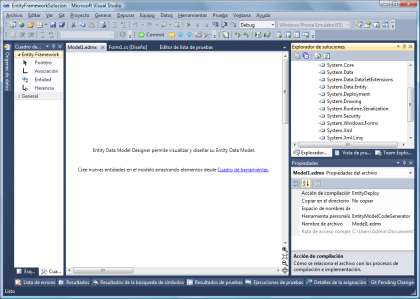
**En este caso elegiremos Entity Data Model.**

Como observais a continuación es posible generar el modelo desde una base de datos (lo veremos luego) o generar un modelo vacio.

****

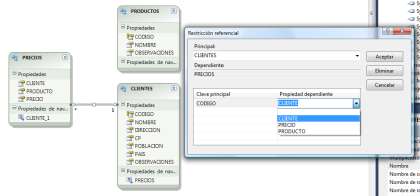
**En este primer ejemplo seleccionaremos que queremos usar un modelo vacio.**

A continuación se muestra la vista de diseño de nuestro modelo (vacio obviamente):

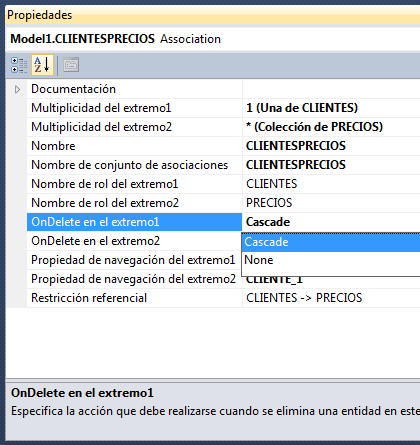
[](http://dl.dropbox.com/u/4152145/blogs.gamefilia.com.ollydbg/Tutorial57/images/EF_4.png)

**En esa pantalla podremos ir agregando entidades (básicamente tablas), con sus respecitvos campos, relacciones, etc.**

**Como se muestra a continuación es posible asociar campos de las distintas entidades:**

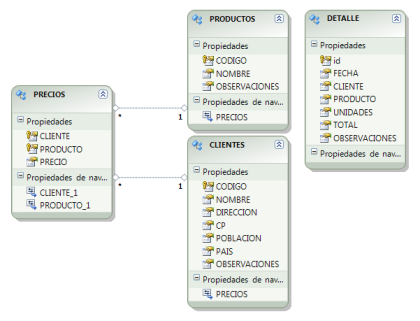
[](http://dl.dropbox.com/u/4152145/blogs.gamefilia.com.ollydbg/Tutorial57/images/EF_5.png)

...y determinar el comportamiento de dichas asociaciones:

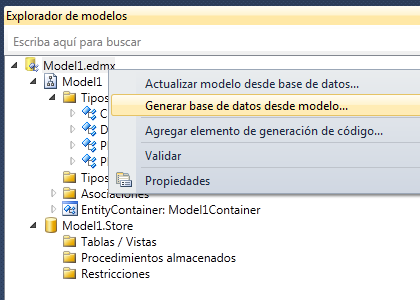


Con la asociación anterior, estamos creando una foreign key (FK) en la entidad PRECIOS, de tal forma que cuando se elimine un registro de la entidad CLIENTES, automáticamente se eliminarán los registros de PRECIOS cuyo cliente sea el que se ha eliminado de CLIENTES: Vamos, lo que se conoce como borrado en cascada de un FK de toda la vida.

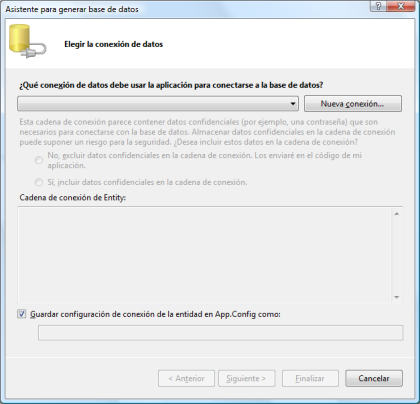
Podemos añadir mas entidades, campos, etc. Todo ello de forma visual y bastante intuitiva:



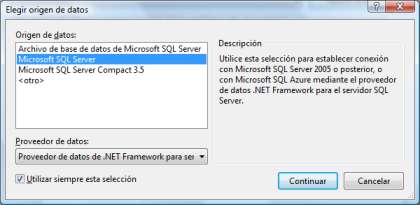
**Una vez finalizado el diseño, podemos, a partir de dicho diseño, decirle a Visual Studio que genere un script para realmente tener esa base de datos en nuestro servidor SQL Server:**



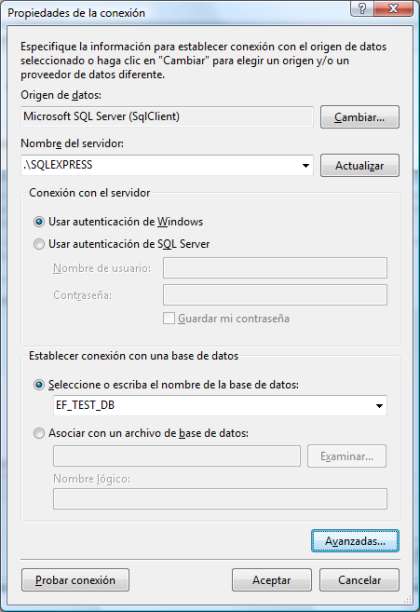
Si no teniamos la conexión de datos, podemos crear una nueva conexión:



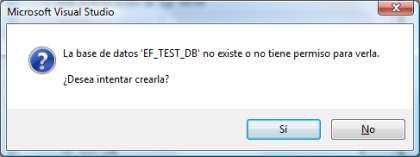
**Indicamos Microsoft SQL Server**:



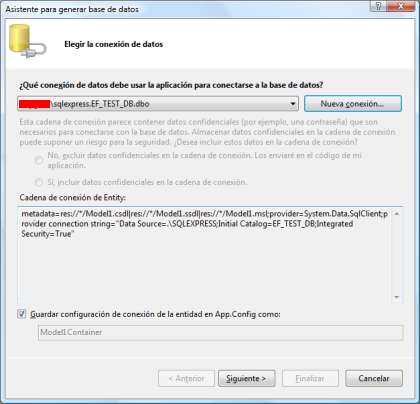
**Establecemos los datos de nuestra conexión e indicamos el nombre de la Base de datos a crear:**



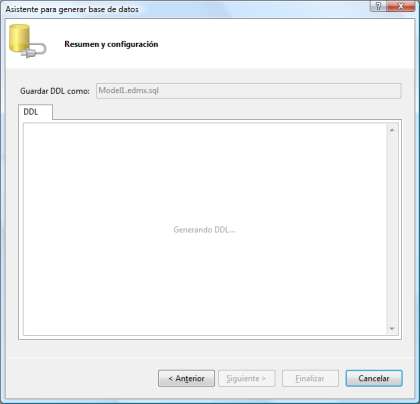
**Visual Studio nos indica que la base de datos no existe y si queremos crearla:**



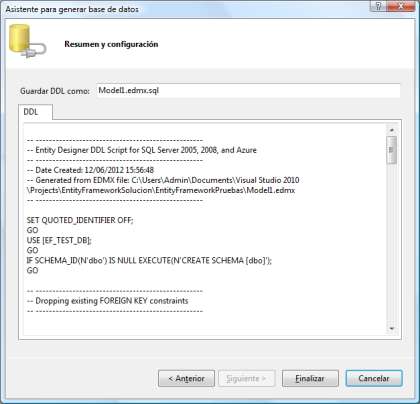
Pulsamos en Siguiente:



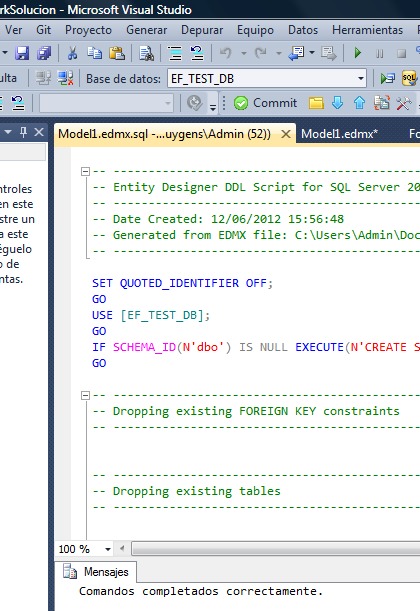
Y en esta pantalla se está construyendo el script para la generación de la base de datos:



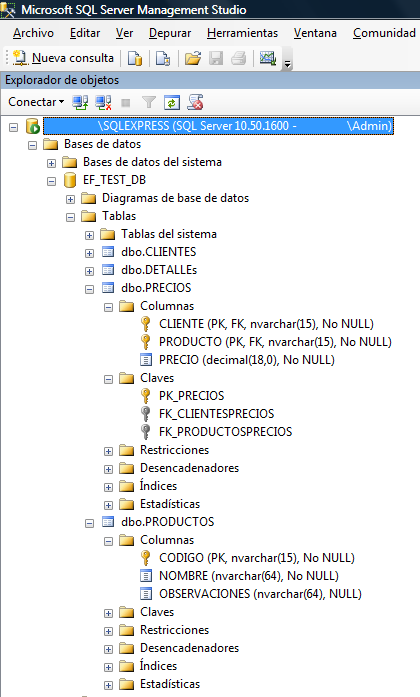
**Una vez generado el script se muestra tal que así y se genera un nuevo archivo en nuestra solución:**



**Desde el propio Visual Studio se puede ejecutar el script:**



Si accedemos a nuestro SQL Server mediante el Management Studio, vemos que efectivamente, se ha creado la base de datos y dentro de ella están las tablas de nuestro módelo que diseñamos desde Visual Studio:

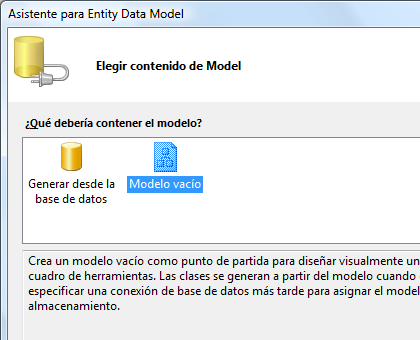


**Obtener un modelo de entidades desde una base de datos existente**

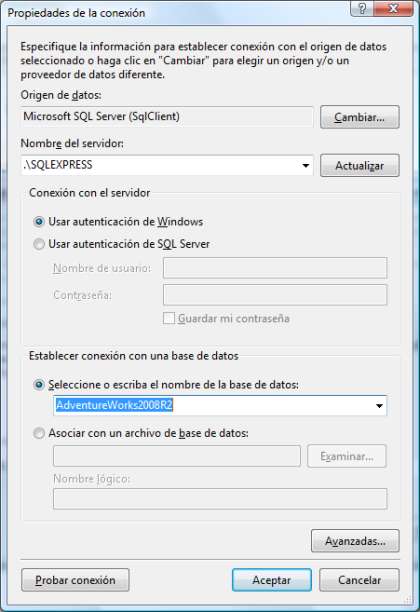
**Para este ejemplo usaré una base de datos de ejemplo que proporciona Microsoft, en este caso la base de datos AdventureWorks**

Puedes encontrar dicha base de datos en esta dirección: <http://sqlserversamples.codeplex.com/>

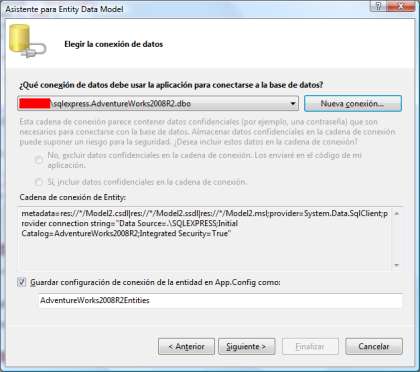
En el asistente del modelo, escogeremos Generar desde base de datos:



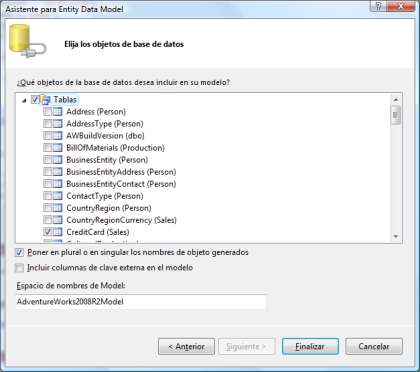
Si no tenemos una conexión establecidad, pulsaremos en el botón Nueva conexión y se mostrará esta ventana:



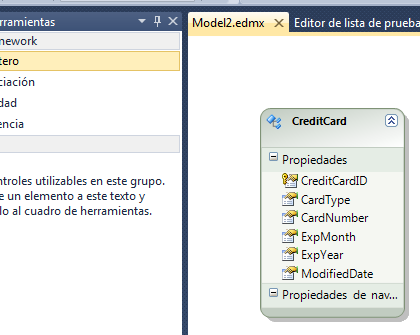
De nuevo en la pantalla del asistente dejaremos los datos tal así:



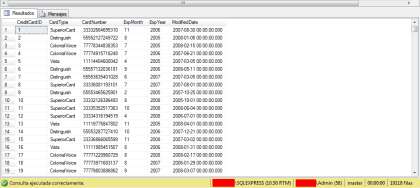
El asistente recupera toda la información de la base de datos (tablas, vistas y stored procedures). Para este ejemplo usaremos únicamente la tabla CreditCard:



Finalmente Visual Studio genera la entidades correspondientes y nos las muesrta en nuestra vista de diseño.

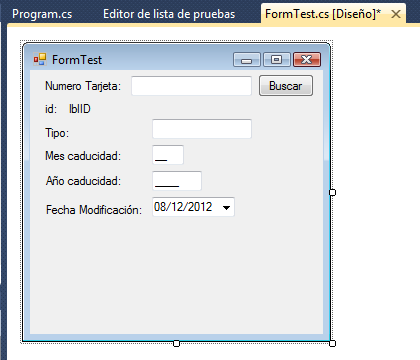


Si abrimos dicha tabla con el Visual Studio o con el Management Studio observamos todos los datos que contiene dicha tabla:

[](http://dl.dropbox.com/u/4152145/blogs.gamefilia.com.ollydbg/Tutorial57/images/EF_22.png)  
(Haz click para agrandar)

Seleccionar registros del modelo

En nuestro formulario creamos una plantilla con los siguientes controles:



Lo primero que haremos será definir una variable privada llamada dbEntity de esta forma:

private AdventureWorks2008R2Entities dbEntity;

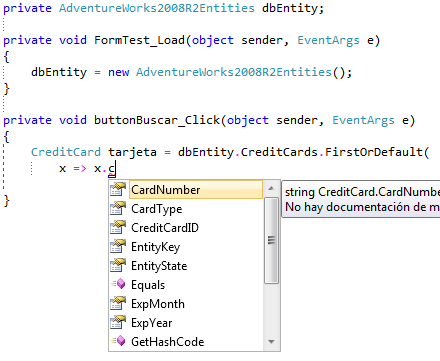
Y en el form load la inicializaremos así:

dbEntity = new AdventureWorks2008R2Entities();

Nota: Ya que se que se podría haber usado un using (db = new entidad) { } pero a efectos del tutorial lo voy a dejar tal como está.

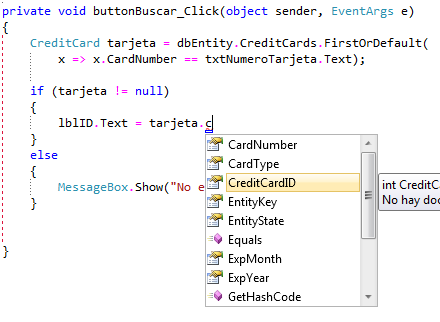
Bien, al pulsar en el botón "Buscar" se recuperarán los datos de aquella tarjeta cuyo campo "CardNumber" sea el especificado en la caja de texto.

En el evento click del botón escribimos lo siguiente:



Si observáis, la "gracia" de este asunto es que en el propio Intellisense se nos muestran todos los objetos de la entidad.

Lo mismo ocurre cuando queremos asignar el valor de un propiedad a un control:

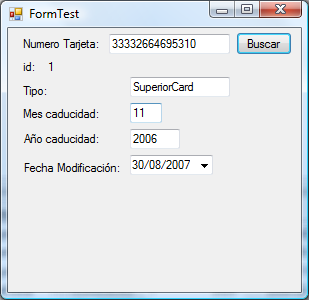


El código final sería este:

private void buttonBuscar\_Click(object sender, EventArgs e)  
{  
CreditCard tarjeta = dbEntity.CreditCards.FirstOrDefault(  
x => x.CardNumber == txtNumeroTarjeta.Text);

   if (tarjeta != null)  
{  
lblID.Text = tarjeta.CreditCardID.ToString();  
txtTipo.Text = tarjeta.CardType;  
mskMesCaducidad.Text = tarjeta.ExpMonth.ToString();  
mskAñoCaducidad.Text = tarjeta.ExpYear.ToString();  
dTPModificacion.Value = tarjeta.ModifiedDate;  
}  
else  
{  
MessageBox.Show("No existe ninguna tarjeta con el código " + txtNumeroTarjeta.Text);  
}  
}

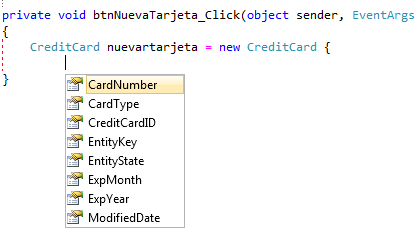
Ejecutamos la aplicación y comprobamos que es lo que sucede al introducir un Numero de tarjeta:



Efectivamente, vemos como se recuperan los datos y se colocan en los controles correspondientes.

Añadir un registro

Añadimos un nuevo botón a nuestro formulario para dar de alta una nueva tarjeta. En coódigo del evento click escribimos lo siguiente:



De nuevo observar que en el propio intellisense se muestran todos los campos de la "entidad" correspondiente.

El código completo es el siguiente:

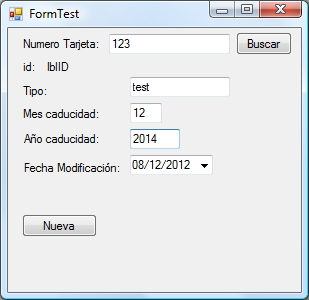
private void btnNuevaTarjeta\_Click(object sender, EventArgs e)  
{  
CreditCard nuevartarjeta = new CreditCard  
{  
CardNumber = txtNumeroTarjeta.Text,  
CardType = txtTipo.Text,  
ExpMonth = Convert.ToByte(mskMesCaducidad.Text),  
ExpYear = Convert.ToInt16(mskAñoCaducidad.Text),  
ModifiedDate = dTPModificacion.Value  
};

   dbEntity.AddToCreditCards(nuevartarjeta);              
//dbEntity.CreditCards.AddObject(nuevartarjeta);

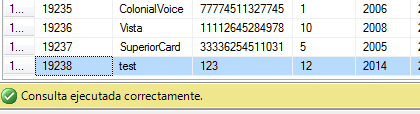
   dbEntity.SaveChanges();

   MessageBox.Show("La tarjeta se ha añadido a la base de datos");  
}

Lanzamos la aplicación y verificamos:



Al pulsar en el botón "Nueva" se guarda el registro. si verificamos los datos en la propia tabla vemos que efectivamente, el nuevo registro está en la tabla:



Eliminar un registro

El código para eliminar un registro, a partir del campo CardNumber es el siguiente:

private void btnEliminarTarjeta\_Click(object sender, EventArgs e)  
{  
CreditCard borrartarjeta = dbEntity.CreditCards.FirstOrDefault(  
x => x.CardNumber == txtNumeroTarjeta.Text);

   dbEntity.DeleteObject(borrartarjeta);  
dbEntity.SaveChanges();

   MessageBox.Show("La tarjeta se ha borrado de la base de datos");  
}

Modificar un registro

private void btnModificarTarjeta\_Click(object sender, EventArgs e)  
{  
CreditCard modificartarjeta = dbEntity.CreditCards.FirstOrDefault(  
x => x.CardNumber == txtNumeroTarjeta.Text);

   modificartarjeta.CardType = txtTipo.Text.Trim();  
modificartarjeta.ExpMonth = Convert.ToByte(mskMesCaducidad.Text);  
modificartarjeta.ExpYear = Convert.ToInt16(mskAñoCaducidad.Text);  
modificartarjeta.ModifiedDate = dTPModificacion.Value;

   dbEntity.SaveChanges();

   MessageBox.Show("Se ha modificado los datos de la tarjeta: "+ txtNumeroTarjeta.Text);  
}

Mostrar datos en un GridView

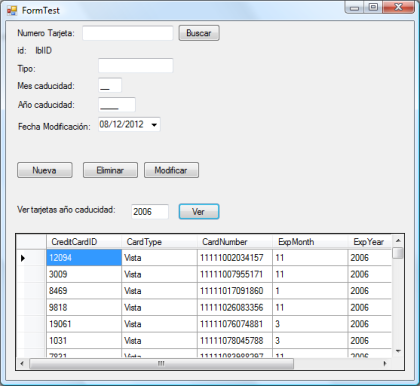
Este código muestra los datos en un gridview de todas aquellas tarjetas cuyo campo "año de caducidad" es un determinado valor y además se muestran los datos ordenados por "numero de tarjeta":

private void btnVer\_Click(object sender, EventArgs e)  
{  
short vMes = Convert.ToInt16(mskVerAño.Text);

   var query = from c in dbEntity.CreditCards  
where c.ExpYear == vMes  
orderby c.CardNumber  
select c;

   var tarjetas = query.ToList();  
dGWver.DataSource = tarjetas;             
}

El resultado sería similar a este:



Ejecutar un "select" con una instrucción SQL

Aunque mediante LINQ es casi improbable que sea innecesario ejecutar una consulta SQL lo podemos hacer usando el método ExecuteStoreQuery

Un ejemplo:

private void btnSQLNativa\_Click(object sender, EventArgs e)  
{  
   string query = "SELECT \* FROM Sales.CreditCard WHERE ExpYear=2006 AND CardNumber LIKE '%1111138%'";

   var tarjetas = dbEntity.ExecuteStoreQuery<CreditCard>(query);

   foreach (var item in tarjetas)  
{  
Console.WriteLine("{0} {1}",item.CardNumber,item.ExpYear);  
}  
 }

Ejecutar un "insert", "delete", o "update" con una instrucción SQL

Se pueden ejecutar comandos usando sentencias "nativas" mediante el comando ExecuteStoreCommand

Un ejemplo:

private void btnCommandNativo\_Click(object sender, EventArgs e)  
{  
   string query = "DELETE FROM Sales.CreditCard WHERE CardNumber ='123'";

   var cnt = dbEntity.ExecuteStoreCommand(query);

   Console.WriteLine(cnt);  
}

En el ejemplo anterior, si hay un registro con el valor CardNumber = '123' en la base de datos, éste se borra y el valor de cnt será igual a 1 (si se ha borrado 1 registro) o 44 (si se han borrado 44 registros)

Como se ve, el uso de un ORM, Entity Framework en este caso es realmente sencilla, y clara. Además, te evitas estar "re-inventado la rueda" para el tema del acceso a datos.

Bueno, esto era una introducción. Entity Framework y en general cualquier otro ORM es mucho más complejo. De todos modos, se tenéis alguna duda, pues ya sabéis,