Bloque 2. C# y Visual Studio
UT 3. Confección de interfaces de usuario y creación de componentes visuales

TEMA 13

Acceso a datos. ADO.NET

1. ADO.NET. Introducción

ADO.NET es la tecnología de acceso a bases de datos del mundo .NET utilizada actualmente. Es una evolución de las anteriores ODBC (Open DataBase Connectivity) y ADO (ActiveX Data Objects).

ODBC fue el primer estándar para comunicarse con bases de datos. Se definían unos parámetros que identificaban el tipo de base de datos, el nombre de usuario, contraseña y, dependiendo del tipo de origen de datos, algún otro parámetro. Con esta definición los programas estaban preparados para trabajar con bases de datos. Este modelo de objetos ODBC tuvo su evolución en un modelo más compacto y mucho más potente que se llamó OLEDB todavía vigente en muchísimos sitios. Como esta tecnología era muy difícil de manejar, Microsoft escribió unos controles para facilitar la labor que se llamaron ActiveX Data Objects o ADO.

ADO.NET es la nueva versión del modelo de objetos ADO ofreciendo todo lo necesario para que podamos acceder a las bases de datos.

ADO.NET representa el sustrato que compondrá la base de las aplicaciones .NET compatibles con datos. A diferencia de ADO, ADO.NET se ha diseñado siguiendo específicamente unas directrices más generales y menos orientadas a la base de datos. ADO.NET reúne todas las clases que permiten el manejo de datos. Estas clases representan los objetos que contienen datos y que muestran las capacidades normales de las bases de datos: índices, ordenación, vistas.

ADO.NET es una solución bastante distinta de ADO. Se trata de un nuevo modelo de programación para el acceso a datos que necesita un compromiso y un entendimiento completos, así como un acercamiento distinto.

Aparece el concepto de DataSet, que son los datos que recogeremos en nuestras operaciones con ADO.NET. El **DataSet** de ADO.NET es una representación de datos residente en memoria que proporciona un modelo de programación relacional coherente independientemente del origen de datos que contiene. Un **DataSet** representa un conjunto completo de datos, incluyendo las tablas que contienen, ordenan y restringen los datos, así como las relaciones entre las tablas.

1.1. Modelo de objetos

ADO.NET está basado en una arquitectura desconectada de los datos, es decir, no tenemos "una base de datos abierta" permanentemente sino que cuando necesitamos hacer alguna operación, nos conectamos, hacemos la consulta/operación y nos desconectamos. Está comprobado que mantener los recursos reservados mucho tiempo perjudica el rendimiento ya que los usuarios conectados tienen que ser limitados y debe trabajar intensamente con los bloqueos para impedir operaciones simultáneas sobre datos no deseadas.

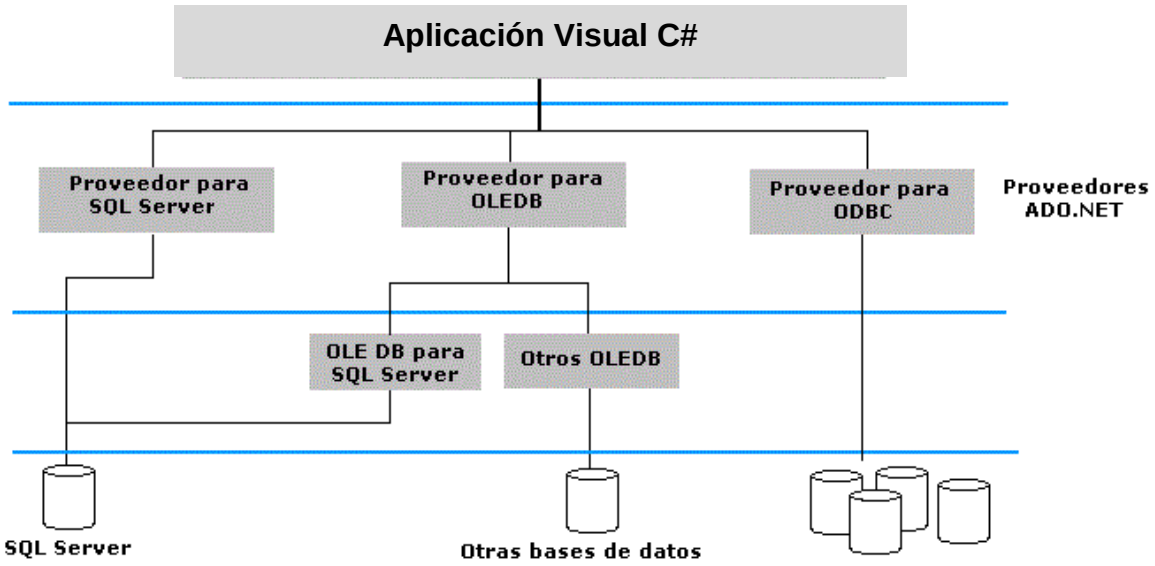
Con ADO.NET se consigue estar conectado al servidor sólo lo estrictamente necesario para realizar la operación de carga de datos en el DataSet, así se minimiza el tiempo de conexión y de uso de la base de datos. Se pueden conectar más usuarios, disminuyen los tiempos de respuesta y se aceleran las operaciones dentro de la base de datos.

Está claro que para obtener datos de un origen debemos establecer una conexión. En ADO.NET disponemos de un administrador de proveedores de datos para hacer esta operación.

Con ADO.NET se proporcionan varios proveedores:

- Proveedor para SQL Server, sólo trabaja para la base de datos de Microsoft SQL Server.
- Proveedor para OLEDB, para los orígenes de datos OLEDB, que son el resto de las bases de datos.
- Proveedor para Oracle.
- Proveedor para base de datos ODBC. Para antiguas bases de datos que no tienen capacidad OLEDB.

La estructura de los controladores de la tecnología ADO.NET es:



Por un lado, vemos un controlador exclusivo para SQL Server, uno "universal" que se conecta con cualquier base de datos que soporte OLEDB (prácticamente todas) y otro más para las antiguas bases de datos ODBC. Además hay una conexión entre OLEDB, este último y SQL

Server, es decir, podríamos definir en la conexión que sea una OLEDB y el destino SQL Server, estaría bien y nos daría buen rendimiento pero Microsoft quiso dar un juego distinto para estas bases de datos de servidor y les ha proporcionado un proveedor exclusivo para ellos. La diferencia al utilizar unos y otros está en la sintaxis, pero veremos que las diferencias son mínimas.

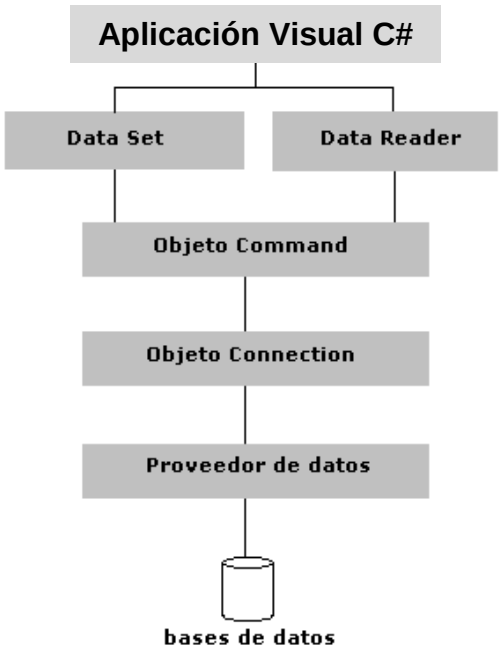
Como hemos dicho antes, un Dataset es una "foto" de unos registros recuperados de una base de datos que se almacenan en un sitio virtual y que contiene una o más tablas reales de la base de datos. Puesto que se cargan y nos desconectamos de la base de datos se convierten en datos pasivos, es decir, si hay cambios en la base de datos no se actualizarán y deberemos recargarlos para que aparezcan los últimos cambios reflejados.

Pero... si estoy trabajando con muchos datos esto será un inconveniente porque ¡¡tendré que volver a cargarlos cuando me conecte otra vez!! Pues sí, lo que parece que no es la mejor solución: cargar una copia completa de los datos de trabajo tiene algunos detalles que lo convierten en ventajas. Las ventajas ya las conocemos: menos tráfico de la red, del motor de base de datos, ... pero aparece otro muy interesante: la **persistencia**.

La persistencia es un mecanismo por el que podemos almacenar el estado de una situación: valores de variables, tablas,... en un soporte. Luego podemos volver a cargarlos y estaríamos en la misma situación inicial. Por ejemplo, cuando en nuestro equipo damos la opción "Hibernar" el equipo se desconecta pero almacena su estado, luego al pulsar el botón otra vez aparecemos exactamente donde estábamos: los datos son persistentes.

En la práctica podremos cerrar un programa y mantener los DataSets que queramos de forma persistente, así al poner el programa en marcha otra vez nos encontramos con esos datos disponibles para trabajar. Hemos ahorrado tiempo al servidor, tráfico de red,... luego ya vemos que uno de los elementos más importantes de nuestro ADO.NET son los DataSet.

El punto de partida del acceso a datos es la conexión con el proveedor de datos seleccionado, esta operación la hace el objeto Connection. El flujo de información podría verse de esta forma:



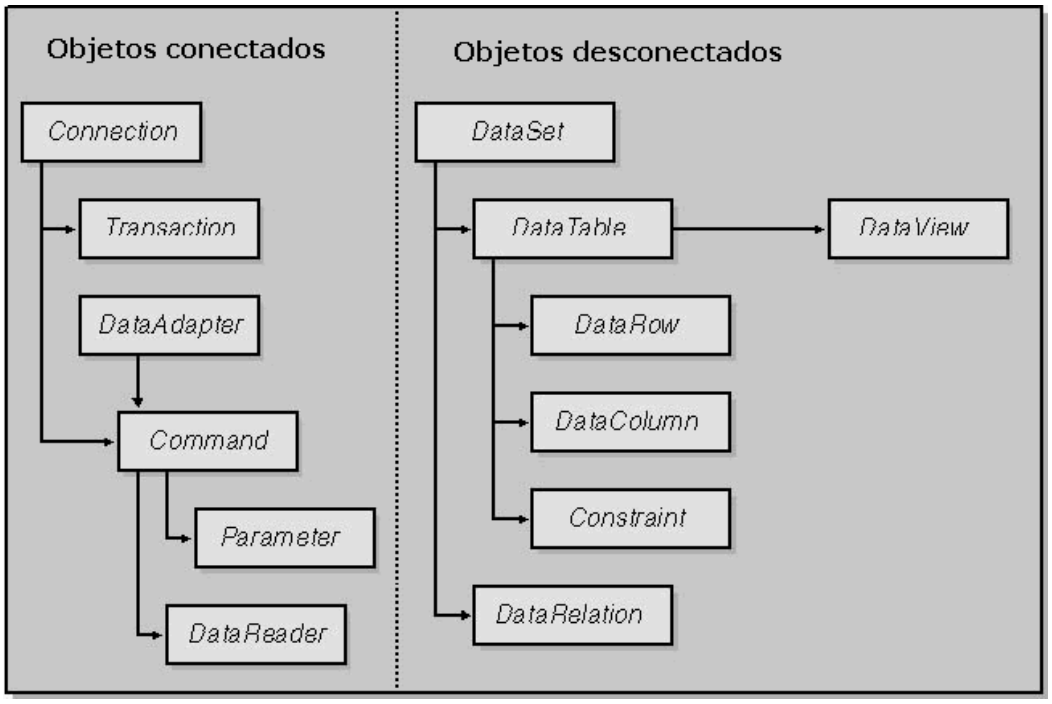
Visto de arriba hacia abajo, nuestro programa quiere hacer una consulta para rellenar una tabla de datos en pantalla ("DataSet" o "DataReader") para esto utilizará un comando "Command" que accederá a los datos mediante la conexión "Connection". Esta última ya sabe cómo y dónde debe realizar la consulta.

Para poder trabajar con datos en nuestras aplicaciones necesitaremos 6 elementos. Puede haber alguna variación en algún caso especial, pero hazte idea de que estas son las seis partes que necesitaremos para acceder a datos:

- Origen de datos, es decir, los datos en el formato admitido: SQL Server, Access, Oracle, ...
- Proveedor para acceder a esa base datos. Recuerda que ya tenemos para las bases de datos OLEDB y SQL Server. Las OLEDB son prácticamente todos las "pequeñas": Access,...
- El objeto "Connection" de ADO.NET. La conexión con los datos anteriores.
- El objeto "Command", contiene instrucciones de cómo leer datos y otras operaciones.
- Los objetos "DataReader" y "DataSet", son los datos que estamos leyendo o manipulando.
- Los controles de presentación de datos como el DataGridView.

La secuencia es bastante lógica: primero debemos saber dónde están los datos y tener el controlador para poder conectarnos a esos datos. Luego establecemos una conexión y ejecutamos una serie de comandos. Estos comandos almacenan los resultados en unos objetos que finalmente exponemos en una cuadrícula de Visual Studio.

Veamos el modelo de objetos de ADO.NET, es decir, de qué objetos dispongo para trabajar:

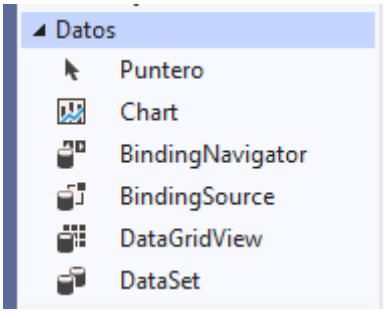


Como en otras ocasiones es muy útil tener esta estructura a mano porque nos servirá para comprender la jerarquía de los objetos y nos ayudará al escribir el código porque sabremos los objetos que debemos utilizar.

Como puedes ver tenemos unos objetos conectados y otros desconectados. Como ya hemos comentado los referentes a la recuperación de datos,... es decir, los DataSet trabajan de forma desconectada así que son los que aparecen en la parte derecha.

1.2. Clases del proveedor de datos

Si echamos un vistazo a la parte de acceso a datos en una aplicación de Windows Forms en los controles veremos estos:



Nota: SQL Server es el servidor de base de datos de Microsoft. MS Access sólo es un pequeño programa para soluciones locales de bases de datos, si necesitamos potencia, acceso multiusuario, seguridad,.. necesitaremos un servidor de base de datos como SQL Server. Es fácil de aprender y proporciona una potencia incomparable a la de Access, por ejemplo.

Hemos visto que tenemos tres grandes proveedores. El de ODBC lo descartamos por ser ya obsoleto así que nos centraremos en los dos principales: SQL Server y OLE DD. Utilizaremos las clases SQL para obtener acceso a las tablas de SQL Server, ya que se dirigen directamente a la API interna del servidor de base de datos, omitiendo el nivel intermedio representado por el proveedor OLE DB.

En el caso del proveedor OLE DB esto significa que es genérico para todos los tipos de bases de datos que funcionen con este estándar, cuando hagamos ejemplos con Access utilizaremos lógicamente este formato. Hoy en día cualquier base de datos que se precie se puede programar con OLE DD. A lo largo de este tema comentaremos sólo los casos más generales que son los de bases de datos SQL Server y OLE DB

Cada proveedor de datos contiene las mismas clases base: Connection, Command, DataReader, Parameter y Transaction dependiendo su prefijo del proveedor. Por ejemplo, el proveedor de datos de SQL Server Client .NET tiene el objeto SqlConnection, y OLE DB .NET tiene el OleDbConnection. Esto es por la gran importancia que tiene SQL Server como servidor de bases de datos para Microsoft. .

Cada proveedor de datos tiene su propio "namespace". Los dos incluidos con .NET son subconjuntos del namespace "System.Data", donde se encuentran los objetos desconectados. El proveedor OLE DB .NET reside en el namespace System.Data.OleDb y el de SQL Server en System.Data.SqlClient.

Recordemos que un "namespace" no es más que un grupo lógico de objetos, por ejemplo, el espacio de nombres de los gráficos. Veamos ahora espacios de nombres disponibles en ADO.NET.

Los componentes de ADO.NET los tenemos en distintas bibliotecas de clases y todas juntas conforman ADO.NET. Fíjate en los espacios de nombres (Namespace) que tenemos disponibles en ADO.NET

Namespace	Función
System.Data	Contiene las clases fundamentales del núcleo de ADO.NET. Incluye los objetos "DataSet" y "DataRelation" para modificar la estructura relacional de los datos. Estas clases son independientes del tipo de controlador que se utilice.
System.Data.Common	No se utilizan en nuestro código. Las utilizan otras clases proveedoras de datos.
System.Data.Oledb	Contiene las clases necesarias para conectarnos a orígenes de datos OLE DB, y la ejecución de comandos: "OleDbConnection" y "OleDbCommand"
System.Data.SqlClient	Contiene las clases para conectarnos y ejecutar comandos en un servidor de base de datos SQL Server. Las propiedades y métodos de "SqlConnection" y "SqlCommand" son las mismas que cuando la bbdd es de tipo OLEDB con los objetos anteriores.
System.Data.SqlTypes	Contiene estructuras para tipos de datos específicos de SQL Server, como SqlMoney y SqlDateTime. De esta forma no hay que convertir tipos de datos.
System.Data.ODBC	Contiene las clases para conectarnos y ejecutar comandos a través de un controlador ODBC: "OdbcConnection" y "OdbcCommand

Aunque tienen distintos espacios de nombres las propiedades y métodos son prácticamente iguales, tanto que incluso podríamos cambiar sólo los nombres de los objetos que vamos a ver ahora por su equivalente de SQL Server a Oracle y nuestra aplicación funcionará con normalidad. Veamos las clases de los proveedores de nombres:

Proveedores -->	SQL SERVER	OLE DB	ORACLE	ODBC
Connection	SqlConnection	OleDBConnection	OracleConnection	OdbcConnection
Command	SqlCommand	OleDBCommand	OracleCommand	OdbcCommand
DataAdapter	SqlDataAdapter	OleDBDataAdapter	OracleDataAdapter	OdbcDataAdapter

La diferencia real entre ellas son los nombres, la cadena de conexión y algunas características avanzadas que pueden ofrecer algunos.

Vamos a hacer los siguientes ejemplos con SQL Server. ¿Por qué? Es la mejor forma de meternos en el mundo profesional. Access es muy limitada y no debemos utilizarla más allá de pequeñas aplicaciones de escritorio. SQL Server es relativamente sencillo de aprender y el rendimiento y seguridad que obtenemos es incomparable.

1.3. System.Data

Por su importancia veamos ahora el espacio de nombres System.Data para ver qué clases contiene, ya que como hemos dicho son la base de ADO.NET:

- DataSet. Almacén de datos para ADO.NET. Representa una base de datos desconectada del proveedor de datos. Almacena tablas y sus relaciones.
- DataTable. Contenedor de datos. Es una estructura de filas y columnas que almacenan datos
- DataRow. Registro que almacena n valores. Representa en ADO.NET a una fila de una tabla de datos.
- DataColumn. Contiene la definición de la columna.
- DataRelation. Enlace entre dos o más columnas iguales de dos o más tablas.
- Constraint. Reglas de validación de las columnas de una tabla.
- DataColumnMapping. Vínculo lógico entre una columna de un objeto del DataSet y la columna física de la tabla de la base de datos.
- DataTableMapping. Vínculo entre una tabla del DataSet y la tabla física de la base de datos.

2. Un avance de las tablas y relaciones

La manera lógica da manipular datos es obviamente una base de datos. Desde el almacenamiento de unos usuarios con sus contraseñas hasta la lista de artículos de una tienda. Esta información suele estar relacionada, es decir, mantiene una coherencia de datos y comparten información: ventas a clientes, compras a proveedores... Las bases de datos tienen cada una su formato particular y su tecnología.

SQL Server es un sistema de administración de bases de datos relacionales. Esta es una de las claves: las relaciones entre las tablas. Muy pocas veces nos encontraremos con bases de datos donde las tablas están aisladas y no tienen relaciones entre sí. Cualquier pequeño problema que te plantees con toda seguridad necesitará de varias tablas relacionadas para obtener una información coherente. Por ejemplo, imagina nuestra cuenta del banco, tiene una serie de datos

personales: nombres, dirección,.... Habitualmente utilizamos la tarjeta para realizar toda clase de operaciones que obviamente se registrarán una base de datos. Estos movimientos o transacciones no almacenan nuestros datos completos: nombre, dirección, transacción realizada e importe, sino que almacena sólo nuestro número de cliente que tenemos asignado en nuestra cuenta. Por tanto, al sacar dinero solo almacenará: número de cliente, tipo de transacción, cantidad y fecha de la operación. Por tanto, estamos relacionando por el número de cliente dos tablas: una para nuestros datos y otra para las operaciones.

Visto este ejemplo, vamos con otro más práctico para nuestras pruebas. La forma más natural de hacer las relaciones es de "uno a muchos":

- Un usuario puede comprar muchos elementos.
- Un elemento es comprado por muchos usuarios.

En el momento que tenemos dos relaciones de éstas se convierte ya en tipo de "muchos a muchos": muchos usuarios compran muchos elementos.

Veamos de forma abreviada una tabla de usuarios y una de unos artículos que forman parte de un catálogo:

UserId	UserName
905682e6-f67b-...	antonio
a18fc8bf-ec1e-4...	jose2
1f0b3846-8e72-...	Josemari
1299e1eb-968e-...	antonio2
ff45a218-f080-4...	jose

ArticuloID	Nombre
1	Introducción a Visual Basic
2	Los For-Terrier
3	Fotografia
4	Fotografia II
5	ASP.NET

La tabla de la derecha muestra unos libros que forman parte de un catálogo de una tienda. En la de la izquierda la lista de los usuarios que son los posibles compradores que tienen dos campos: un identificador de usuario (un poco raro pero válido) y el nombre de usuario. Ahora viene lo "complicado"... ¿cómo conectamos esas dos tablas? Quiero gestionar de alguna forma que esas personas que tengo en una tabla de usuarios puedan realizar compras de los libros de la tabla de la derecha.

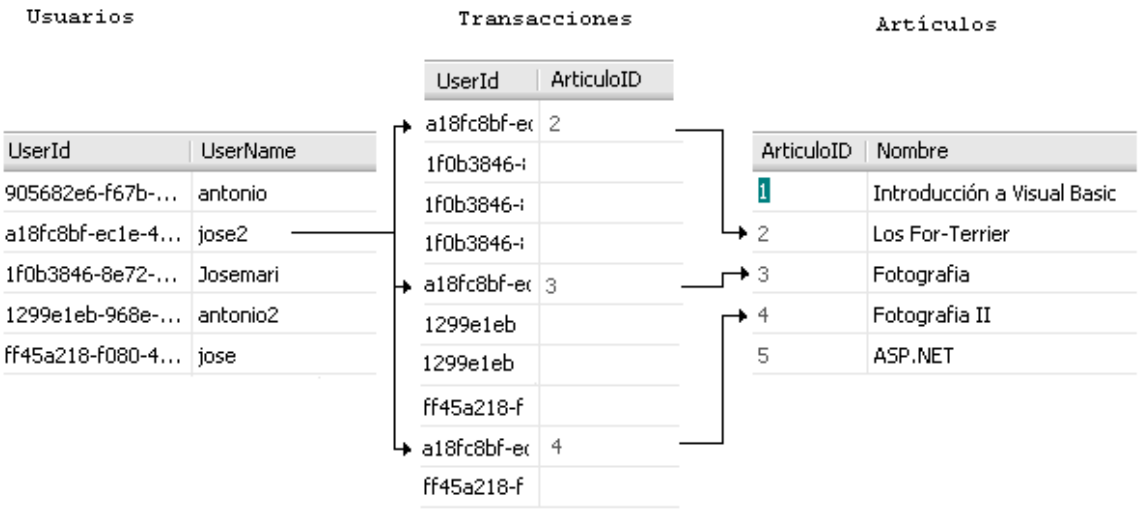
La respuesta es "creando una tabla que almacene las compras realizadas". Cada vez que un usuario realice una compra debemos almacenar esta información por lo menos:

- Usuario
- Libro
- Fecha de compra
- ¿Está pagado?

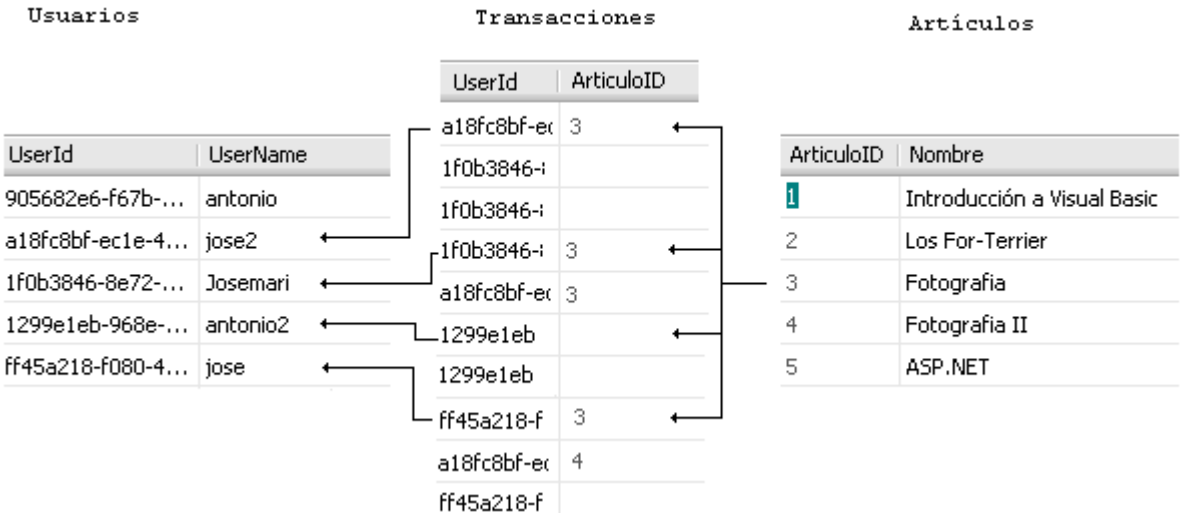
Así que debemos almacenar los identificadores de esas dos tablas. No es necesario que almacenemos el nombre y apellidos ni el título del libro, esos datos no, simplemente almacenaremos el valor del identificador del usuario "UserId" y del libro "ArticulosId". Estos identificadores son únicos y apuntan a una sola fila de cada tabla, luego son perfectos para almacenar esta información. Nos ahorramos además toda esa información de usuarios y libros que no hacen falta para almacenar nuestra "transacción".

Luego, a la hora de hacer un informe en lugar de poner esos indicadores simplemente los sustituiremos por una consulta en las tablas de los usuarios y artículos, de esto último se encarga el propio lenguaje de consulta.

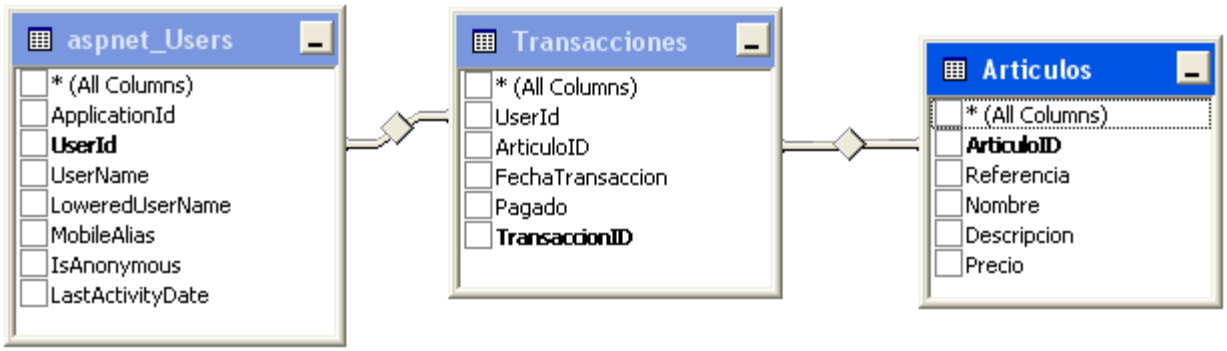
Así que en cada compra que hagamos meteremos una línea en la tabla de transacciones indicando quién y qué ha comprado. Aquí la relación es "muchos a muchos" ya que muchos usuarios pueden comprar el mismo libro e igualmente muchos libros pueden ser comprados por un usuario.



Ahí vemos como el usuario "XXX" ha hecho 3 transacciones con 3 libros distintos. Y al revés:



A la hora de diseñar las tablas y establecer esas relaciones artículos-transacciones-usuarios utilizaremos las herramientas de diseño de bases de datos que nos muestran algo parecido a esto:



Para hacer relaciones las dos tablas deben tener un campo en común que tenga el mismo tipo de datos. Si nuestra tabla de transacciones tiene una lista de artículos, esos identificadores deben ser del mismo tipo de datos que los identificadores de la tabla de artículos. Al estar relacionadas se establece una interesante coherencia ya que si borramos un usuario borraremos todas las transacciones de ese "ID" de la tabla correspondiente y ya no quedará información aislada.

3. Los primeros pasos con SQL Server

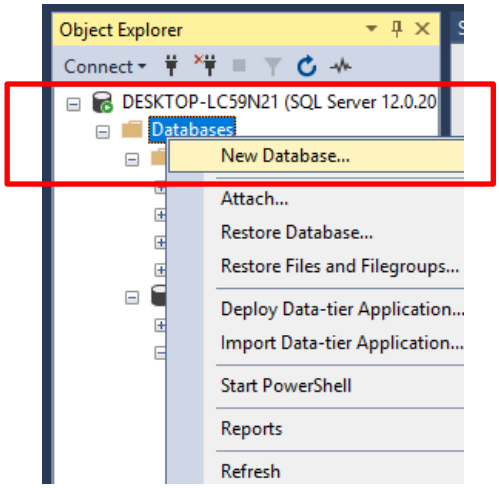
Existen varias versiones de SQL Server, dos de ellas son gratuitas y se llaman SQL Server Developer y SQL Server Express. Este servidor tiene gran potencia y nos metemos de lleno en el mundo profesional. Utilizaremos SQL Server 2022 Express.

Se puede descargar desde: https://www.microsoft.com/es-es/sql-server/sql-server-downloads

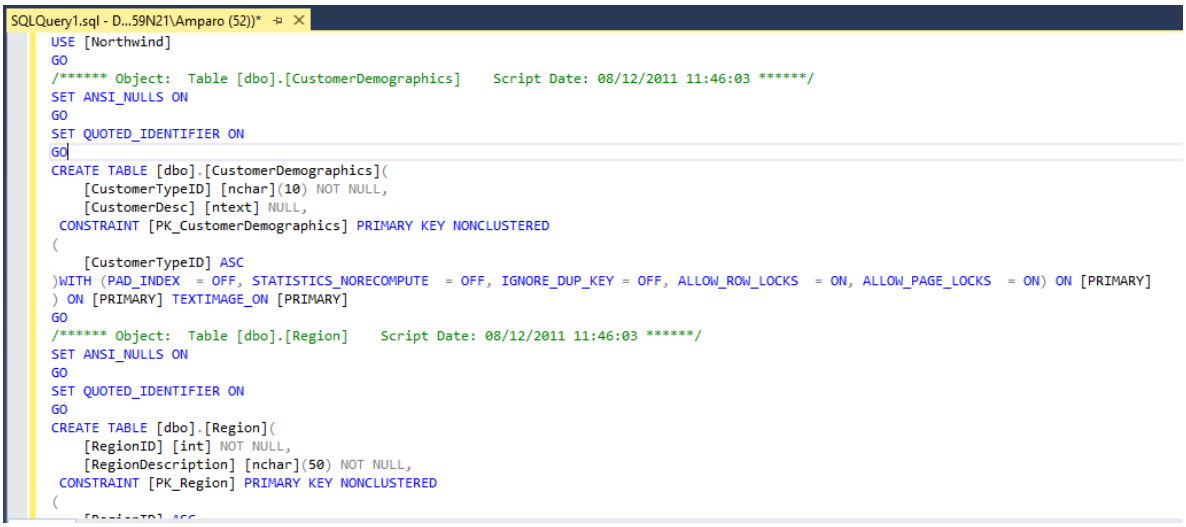
Necesitaremos tener instalados el servidor de bases de datos SQL Server Express y el entorno SQL Server Management Studio para administrar las bases de datos. SSMS (SQL Server Management Studio) proporciona herramientas para configurar, consultar, diseñar y administrar bases de datos y almacenes de datos, ya estén en el equipo local o en la nube.

En Microsoft SQL Server Management Studio:

1. Creamos una base de datos de nombre northwind en SQL SERVER.



2. En la base de datos creada, ejecutamos la sentencia SQL que contenga todas las sentencias de construcción de la bbdd, northwind.sql.



Con esto ya tenemos la bbdd creada. Podemos comprobarlo haciendo algún Select.

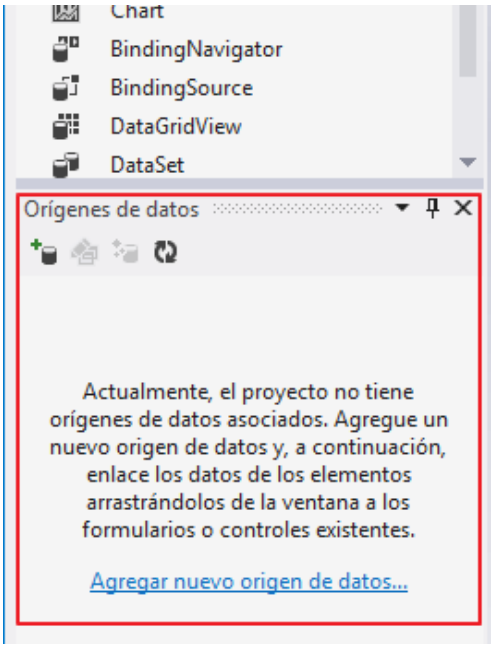
Ejemplo:

```
SELECT *
  FROM [Products]
```

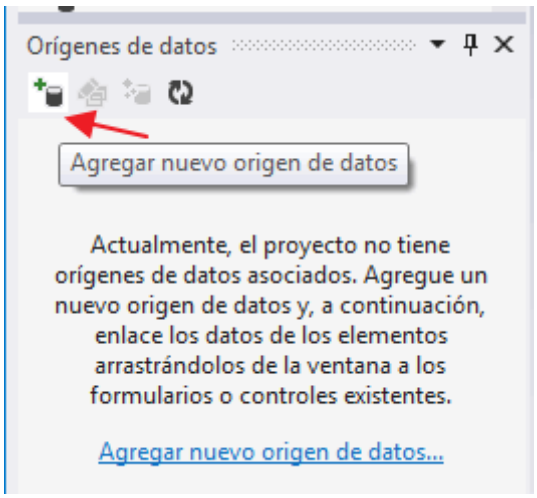
3. Nos fijamos en las propiedades de la base de datos Northwind en Server Name o Server "*SERVER*\SQLEXPRESS" porque lo tendremos que utilizar para la conexión desde Visual Studio. Para verlas, sobre la bbdd botón derecho → Propiedades.

3.1. Conexión de la base de datos

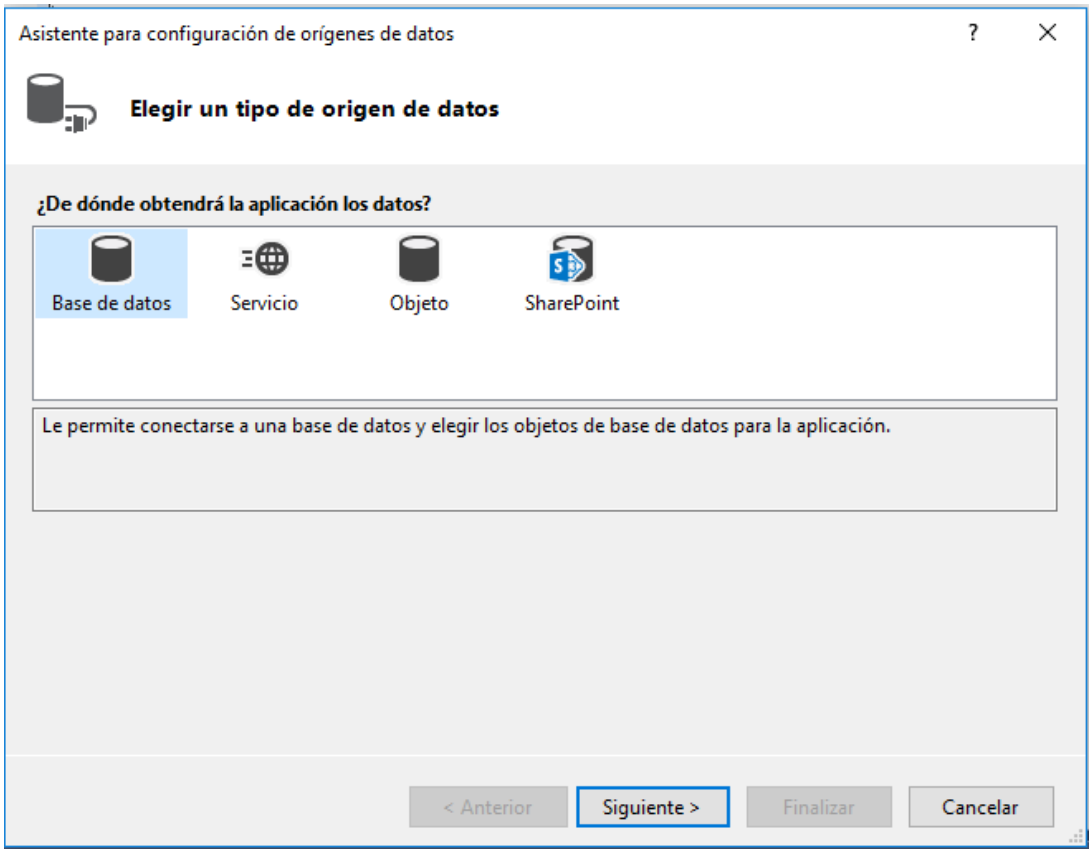
Vamos a establecer una primera conexión para explorar una base de datos de prueba. Creamos una nueva aplicación, y haremos clic "orígenes de datos" que tenemos como segunda solapa del cuadro de herramientas. Si no te aparece la activas en el menú "Ver":



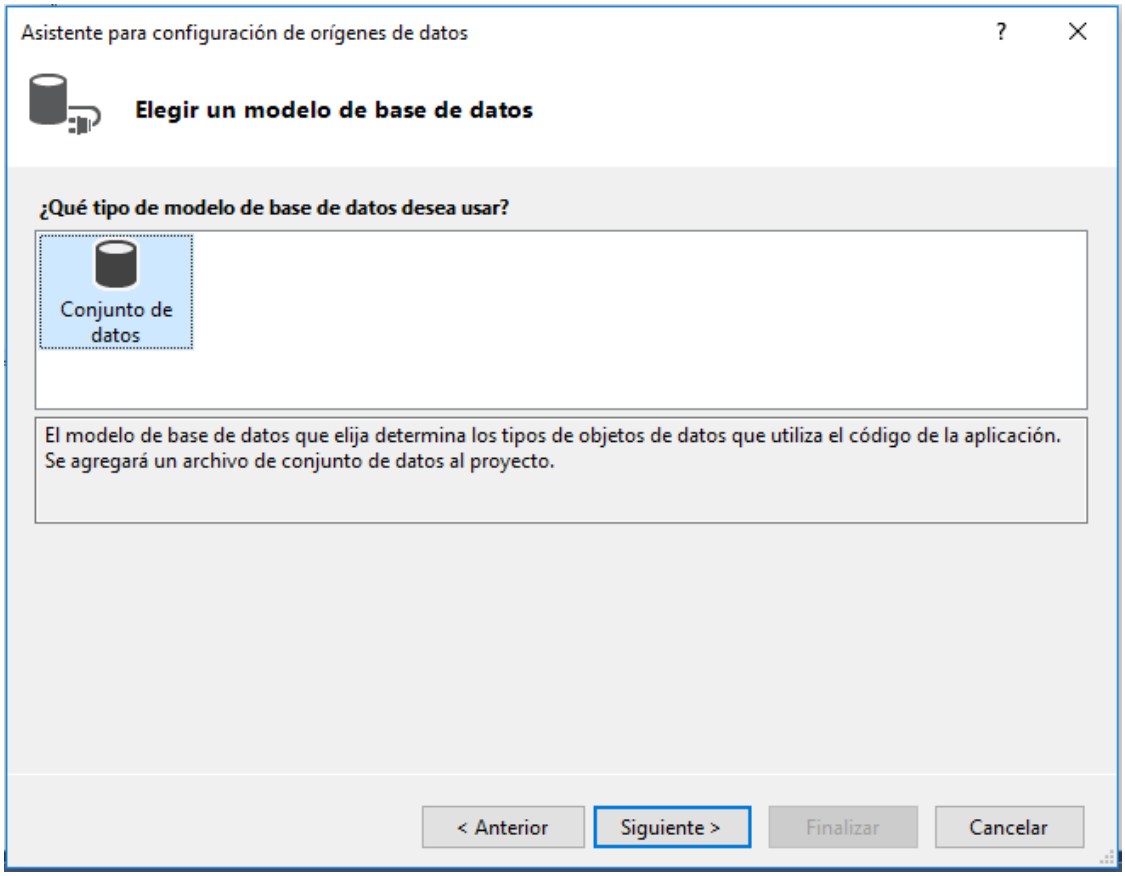
Hacemos clic en el enlace o en el primer icono de su barra de herramientas:



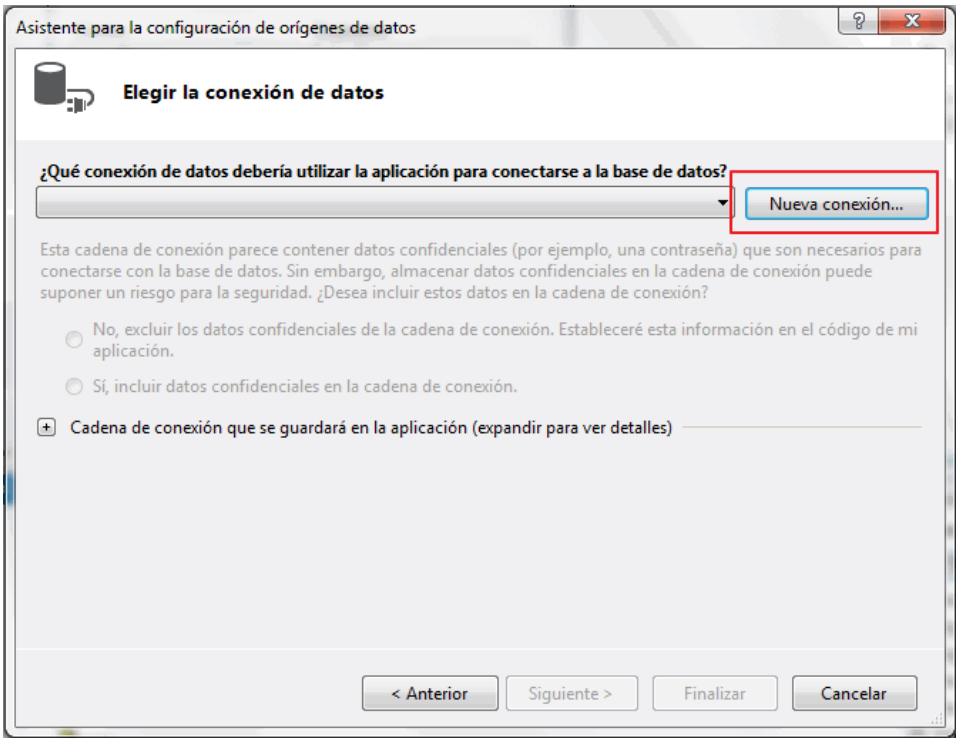
Le indicamos que va a ser una base de datos:



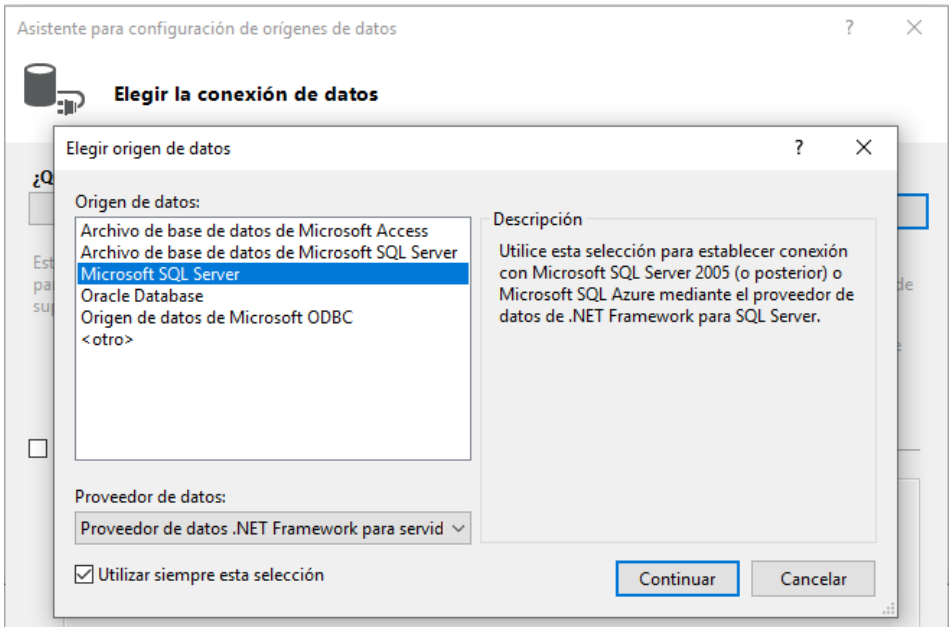
Nos preguntará ahora si queremos conectarnos con un origen de datos:



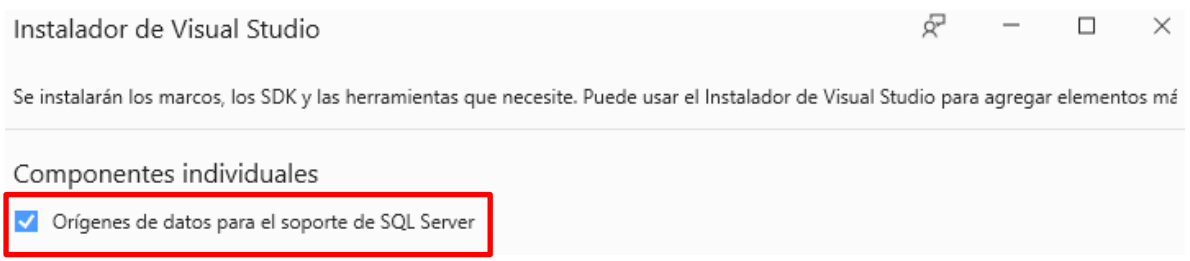
Lo seleccionamos y pulsamos en "Siguiente"



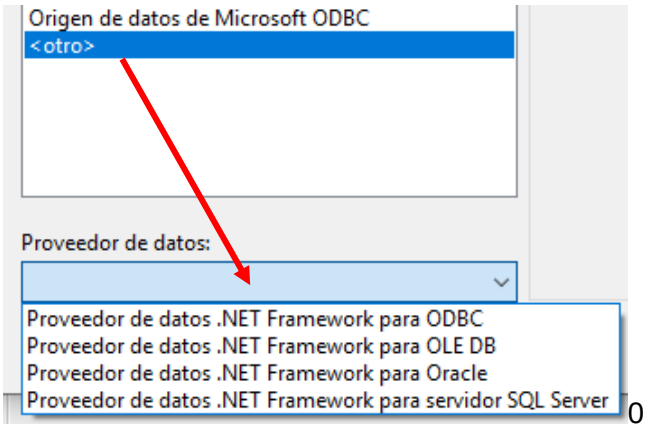
Vamos a crear una nueva conexión de datos, así que pulsamos en el botón de "Nueva conexión":



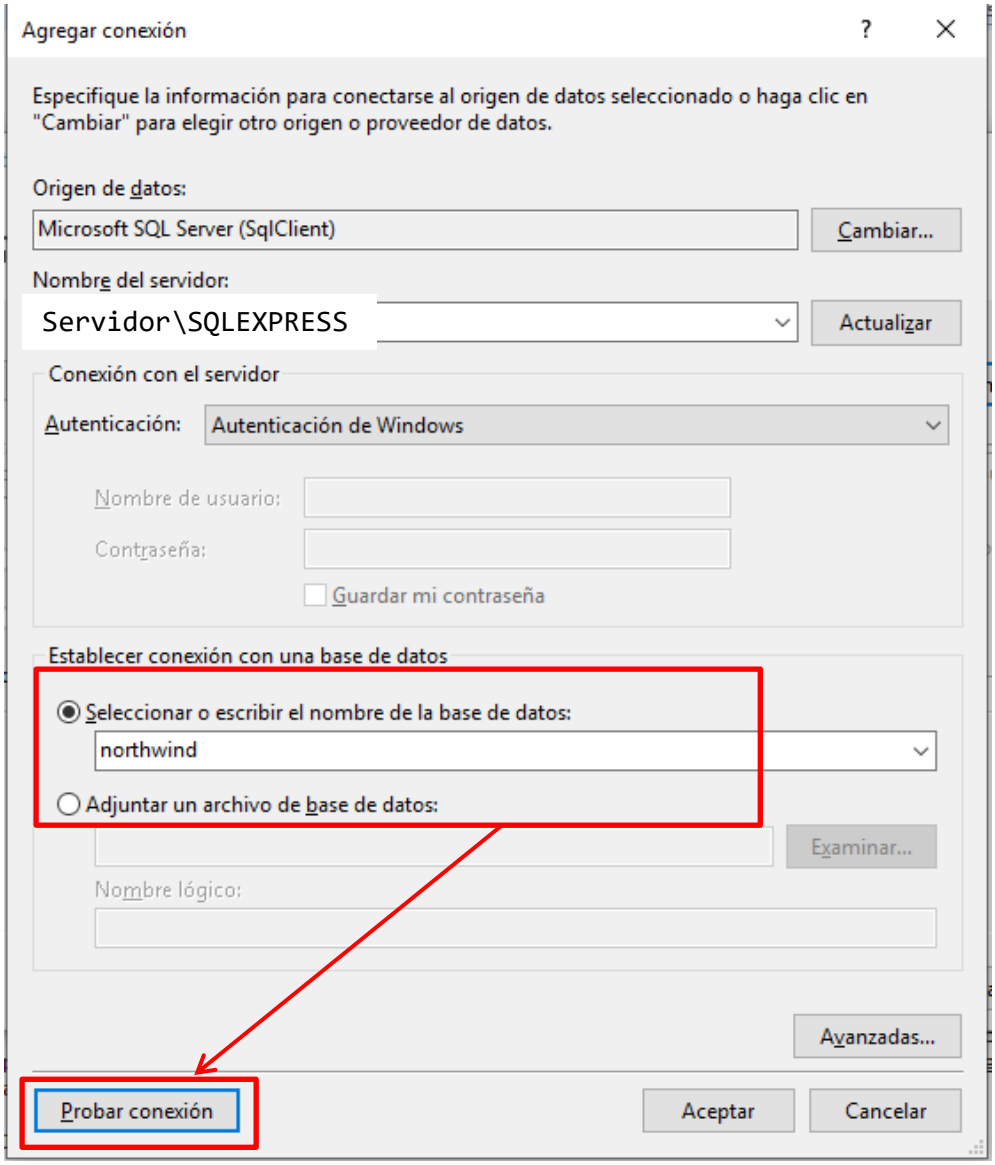
Al realizar esta acción nos indicará que necesitamos instalar el siguiente paquete para trabajar con SQL Server: “Orígenes de datos para el soporte de SQL Server”



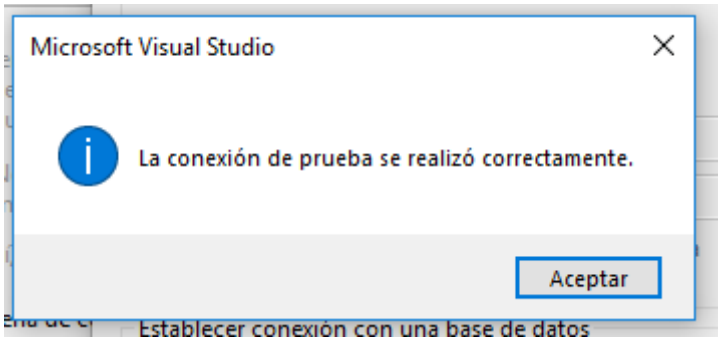
Seleccionamos el origen de datos, que va a ser "Microsoft SQL Server". Si pulsamos "Otro" podemos ver los otros proveedores:



Pero de momento nos quedamos con el "SQL". Al continuar nos mostrará la pantalla de configuración de la conexión:

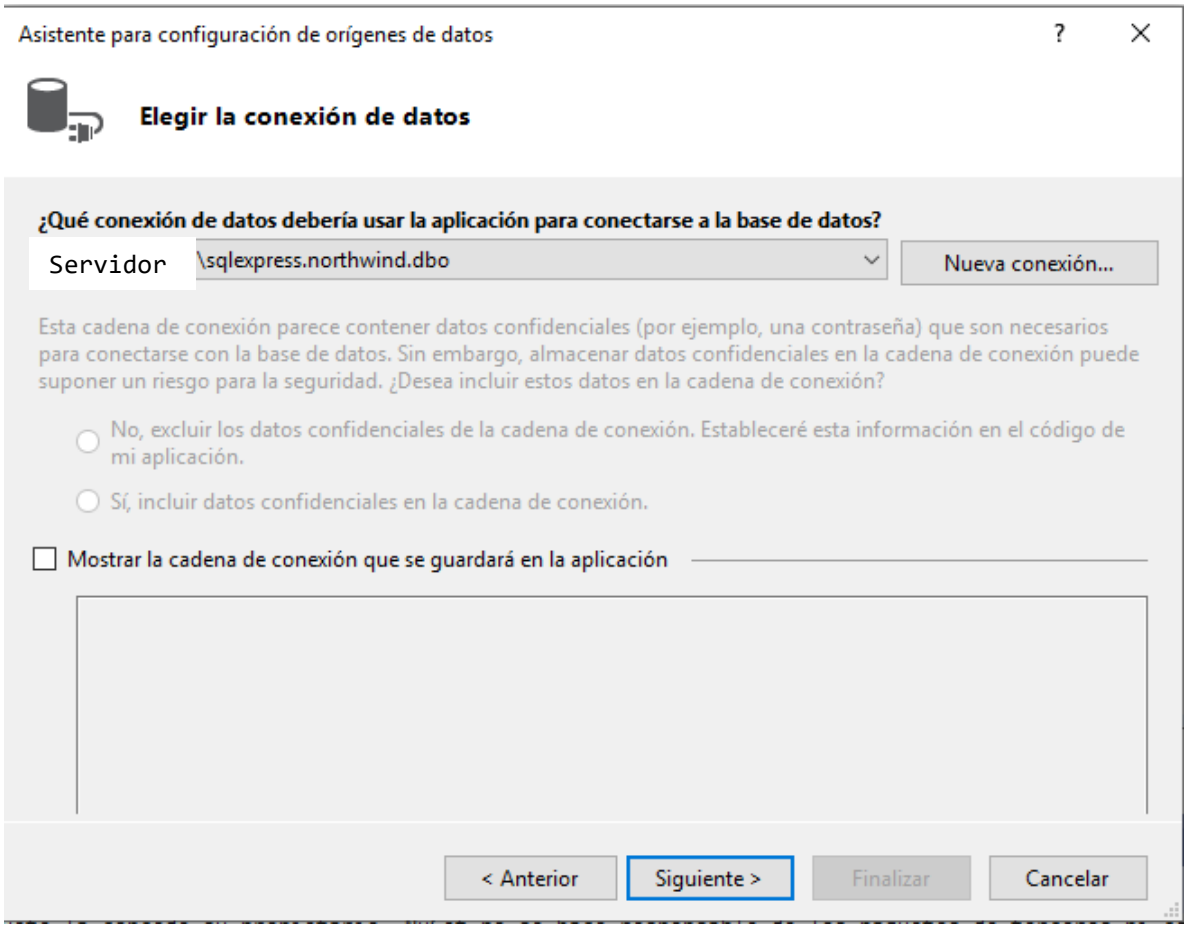


Pulsamos en "Probar Conexión" para que nos indique si el proceso ha ido bien:

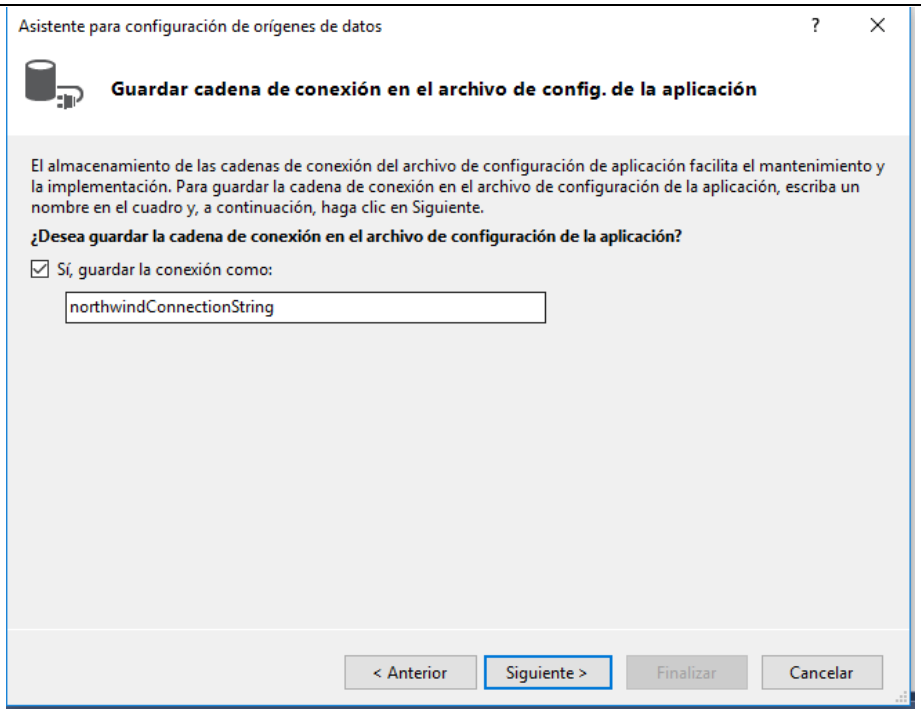


Pulsamos ahora en "Aceptar".

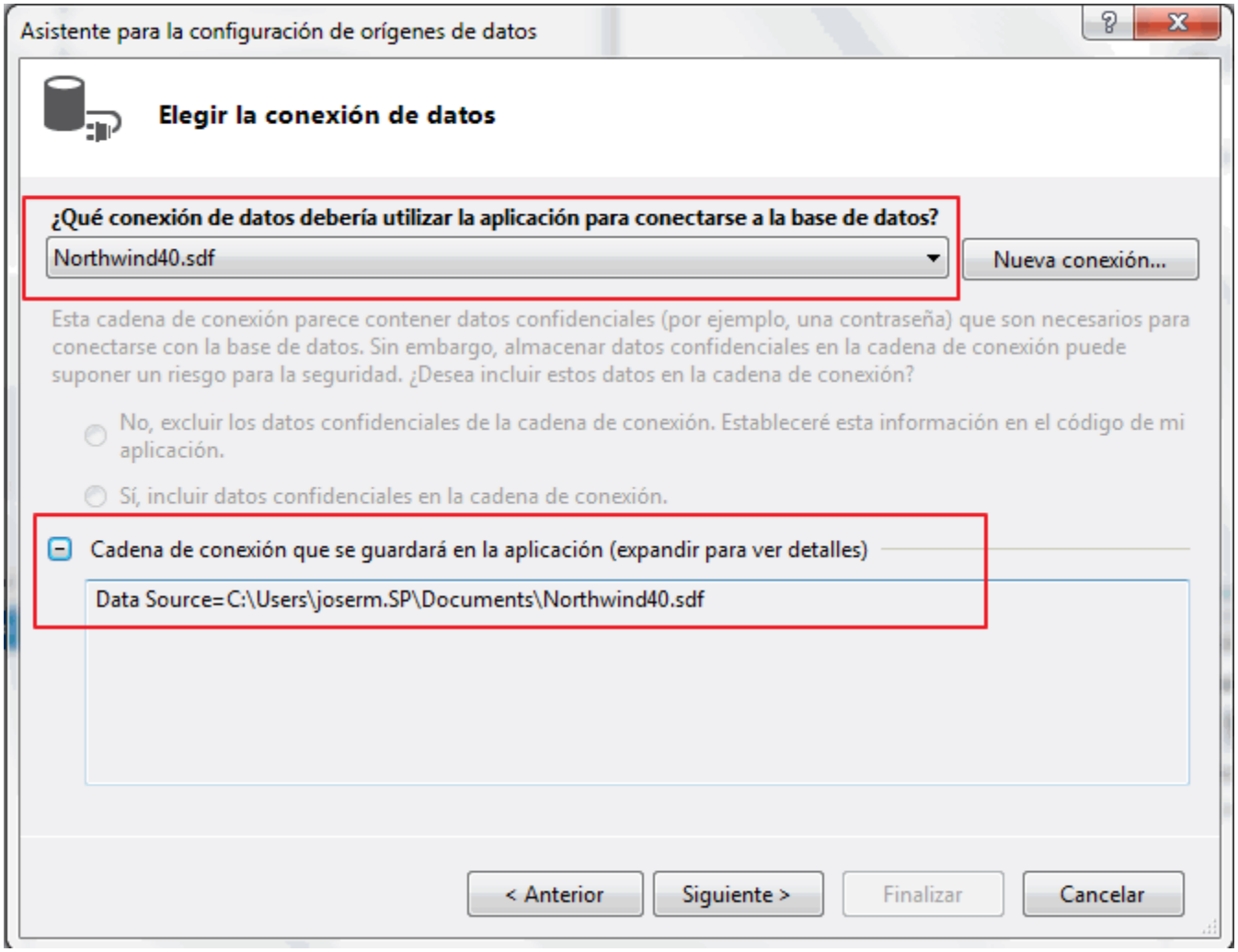
De vuelta a la ventana anterior, pulsamos en “Aceptar” para adjuntar esta base de datos y llegamos a:



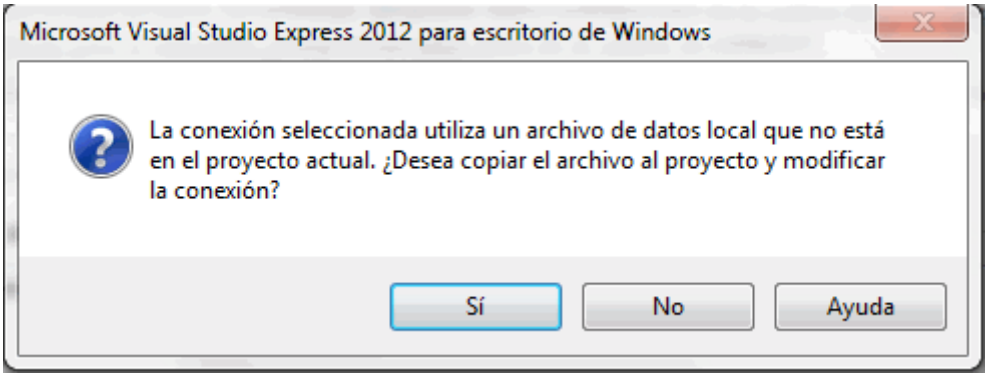
Si marcamos el checkbox *Mostrar la cadena de conexión que se guardará en la aplicación,* veremos la cadena de conexión a la base de datos que se almacenará en la aplicación.



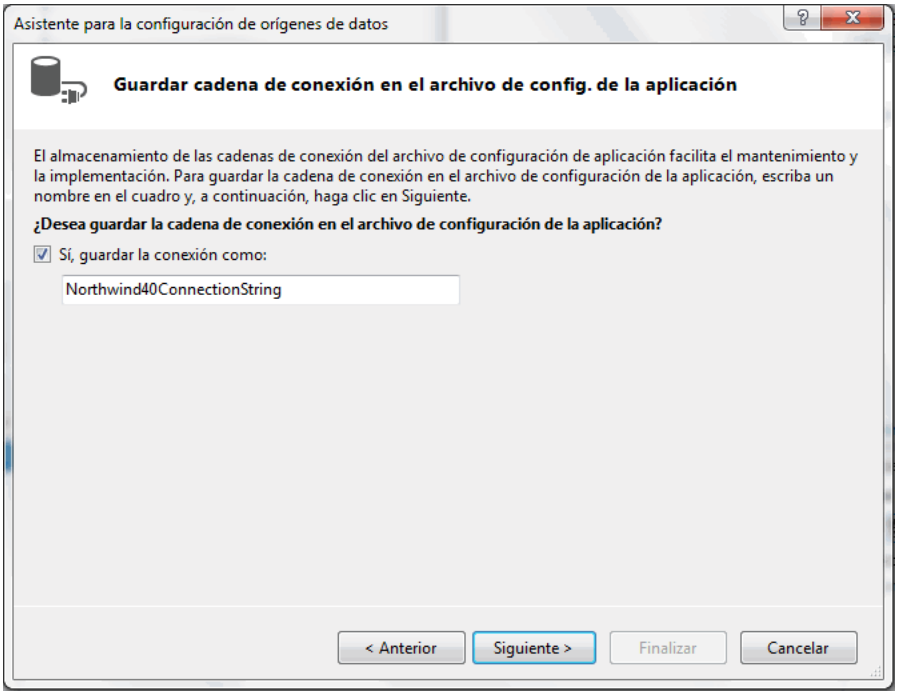
Si el tipo de conexión es otro llegaremos a una pantalla similar a ésta:



Pulsa ahora en el "+" de la parte de abajo donde tenemos la cadena de conexión necesaria para conectarnos a esta base de datos. Ahí está la parte importante de esto que estamos aprendiendo ya que a los controles les diremos que utilicen esta cadena de conexión que ahora asociaremos con un nombre. Pulsa en "Siguiente":

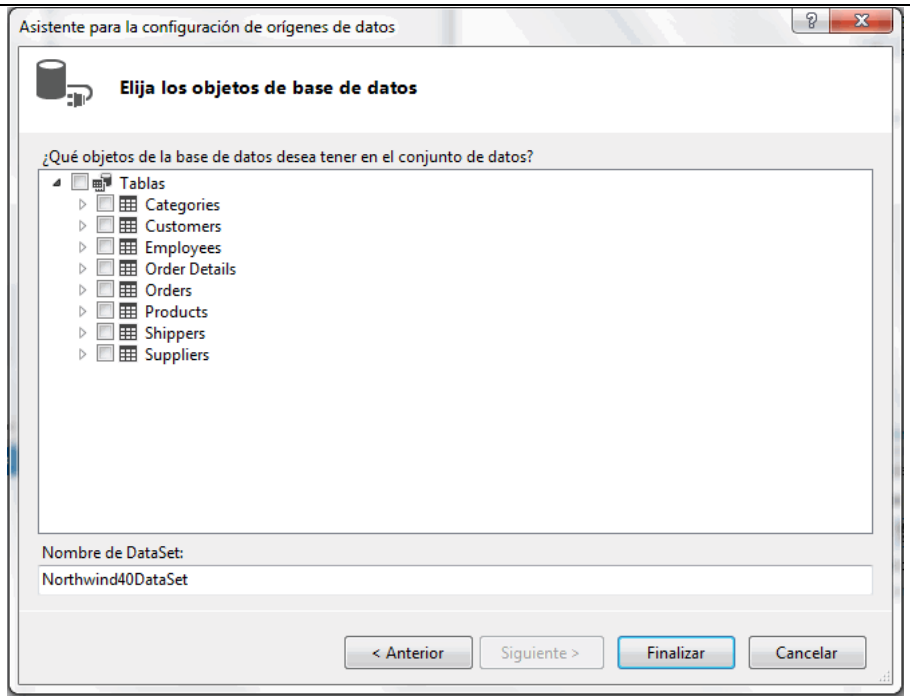


Y nos pregunta si queremos anexar esta base de datos a nuestro proyecto, le diremos que sí:

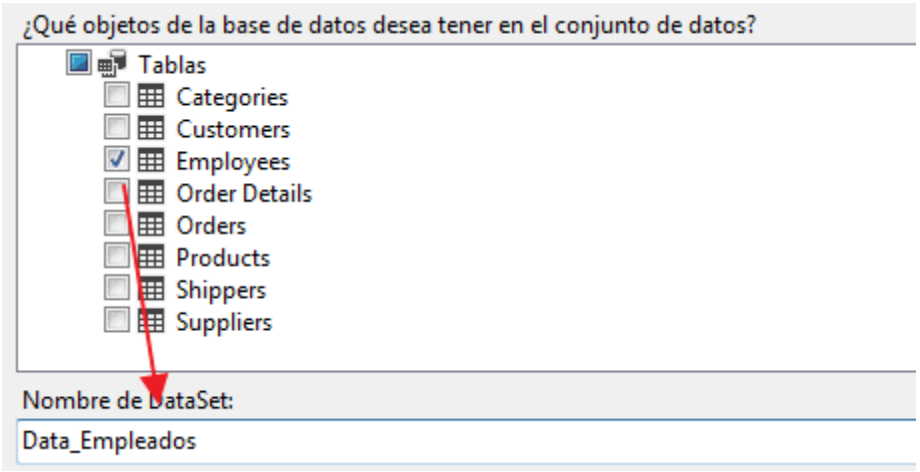


Le asociaremos un nombre más legible para hacer referencia en el código que hagamos a él en lugar de la cadena anterior. El IDE nos va a guardar en la configuración de nuestro proyecto un enlace a esta base de datos, llamando a este enlace con el nombre que le hayamos puesto en esa pantalla.

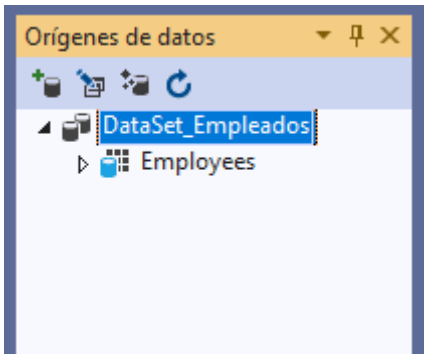
Pulsamos siguiente para ver ya el contenido de esa conexión de base de datos:



Vamos a marcar la tabla "Employees" y le ponemos un nombre a este "Dataset":



Pulsamos en Finalizar y ya tenemos todo listo:



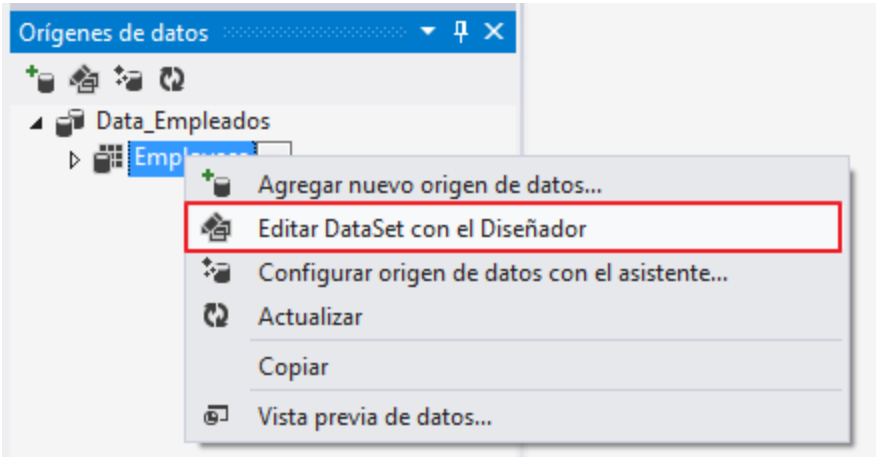
Recapitulemos:

1. Hemos creado una conexión de una base de datos de tipo SQL Server.
2. La hemos vinculado con "northwind".

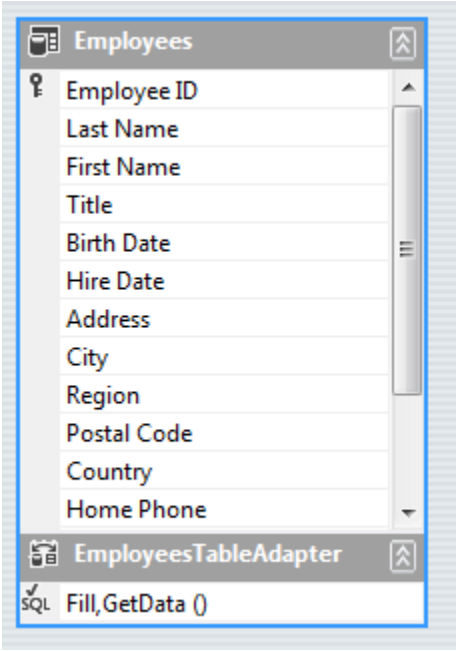
3. Ha creado una cadena de conexión para apuntar a esta definición.
4. Hemos abierto la base de datos y de todas las tablas hemos seleccionado "Employees"
5. Finalmente hemos asociado el nombre "DataSet_Empleados" con el DataSet o conjunto de resultados "Employees"

Con estas operaciones, tenemos un conjunto de resultados asociado al proyecto.

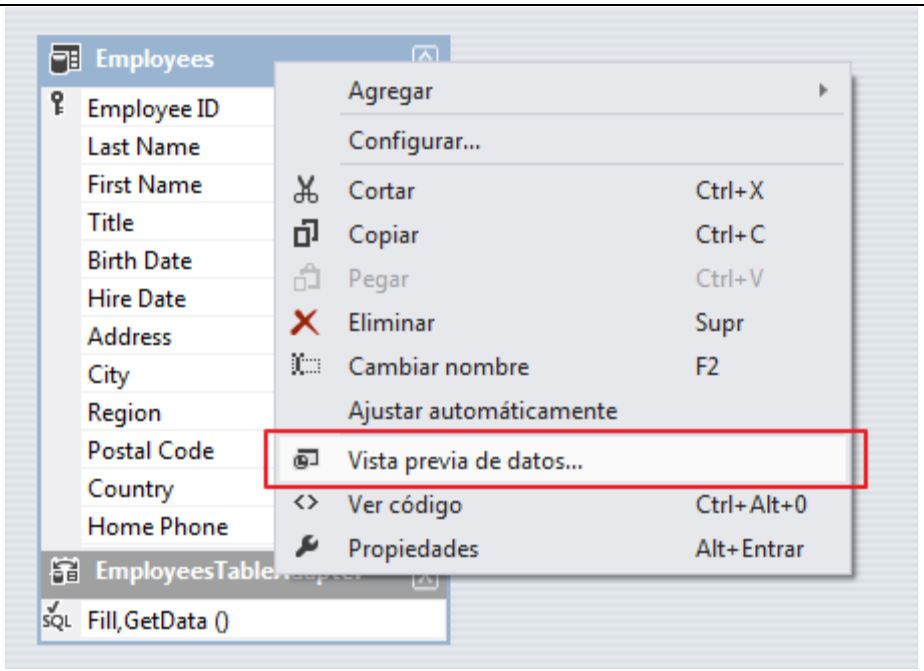
Sigamos viendo más detalles de esta pantalla. Pulsa en el nombre de la vista que hemos creado y seleccionamos:



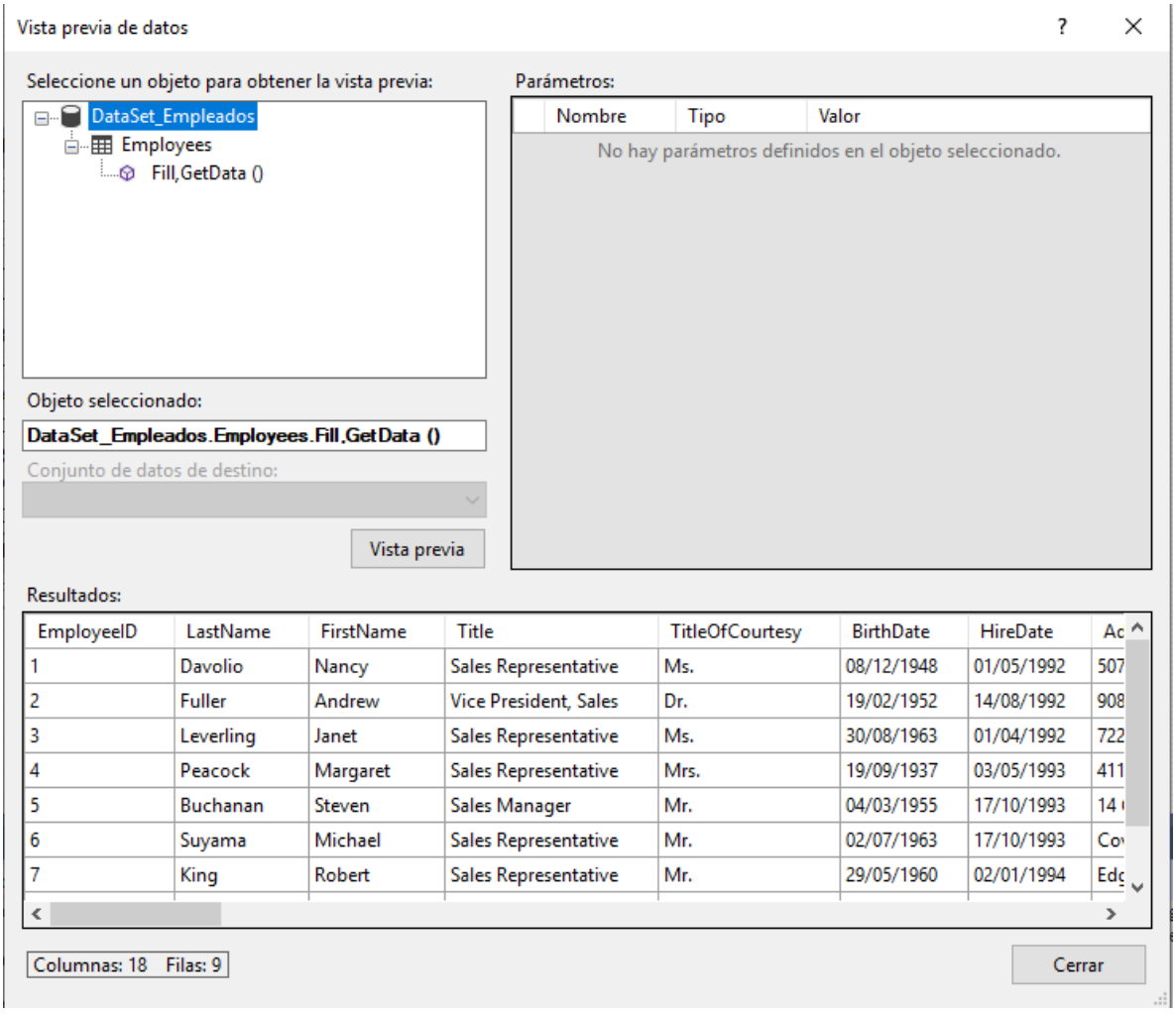
Podremos ver la tabla que hemos añadido a nuestro conjunto de datos, si hubiéramos seleccionado más estarían aquí:



Ahora, haciendo clic con el botón derecho encima de la tabla pulsamos en "Vista Previa de datos":

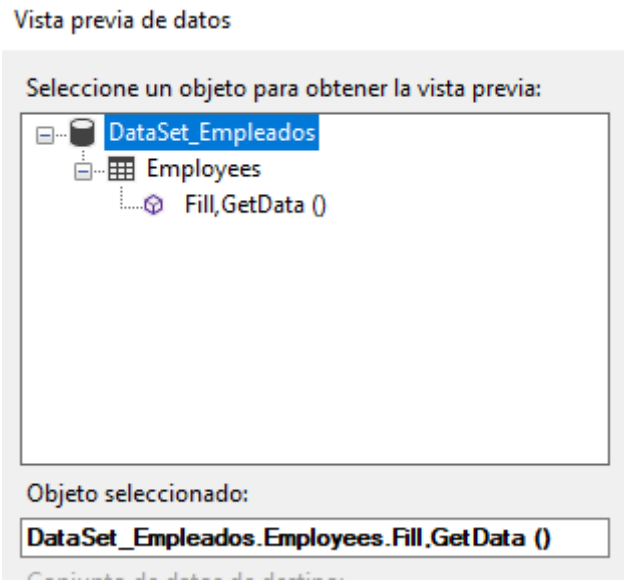


Nos mostrará los datos que contiene esta tabla:



Nos ha mostrado el contenido de la tabla.

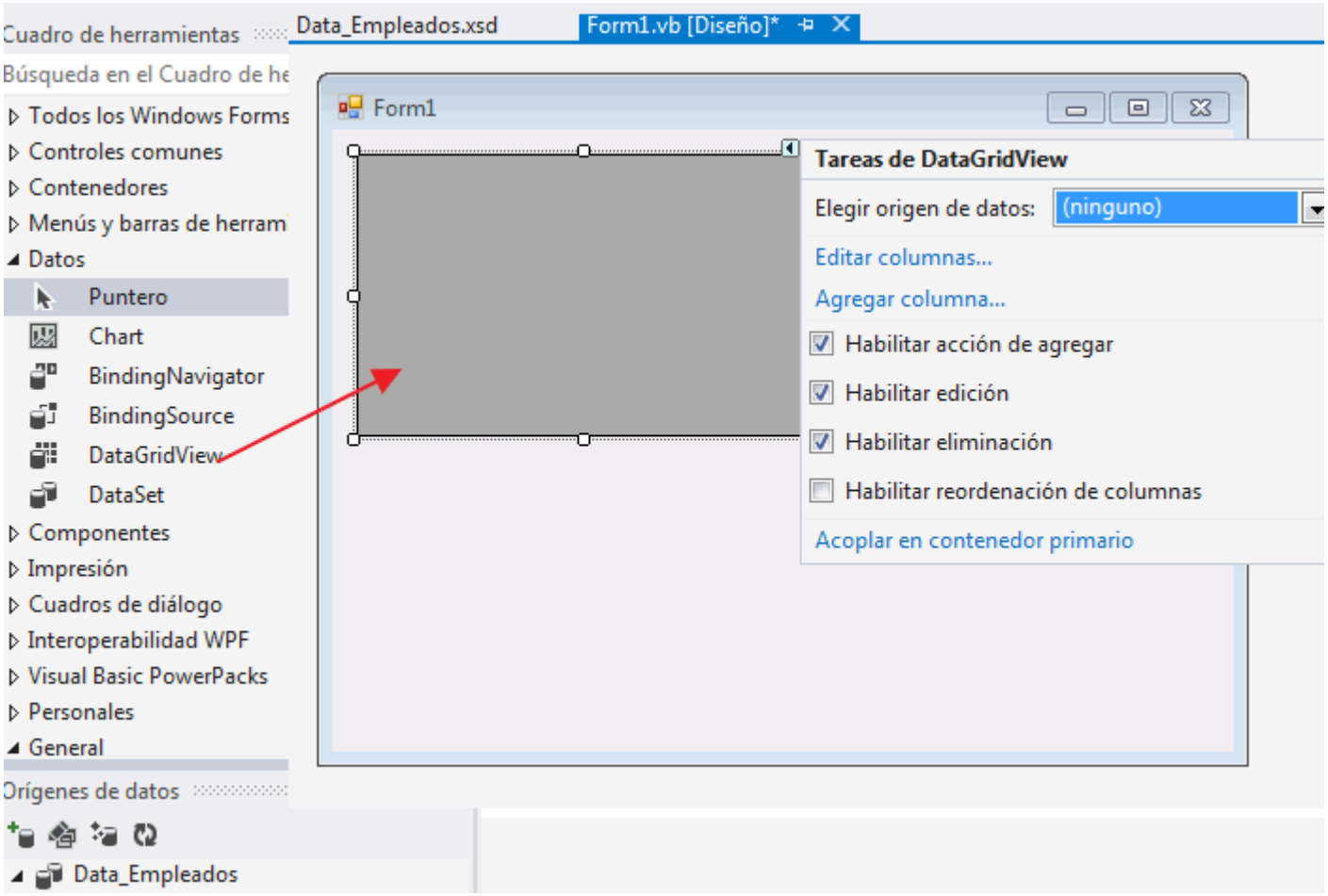
Ahora nos fijamos en el cuadro de arriba a la izquierda:



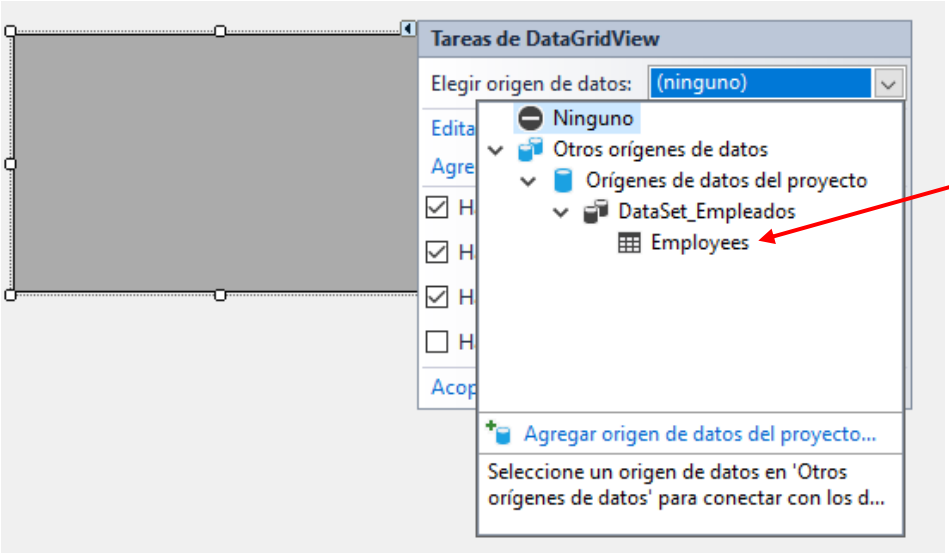
Nos muestra la forma con que ha recuperado los datos que aprenderemos luego. Parece que ese "DataSet" que hemos llamado "DataSet_Empleados" tiene dentro una tabla "Employees" que ejecutando el método "Fill, GetData" nos ha permitido recuperar los datos. Bien, poco a poco: cadena de conexión, conexión y recuperar datos.

3.2. Conectar la conexión de la base de datos con un control

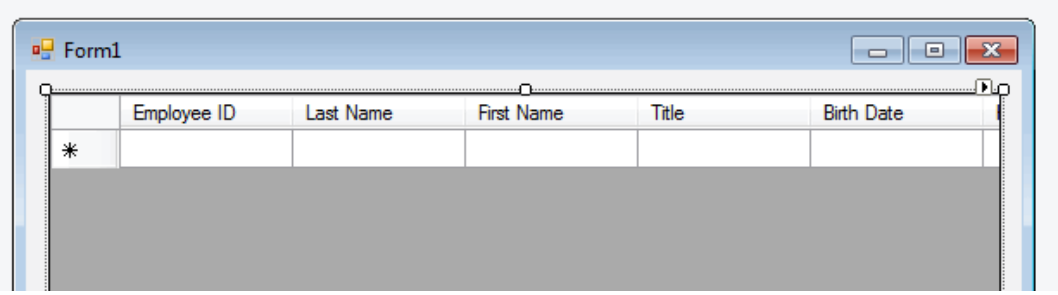
Volvamos a nuestro programa y vamos a añadir un control del tipo "DataGridView" situado en la sección de control de acceso a datos, que tiene estas opciones en su menú de propiedades más comunes:



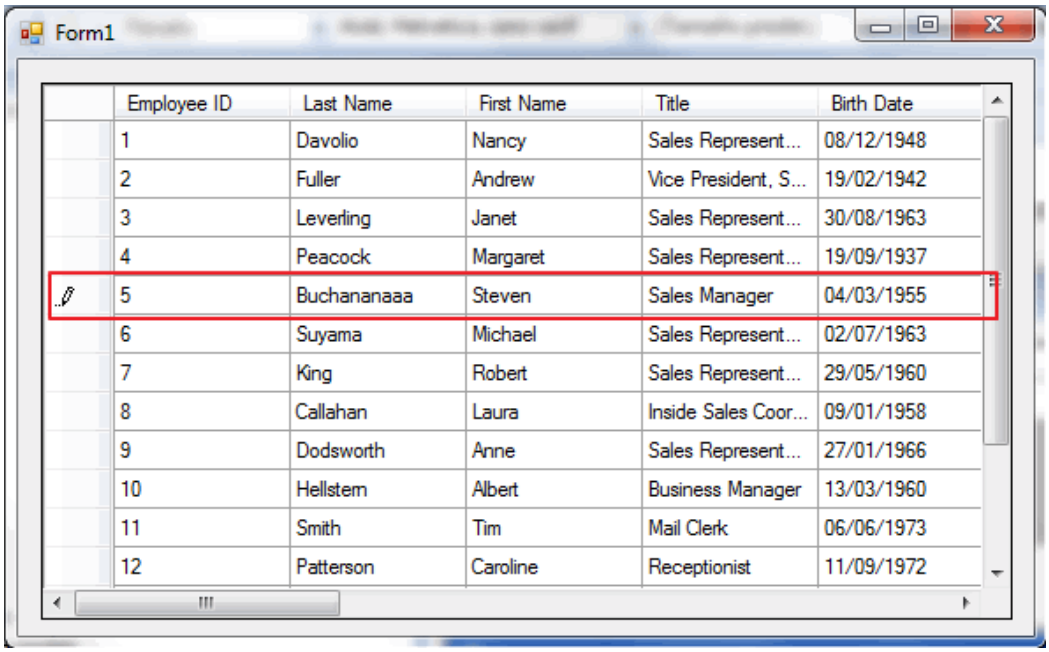
Pulsamos en el origen de datos y seleccionamos:



Veremos cómo cambia el aspecto de la cuadrícula colocándose todas las columnas de la tabla de la base de datos que hemos seleccionado:

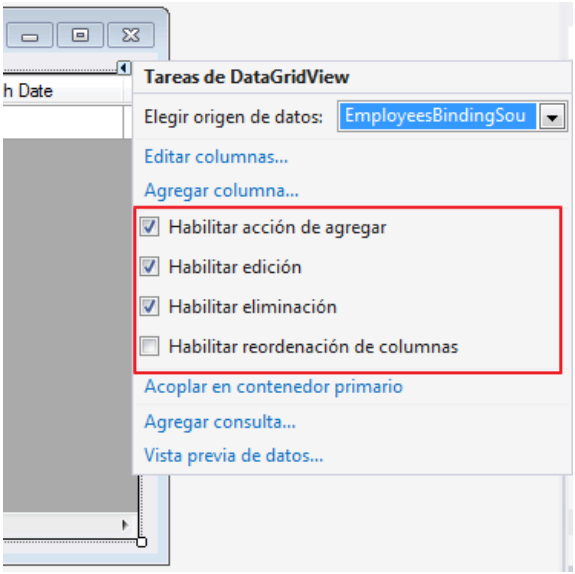


Vamos a ejecutar el programa:

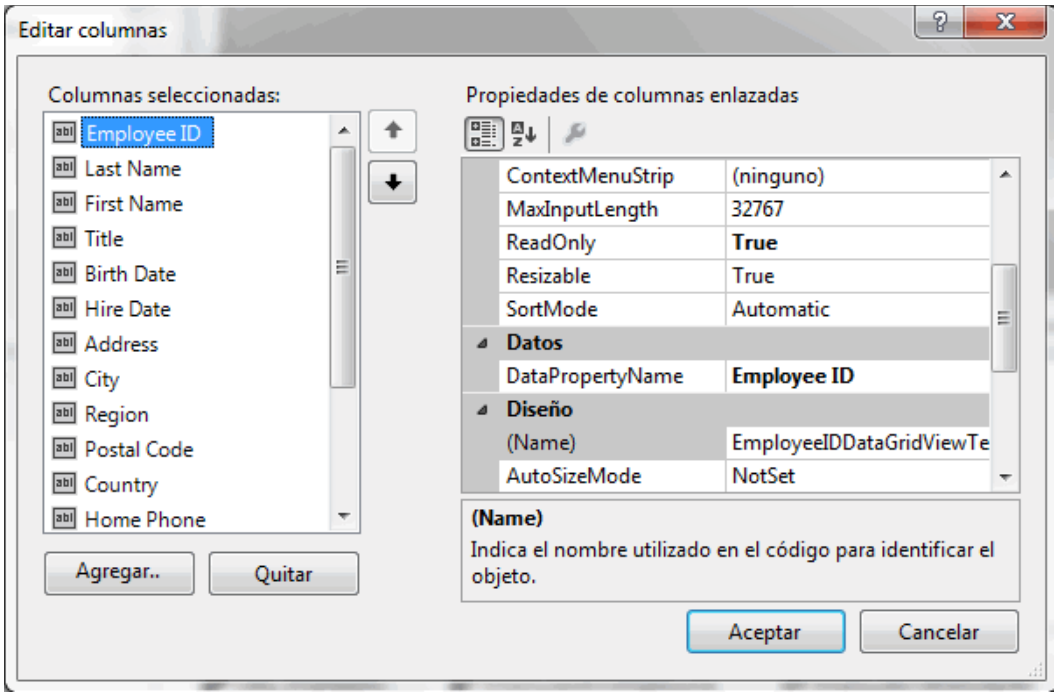


Employee ID	Last Name	First Name	Title	Birth Date
1	Davolio	Nancy	Sales Represent...	08/12/1948
2	Fuller	Andrew	Vice President, S...	19/02/1942
3	Leverling	Janet	Sales Represent...	30/08/1963
4	Peacock	Margaret	Sales Represent...	19/09/1937
5	Buchananaaa	Steven	Sales Manager	04/03/1955
6	Suyama	Michael	Sales Represent...	02/07/1963
7	King	Robert	Sales Represent...	29/05/1960
8	Callahan	Laura	Inside Sales Coor...	09/01/1958
9	Dodsworth	Anne	Sales Represent...	27/01/1966
10	Hellstern	Albert	Business Manager	13/03/1960
11	Smith	Tim	Mail Clerk	06/06/1973
12	Patterson	Caroline	Receptionist	11/09/1972

Y además tenemos la posibilidad de añadir y modificar datos. Esto es porque al hacer el enlace estaban activadas las opciones que permiten estas operaciones:



Si pulsamos en la opción de "Editar columnas" podemos ver cómo configurar todos los aspectos relacionados con la presentación visual:



Con todas estas opciones podemos ocultar columnas, poner títulos a los encabezados, ancho de las columnas, activar solo lectura para algunos campos...

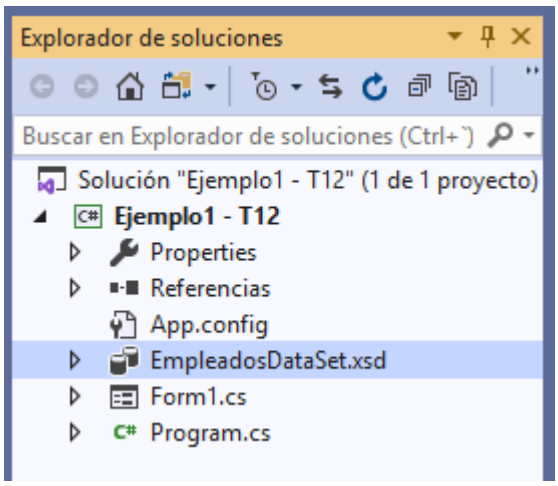
Para terminar, veamos el único código que nos ha puesto para hacer esta carga al iniciarse el programa:

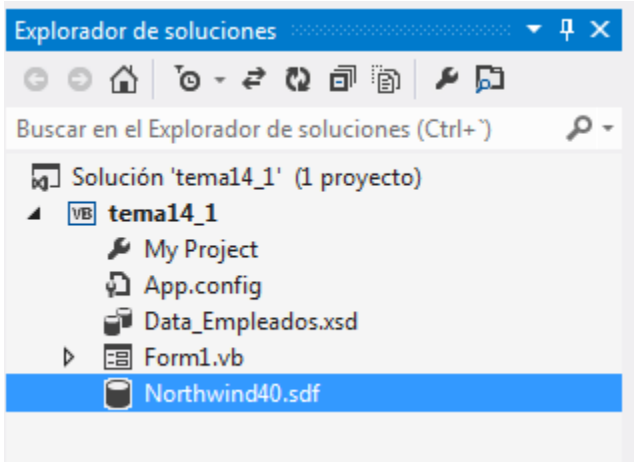
```
public partial class Form1 : Form
{
    1 referencia
    public Form1()
    {
        InitializeComponent();
    }

    1 referencia
    private void Form1_Load(object sender, EventArgs e)
    {
        // TODO: esta línea de código carga datos en la tabla 'empleadosDataSet.Employees'
        // Puede moverla o quitarla según sea necesario.
        this.employeesTableAdapter.Fill(this.empleadosDataSet.Employees);

    }
}
```

En la carga del programa (Form1_Load), rellena (Fill) el adaptador (employeesTableAdapter) con la tabla empleados (Employees) del conjunto de resultados llamado "empleadosDataSet".

3.3. ¿Dónde está la cadena de conexión?

La cadena de conexión es la que define dónde está el origen de datos. Esta definición alimentará los controles de acceso a datos (DataGrid, por ejemplo) para poder recuperar los valores. Todo esto ya la sabemos, pero ¿dónde ha quedado almacenada esta conexión?. Veamos qué tiene ahora nuestro proyecto:





Si la conexión ha sido a una base de datos SQL Server tendremos la situación de la izquierda. A la configuración habitual del proyecto se ha añadido la definición del conjunto de resultados o DataSet llamado EmpleadosDataSet.xsd.

Sin embargo, si la conexión ha sido a otro tipo de base de datos puede que tengamos la configuración de la derecha. Por un lado, el fichero de base de datos que hemos incorporado a nuestro proyecto y, por otro lado, una definición de un conjunto de resultados o DataSet llamado "DataSet_Empleados.xsd". Ahora veamos el contenido del fichero "App.config":

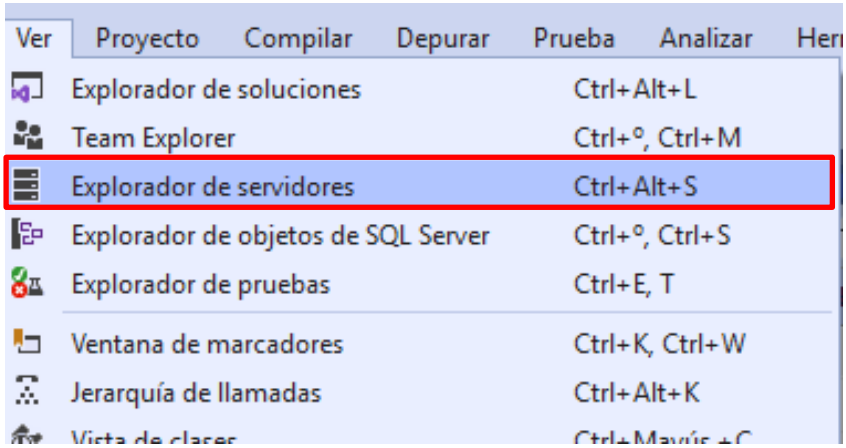
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="Conexion.My.MySettings.northwindConnectionString"
            connectionString="Data Source=SERVIDOR\SQLEXPRESS;Initial Catalog=northwind;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>
</configuration>
```

Dentro de este fichero de configuración podemos ver la sección donde tenemos "connectionStrings". De esta forma podemos utilizar la cadena de conexión que hemos creado con el enlace a esta base de datos para todo el proyecto entero.

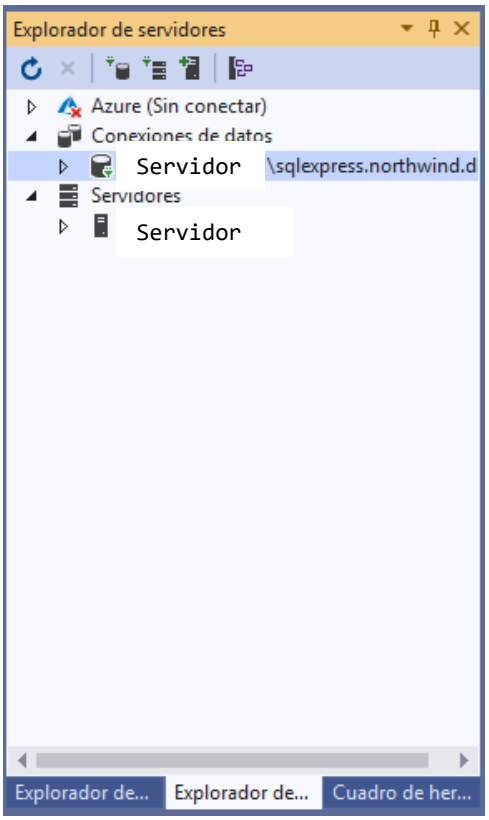
4. SQL y las tablas en las bases de datos

El lenguaje SQL es el lenguaje para realizar el mantenimiento de la base de datos. Este mantenimiento consta de las operaciones ya conocidas de: altas, bajas, consultas y modificaciones.

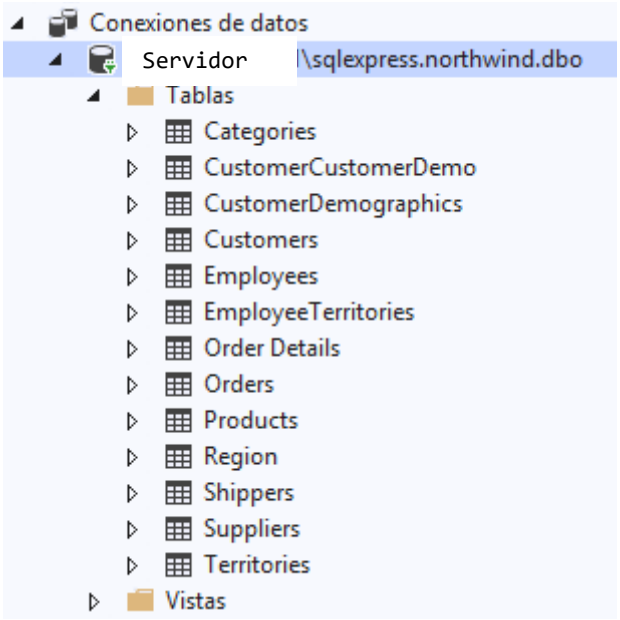
Vamos a activar ahora una vista de las conexiones a las bases de datos, seleccionaremos Ver en el menú y luego:

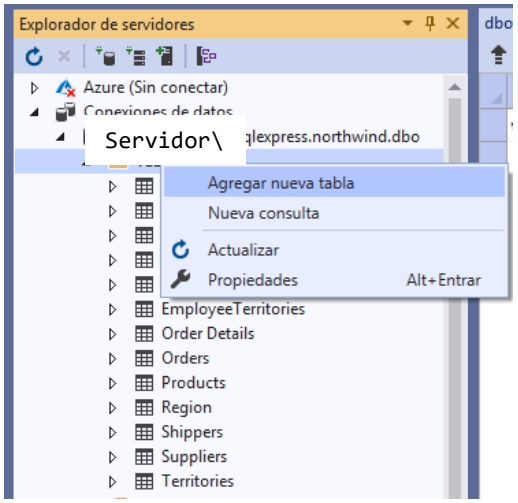


Nos mostrará una nueva solapa en la zona de los controles:



Si expandimos las tablas son todas las que contiene la base de datos a la que nos conectamos antes. Vamos a expandir las tablas y crear una nueva eligiendo la opción:





Nos mostrará el diseñador de tablas en el que podremos añadir los campos que queramos para nuestra tabla. Uno de esos campos debe ser una clave única para la tabla, es decir, debe haber un campo que sea único para cada fila. Por ejemplo, el código de empleado, el código de un artículo, el DNI de una persona.

También debemos pensar un poco en los tipos de datos que vamos a tener, por ejemplo:

- Text
- Number
- Date/Time
- Boolean
- Binary
- Otros, como es identificador único: uniqueidentifier, xml, timestamp, sql_variant.

Dentro de los textos nos encontramos con varios tipos que tienen sus peculiaridades:

- char (n). Texto de longitud "n" fija
- nchar(n). Texto de longitud "n" fija Unicode
- varchar(n). Texto de longitud variable con un valor máximo de "n" no unicode y de hasta 8.000 caracteres
- nvarchar(n). Texto de longitud variable con un valor máximo de "n" unicode y de hasta 4.000 caracteres

Unicode es el tipo de codificación habitual y es la que debemos utilizar es un formato mucho más amplio que el antiguo código ASCII.

En los numéricos tenemos dos tipos:

- Números enteros:
  - tiniyint: admite valores de 0 a 255
  - smallint: desde -32.768 hasta 32.767
  - int: desde -2.147.483.648 hasta 2.147.483.647
  - bigint: para valores mucho más grandes.
- Números en coma flotante, es decir, con decimales los vemos en la siguiente tabla

Otros tipos de datos son:

- Boolean (bit) admite un valor de 0 ó de 1
- smalldatetime: para las fechas/horas
- uniqueidentifier: define un campo que almacena un "Globally Unique Identifier", es una cadena de caracteres que hace de identificador único. Es un valor que no tiene significado pero que es único así que se puede utilizar para relacionar las tablas o como índice de la tabla.

Veamos ahora un resumen de los tipos de datos en SQL Server:

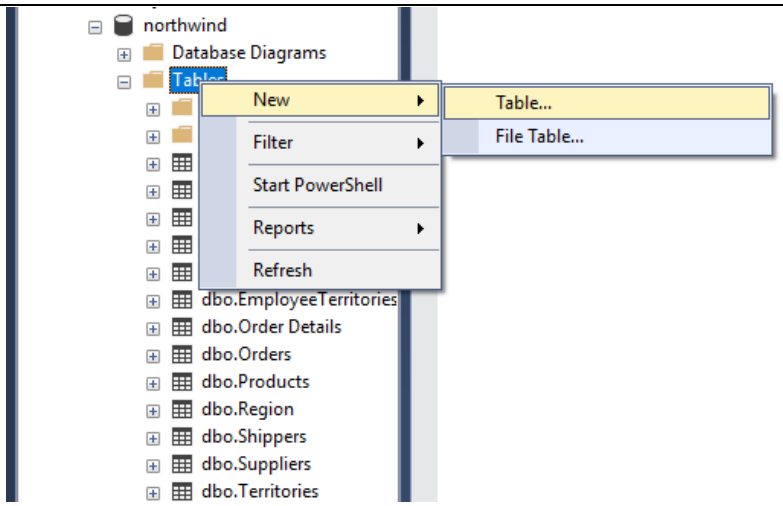
Tipo de datos	Detalles	Espacio
bigint	enteros largos -2^{63} hasta 2^{63}	8 bytes
binary	binario fijo hasta 8000	n bytes
bit	0, 1 ó null	1 byte para 8 campos de bit
char (n)	texto fijo hasta 8.000	n bytes
datetime	desde 1/1/1753 hasta 31/12/9999	8 bytes

decimal (p,s)	números -10^38 +1 hasta 10^38 -1	según p
float (n)	números desde -1,79E+38 hasta -1,79E-38	según n
image	variable hasta 2^31	longitud + 2 bytes
int	enteros desde -2.147.483.648 hasta 2.147.483.648	4 bytes
money	cantidad para moneda	8 bytes
nchar(n)	cadena de longitud fija unicode de hasta 4.000 caracteres	2 * longitud
ntext	cadena de longitud variable de hasta 2^30	2 * longitud bytes
numeric (p,2)	números desde -10^38 hasta 10^38	Depende de p
nvarchar (n)	Cadena de longitud variable unicode de hasta 4.000 caracteres	2 * longitud + 2 bytes
nvarchar (MAX)	Cadena de longitud variable unicode de hasta 2^31 bytes	2 * longitud + 2 bytes
real	-1.18E-38 and 1,18E-38 hasta 3,40E+38	4 bytes
smalldatetime	fechas desde 1/1/1900 hasta 6/16/2079	4 bytes
smallint	enteros desde -32768 hasta 32767	2 bytes
smallmoney	-214748,3648 hasta 214748,3648	4 bytes
sql_variant	hasta 8016 bytes	depende del almacenamiento de datos
text	texto de longitud variable de hasta 2^31 caracteres	depende de la página de códigos del servidor
timestamp	hora del último cambio de la fila	8 bytes
tinyint	enteros de 0 a 255	1 byte
uniqueidentifier	Global Unique Indentifiers	16 bytes
varbinary(n)	datos binarios de longitud variable hasta 8000	longitud + 2 bytes
varbinary(MAX)	datos binarios de longitud variable hasta 2^31	longitud + 2 bytes
varchar(n)	texto de longitud variable hasta 8000 caracteres	n+2 bytes
varchar(MAX)	texto de longitud variable hasta 2^31 caracteres	longitud + 2 bytes
xml	tipo de datos XML	depende de los datos almacenados

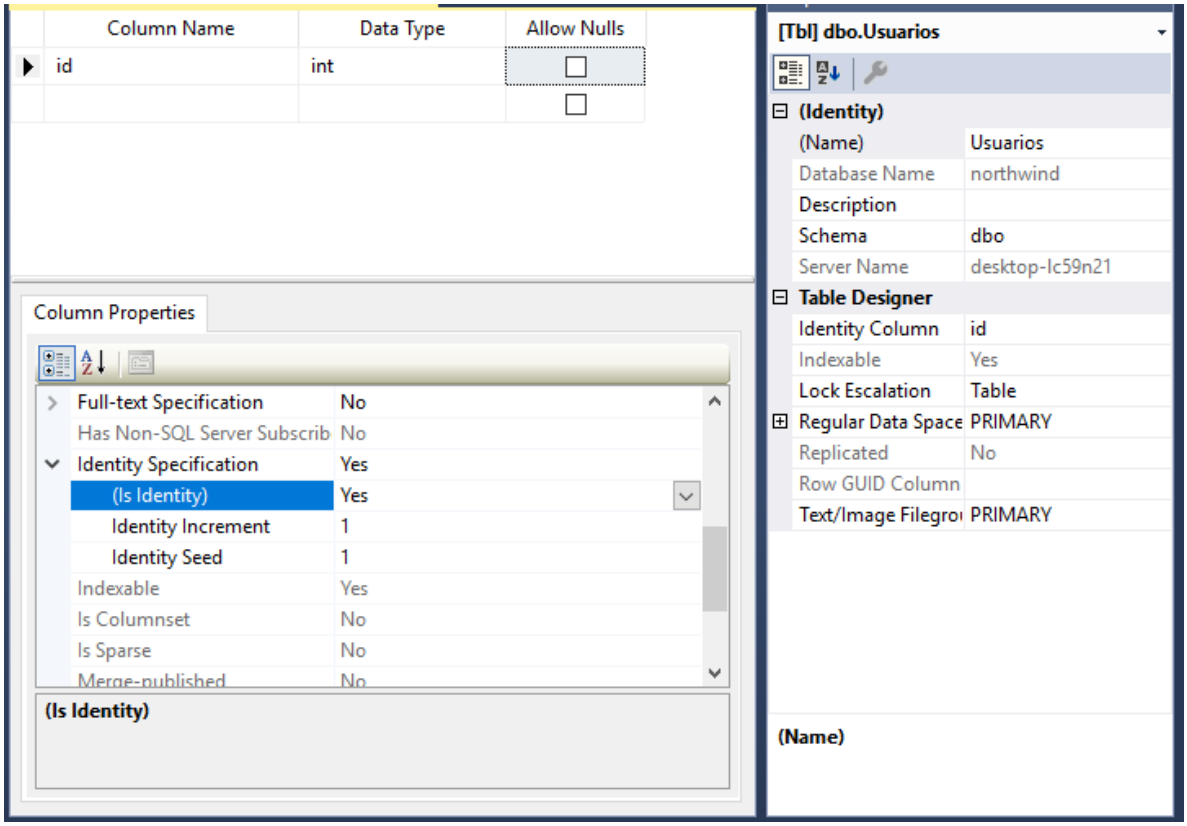
Si continuamos en este entorno para crear las tablas necesitamos conocer más sobre T-SQL.

Para poder crear nuestras tablas y trabajar sobre la base de datos de un modo gráfico más sencillo lo haremos sobre **SQL Server Management Studio**.

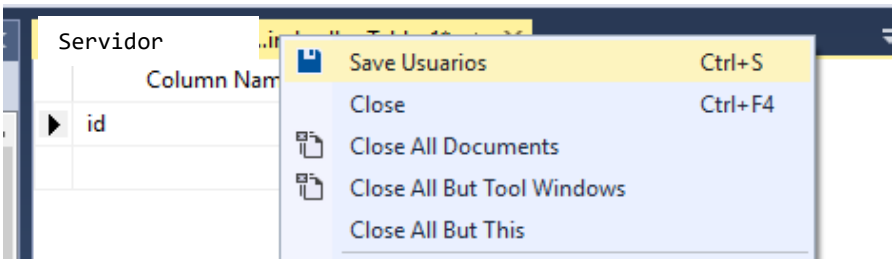
Creamos una nueva eligiendo la opción:

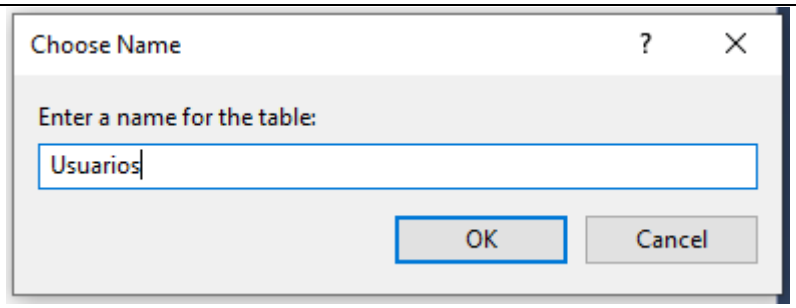


Vamos a introducir un campo como el siguiente:



Le ponemos nombre a la tabla así:





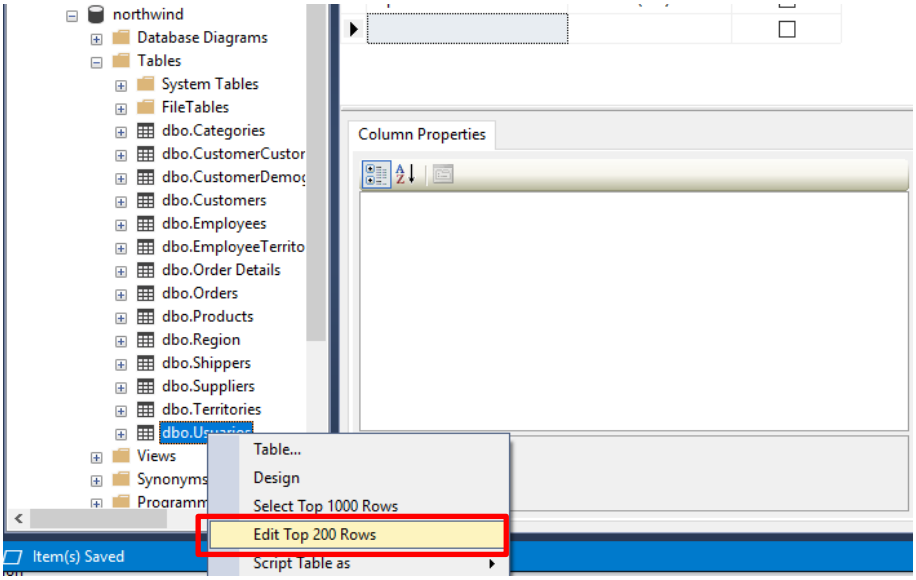
Primero hemos puesto arriba un nombre a esta tabla, "Usuarios", y le hemos añadido un campo que se llama "id" de tipo entero "Int" y luego he marcado debajo que es de tipo "Identity". Esto significa que va a ser un valor autonumérico. Este tipo de característica la "identity" hace que ese campo se rellene automáticamente con un valor numérico cada vez que insertemos una fila. Este valor comienza por el 1 y se incrementa de uno en uno, tal y como vemos en la pantalla

De esta forma siempre tendremos un valor único en mi tabla que podré utilizar como clave. Luego veremos todo esto junto, de momento nos quedamos con la idea. Ahora añadiremos dos campos más: Nombre y Apellidos:

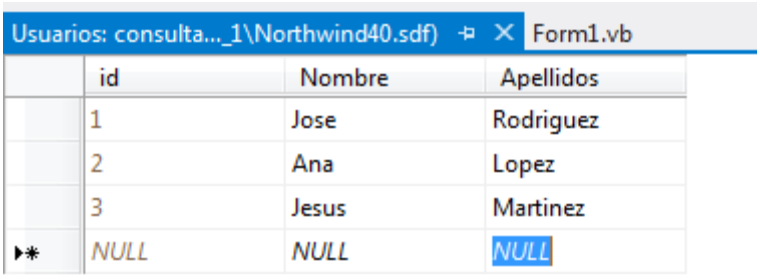
Nombre de columna	Tipo de datos	Longitud	Permitir valores NULL	Único	Clave principal
id	int	4	No	No	Sí
Nombre	nvarchar	100	Sí	No	No
Apellidos	nvarchar	100	Sí	No	No

Un detalle, las columnas de tipo "Identity" no permiten valores nulos así que los debemos marcar. Además, le indicaremos que es la clave principal para nuestra tabla.

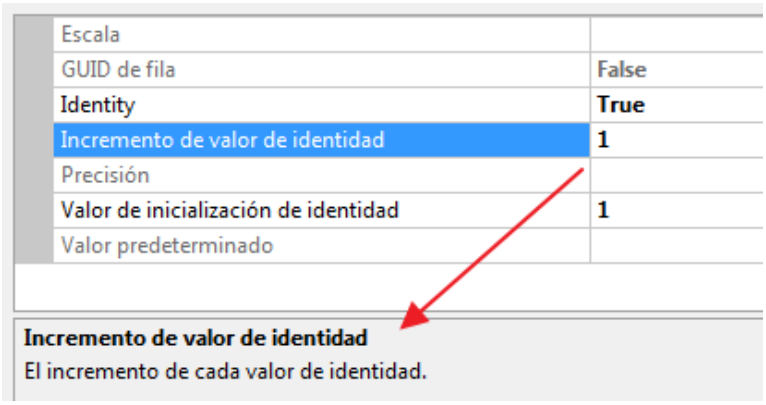
Pulsaremos en "Aceptar" y nos creará la tabla. Luego con el botón derecho indica que queremos ver los datos de la tabla:



Meteremos varios datos de prueba escribiendo solo en las columnas de nombre y apellidos. Veremos como la columna "Id" se va completando automáticamente con valores numéricos consecutivos:



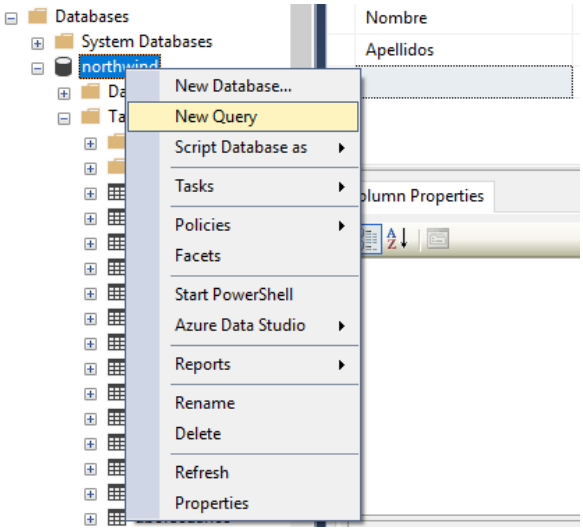
Hemos podido comprobar que los datos inicialmente son "Null" y que al ir introduciendo datos en "Nombre" y "Apellidos" y pulsar el cursor hacia abajo, se rellenaba automáticamente la columna de "Id". Perfecto, eso es lo que queríamos, un campo que se rellene solo con un valor único para así poder definir cada fila de forma única. Y ¿por qué empieza en el número 1 e incrementa de uno en uno? Pues porque por defecto es así y así nos lo indicaba antes cuando le hemos dicho que fuera de tipo "identity":



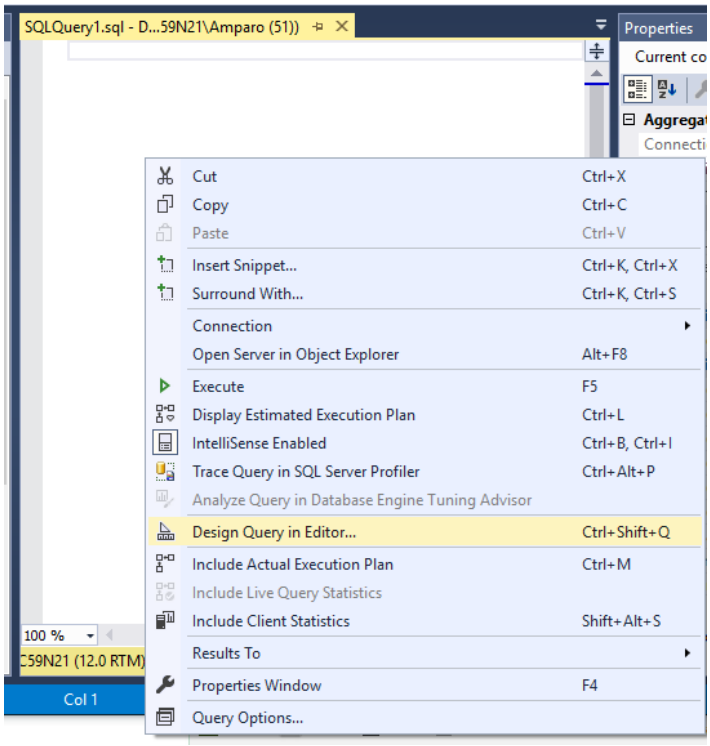
Veamos un rápida explicación de porqué necesitamos una clave única en nuestras tablas... Imaginemos que tenemos dos "fernandez lopez" en nuestra tabla de empleados y ejecutamos una instrucción para cambiar un dato en un "fernandez lopez" que tengo en pantalla. SQL Server o la base de datos no sabrá distinguir entre los dos si no existe un identificador único que los diferencie, como por ejemplo el DNI: actualiza los datos del empleado con DNI número XXX. De ahí que siempre deba existir un campo que apunte de forma única a cada fila. En este caso un "identity" hace que nunca haya dos iguales ya que proporciona automáticamente el siguiente valor del contador.

4.1. Realizar consultas

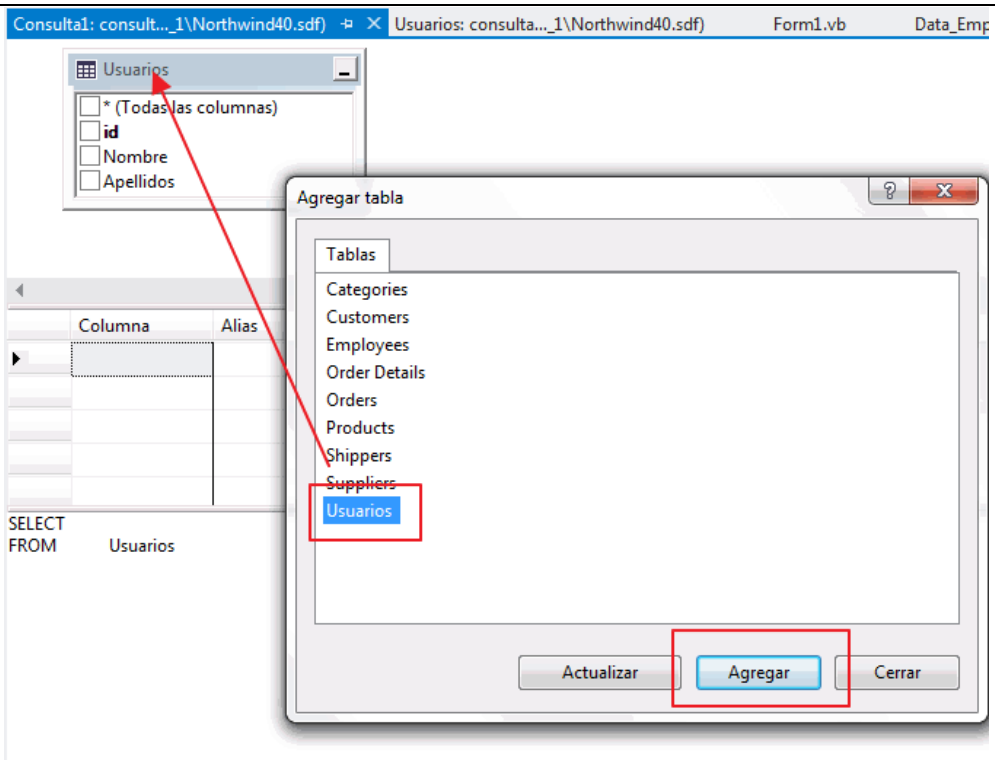
Tenemos más posibilidades para generar consultas con nuestro administrador de conexiones. Seleccionemos la opción:



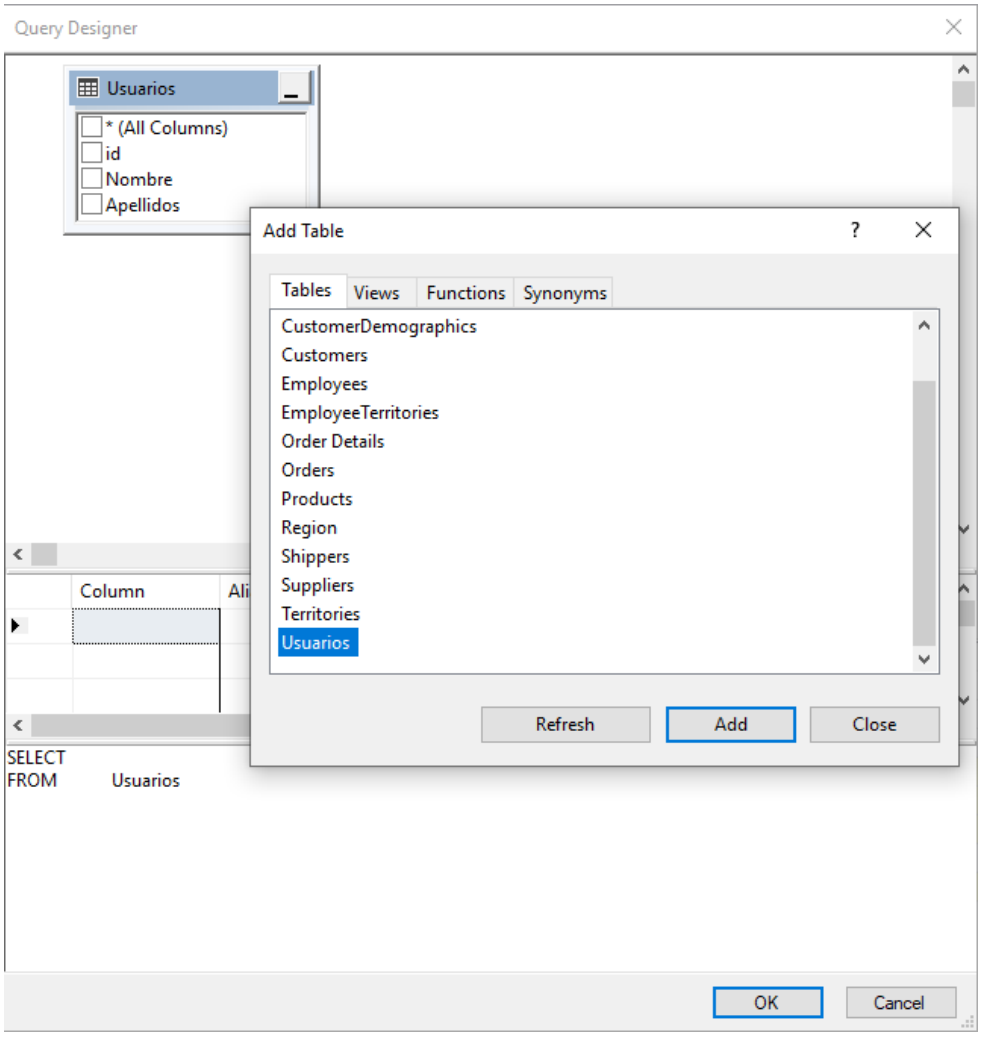
Hacemos botón derecho sobre el espacio en blanco que nos aparece y elegimos “Design Query in Editor”



Es un editor gráfico que nos permite crear una consulta. Añadimos primero las tablas que van a intervenir en la consulta. Seleccionaremos solo la de los usuarios que acabamos de crear:



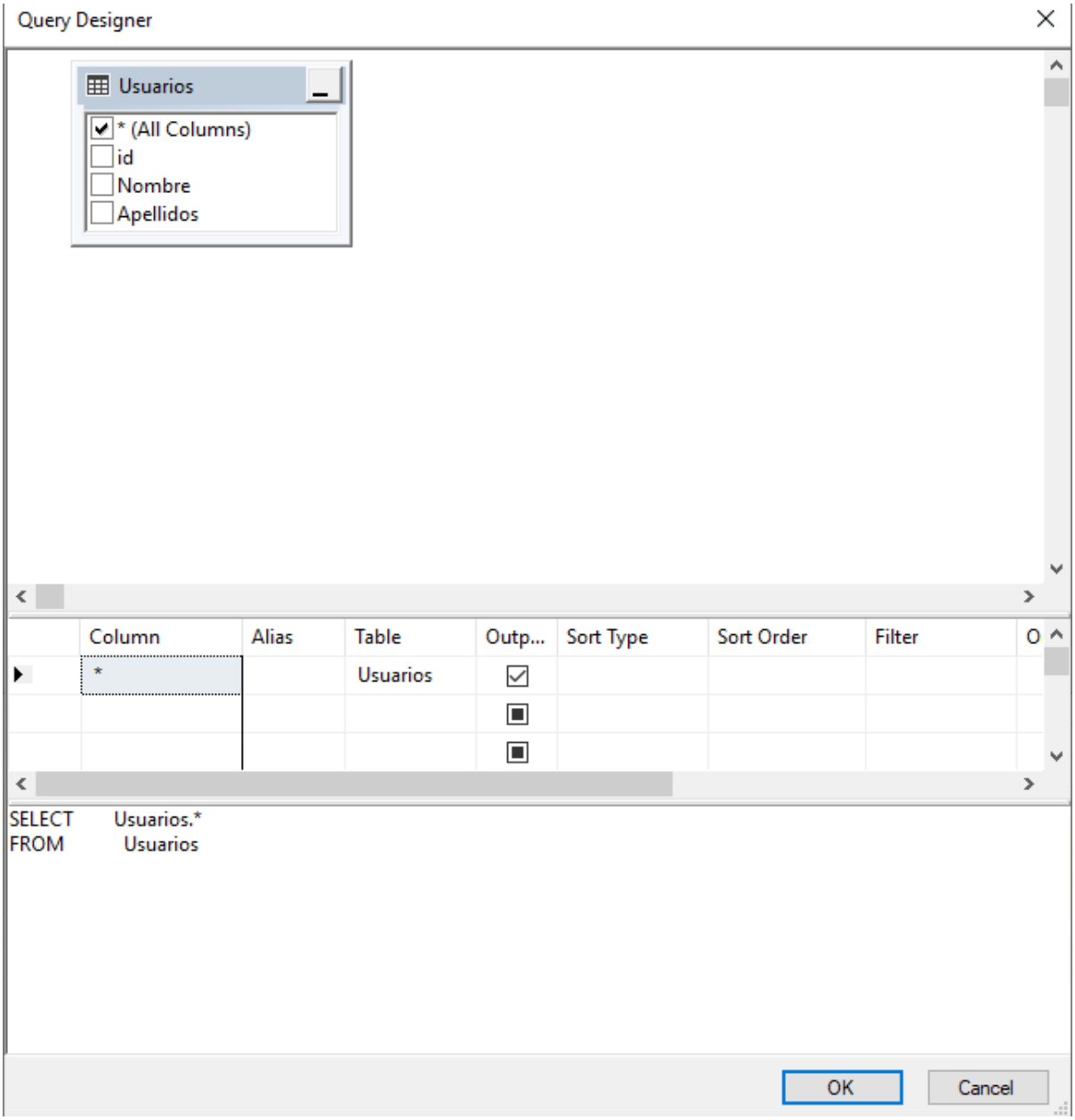
Al pulsar en Agregar aparece ahora en el diseñador.



Añadimos las tablas que necesitemos y pulsamos en “Cerrar”.

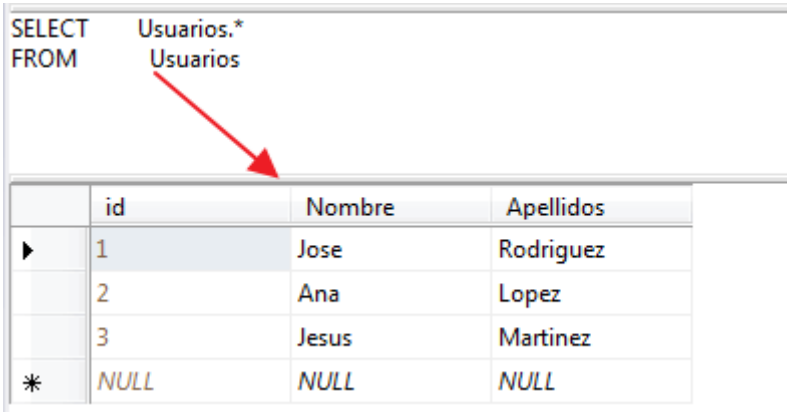
Construimos la consulta seleccionando los campos y las opciones necesarias y pulsamos “Ok”.

En este caso, marcamos que queremos recuperar todas las columnas, seleccionando el "* (Todas las columnas)".



Finalmente para ejecutar la consulta pulsamos F5 o “Ejecutar” y obtendremos los resultados.

La consulta SQL se habrá ejecutado devolviéndome el conjunto de resultados, que, en este caso, son todas las filas de la tabla:



Resumiendo, en la primera sección tenemos las tablas que van a componer la consulta en este generador. El segundo cuadro es muy versátil y nos permitirá seleccionar columnas y añadir criterios. El tercer panel nos mostrará la sentencia SQL que va a ejecutar y por fin en el panel de abajo tenemos el resultado de ejecutar esa sentencia SQL.

Tenemos cuatro operaciones básicas en las bases de datos, que te recuerdo ahora con su sentencia equivalente debajo:

- Altas. "insert into clientes (nombre,dni) values ('JoseRodriguez',123456789)
- Bajas. "delete from clientes where dni=123456789"
- Consultas. "select * from clientes"
- Modificaciones. "update clientes set poblacion='Logroño' where dni=1234567890"

Podemos utilizar cualquiera de estas instrucciones en nuestro generador de consultas escribiéndolas en el panel y luego ejecutándola desde el botón que utilizamos antes.

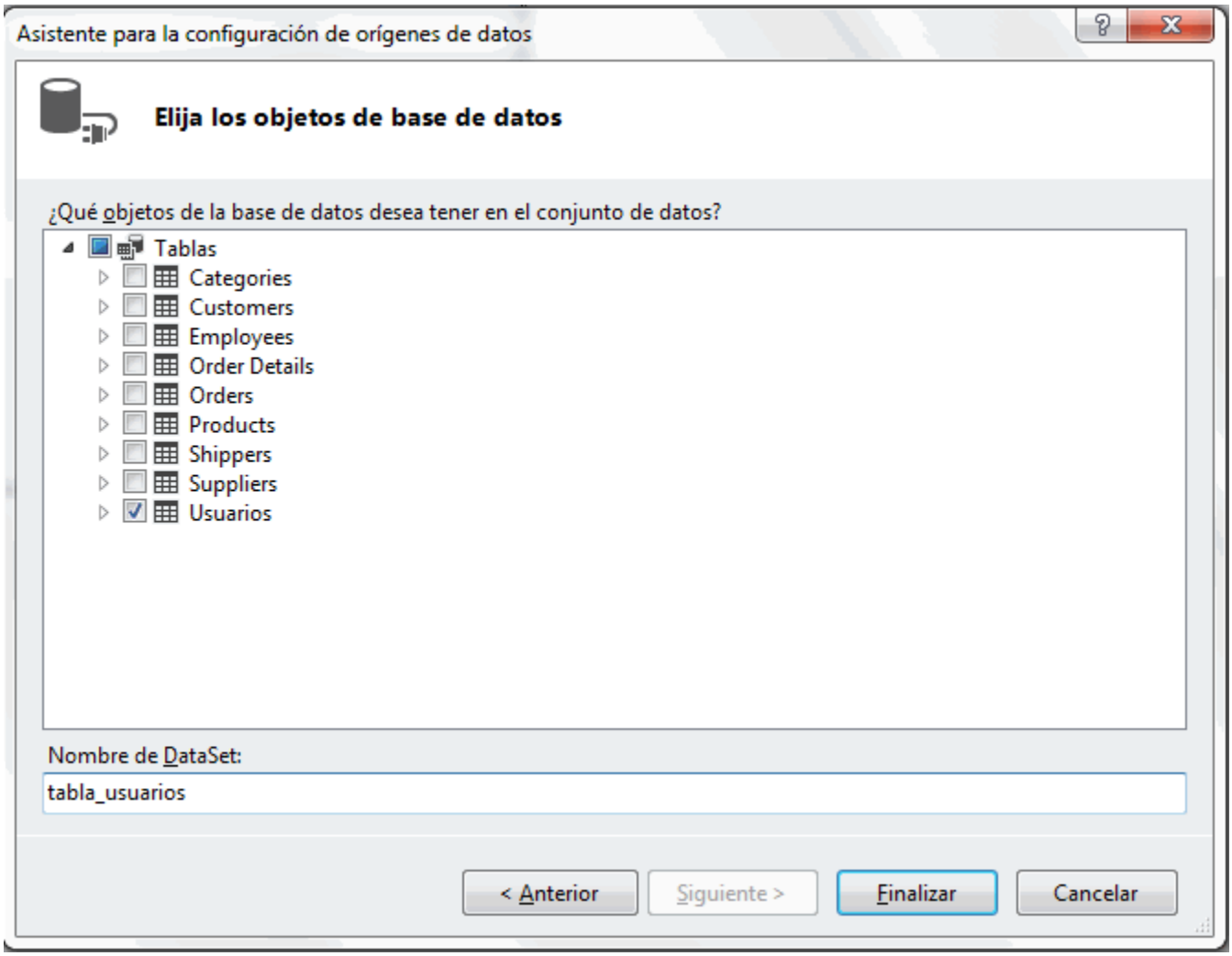
5. Código ADO.NET

Después de esta pequeña introducción a las bases de datos y a la ayuda gráfica que nos ofrece VB.NET vamos a bajar un poco hasta el código para hacer las operaciones "a mano" para aprender qué objetos tenemos y cómo es el código para acceder a las bases de datos.

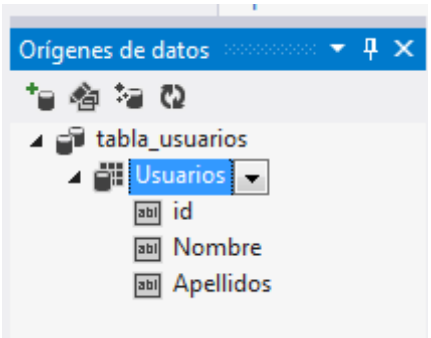
Comenzaremos creando un nuevo proyecto. En este proyecto añadiremos un nuevo origen de datos, tal y como hicimos antes.

La única precaución que debemos tener es que a la hora de seleccionar el fichero de base de datos, elijamos el que utilizamos en el ejemplo anterior, ya que es el que tiene nuestra base de datos con la tabla de "Usuarios" que hemos creado.

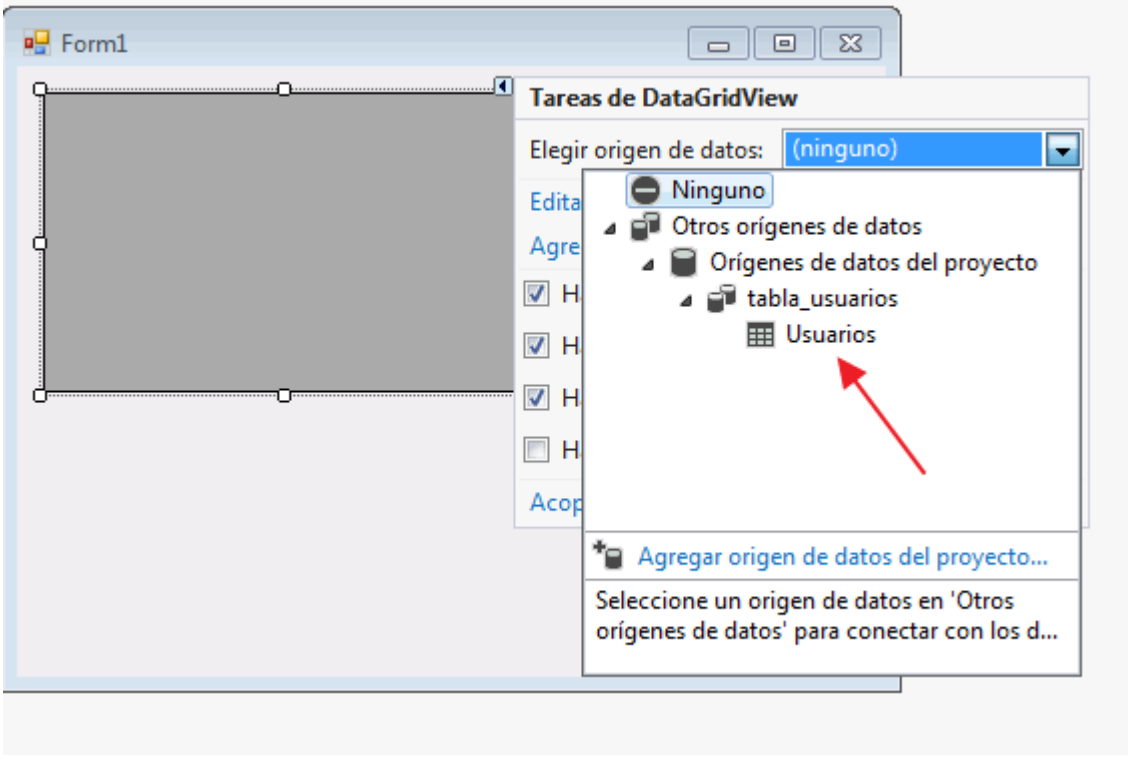
Sigamos con el asistente que nos crea la conexión, seleccionando ahora esta tabla:



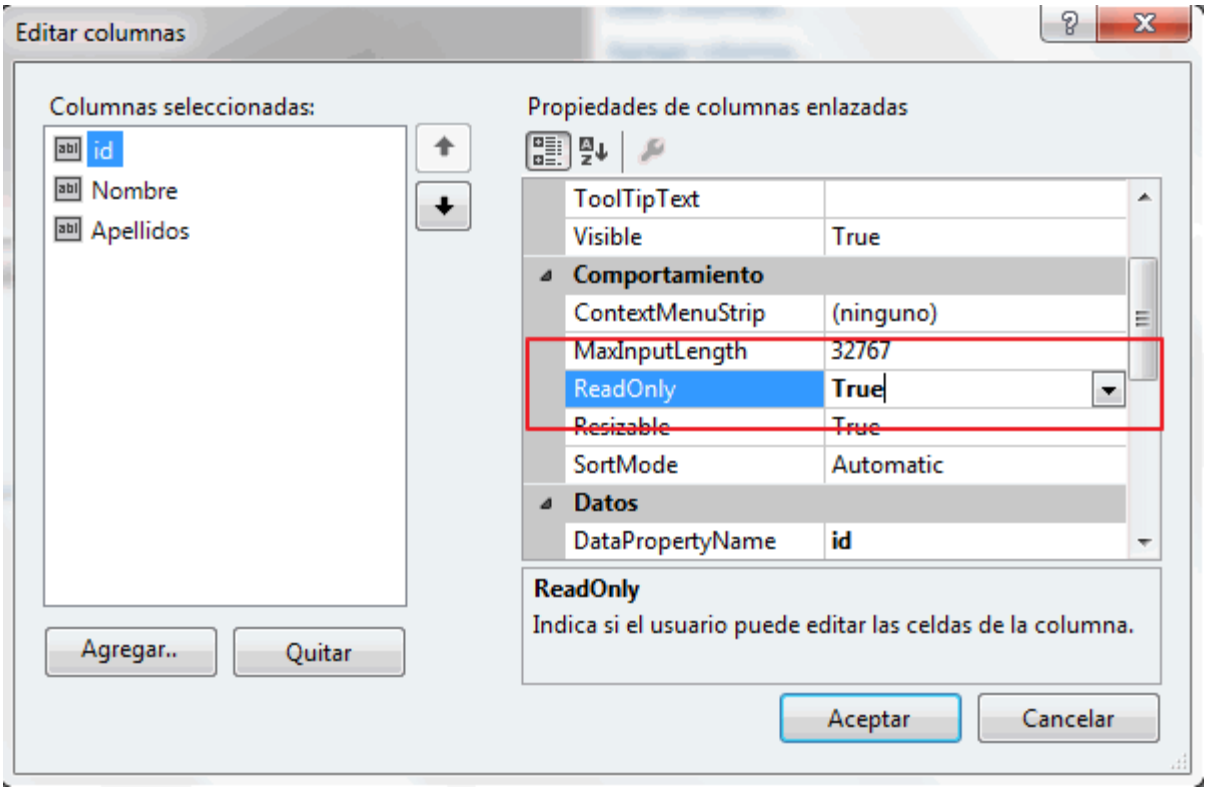
Finalizamos y ya tenemos incorporado a nuestro proyecto una conexión esta tabla:



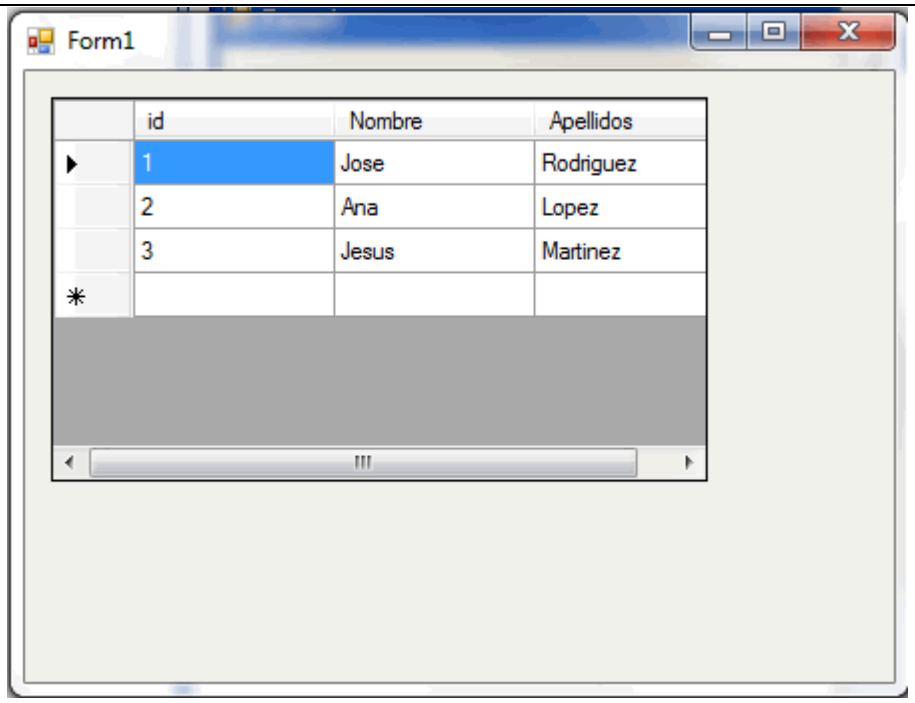
Añadimos un control "DataGridView y seleccionamos este origen de datos:



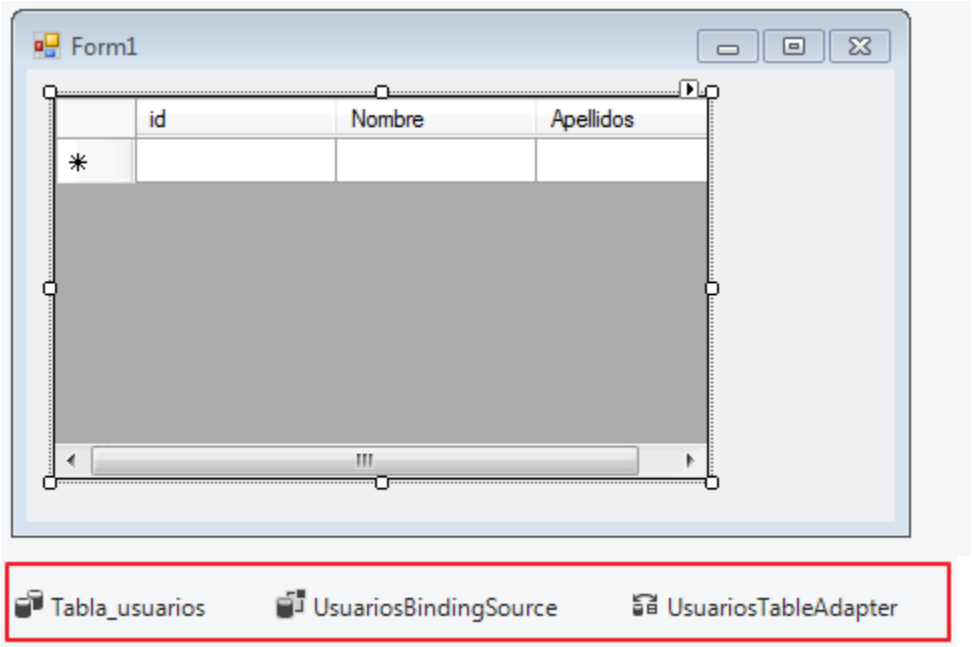
Vinculamos la tabla, le ponemos unos títulos de las columnas y marcamos la del identificador de la fila "id" como de solo lectura ya que no podemos modificarla:



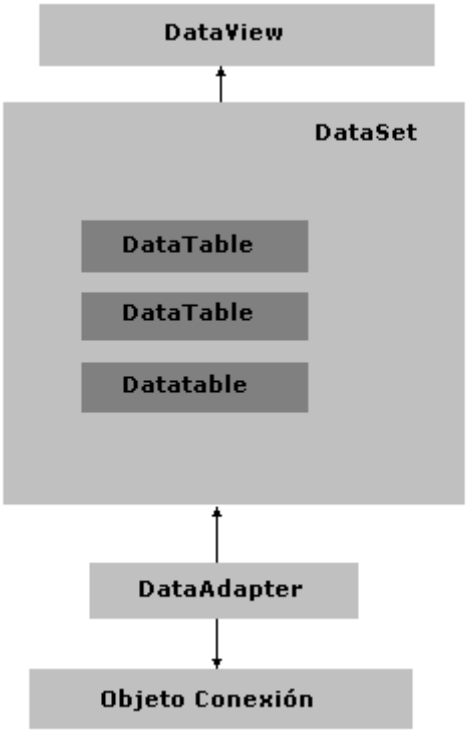
Bueno, no ha hecho falta. Al tratarse de un campo especial que se incrementa automáticamente no debe ser editable y nos lo marca ya como de solo lectura. Ejecutemos el programa para comprobar que funciona:



Volvemos a la vista de diseño y nos fijamos en los controles que nos ha puesto automáticamente VB.NET para poder realizar estas operaciones:

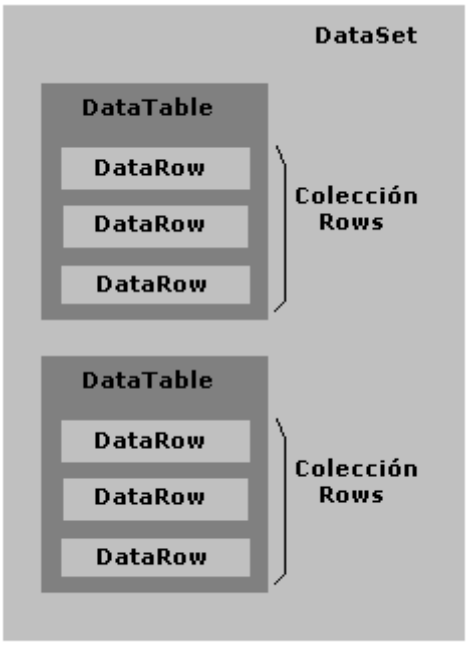


Tenemos varios objetos que iremos viendo a largo de este tema dedicado a las bases de datos. Veamos este esquema de objetos:



Desde un objeto "conexión" establecemos un enlace con un conjunto de resultados "Dataset" a partir de un objeto "DataAdapter", que es el que nos realizará la conexión entre nuestro programa y los datos. Finalmente de ese conjunto de datos podremos realizar una vista de datos con un "DataView".

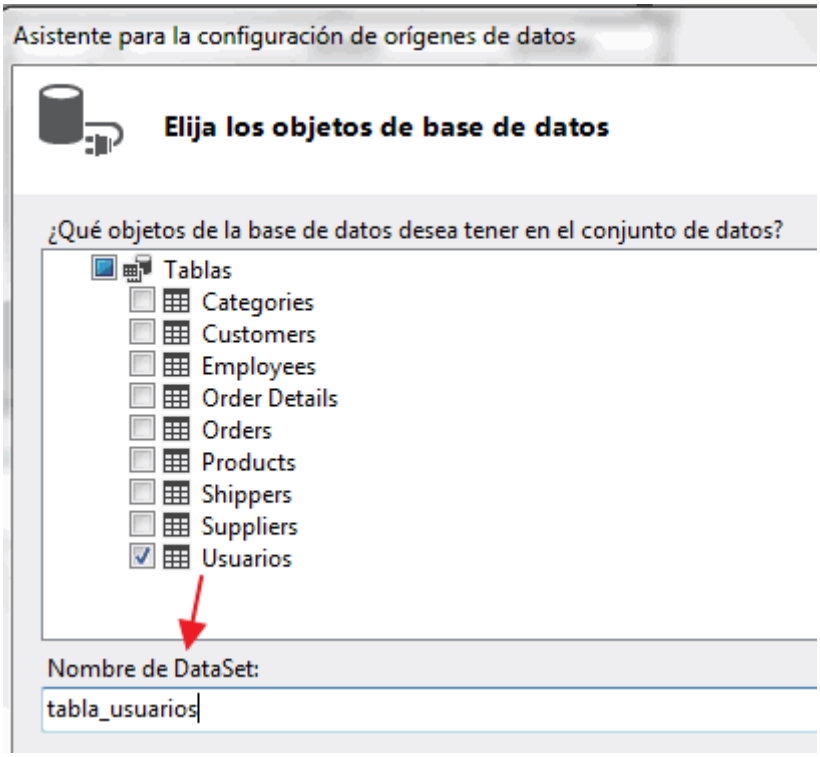
Por tanto, ese adaptador o "DataAdapter" es el que nos comunica con los datos. El otro objeto, el "Dataset" es el contenedor de los datos. Es decir, donde están nuestras consultas, como en el ejemplo que hemos hecho. Estos DataSets son muy versátiles, veamos este esquema:



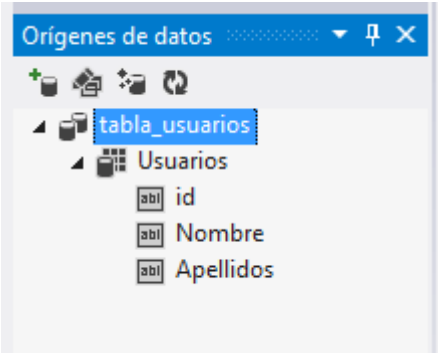
Se compone de una colección de tablas "DataTable" que a su vez tienen una colección de filas. En nuestro ejemplo el Dataset tiene una sola tabla "DataTable" y con una colección de filas "DataRow" que son los registros que hemos metido antes. De momento nos quedaremos con los nombres de los objetos, sigamos escribiendo algo de código.

Los adaptadores o "DataAdapter" proporcionan el enlace entre los Dataset que son los datos que podemos mostrar y la base de datos física. En el ejemplo el Dataset es nuestra tabla de usuarios

Al crear la conexión nos solicitó un nombre para este Dataset, recordemos:



En nuestro programa, el DataSet aparece aquí con la tabla de usuarios que tiene estos tres campos:



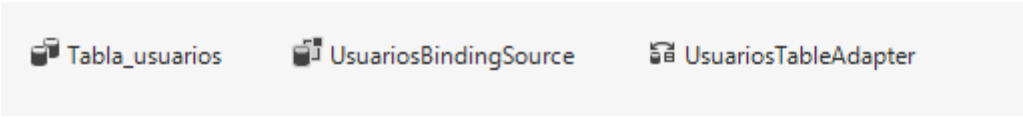
Así vamos aprendiendo lo iconos. Jerárquicamente aparece primero el DataSet y posteriormente la tabla con los datos. Por tanto: "tabla_usuarios" es el "Dataset" , "Usuarios" es un "DataTable" contenido dentro del Dataset y las filas de datos son una colección de "DataRows" de esta tabla. Esto se va a repetir siempre así que debemos quedarnos con estos nombres de los objetos.

En el otro lado tenemos la conexión a la base de datos y el interlocutor entre esta base de datos física y el Dataset anterior. Este interlocutor es el famoso DataAdapter. Como es el interlocutor entre los datos y el fichero físico es donde le indicaremos los comandos que tiene que ejecutar y cómo. Es decir, dentro de ese adaptador definiremos los comandos SQL para poder modificar los datos.

Un objeto DataAdapter puede contener desde una sencilla consulta SQL hasta varios objetos Command. Esta clase tiene cuatro propiedades que permiten asignar un objeto Command con las operaciones estándar de manipulación de datos:

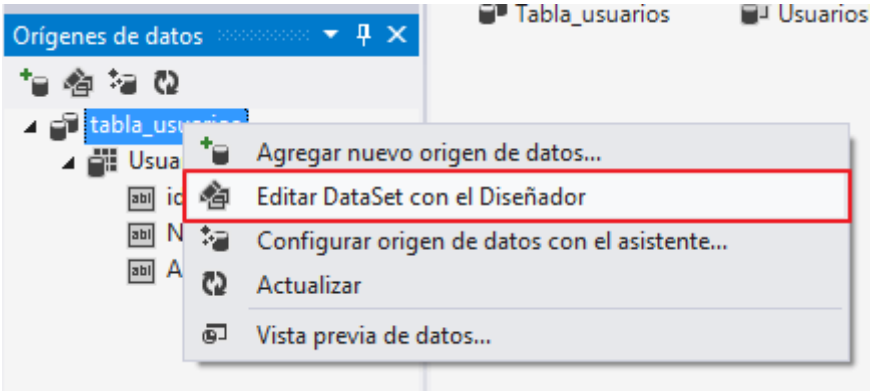
Propiedad	Descripción
InsertCommand	Objeto de la clase Command que realiza una inserción de datos
SelectCommand	Objeto de la clase Command que ejecuta una sentencia Select de SQL
UpdateCommand	Objeto de la clase Command que realiza una modificación de datos
DeleteCommand	Objeto de la clase Command que realiza una eliminación de datos

Volvamos a los iconos que nos ha creado:

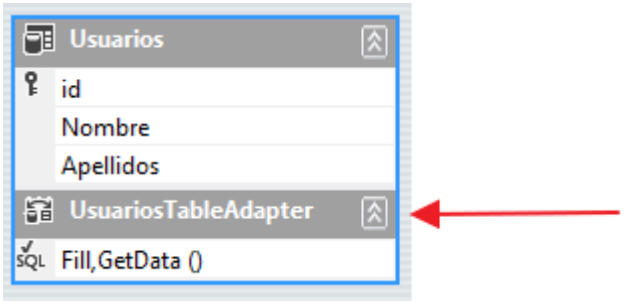


Tenemos un DataSet llamado "tabla_usuarios", una conexión "UsuariosBindingSource" y un DataAdapter que le ha llamado "UsuariosTableAdapter". Tenemos, por tanto, todo lo necesario: conexión, adaptador y dataset. Lógicamente es lo que nos ha añadido VB porque es lo que necesita para poder acceder a los datos y así mostrarlo en la cuadrícula.

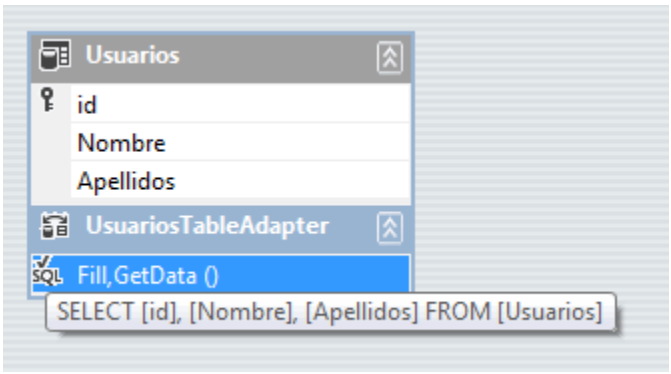
Ahora revisemos esto:



Es la edición del Dataset. Es decir, del objeto que rellenamos con el contenido de la tabla mediante el método "Fill":



Vemos que tenemos la tabla "usuarios" y debajo el adaptador o "DataAdapter" que tiene una instrucción SQL dentro. Si marcamos el adaptador y pasamos el ratón encima del icono que pone "sql":



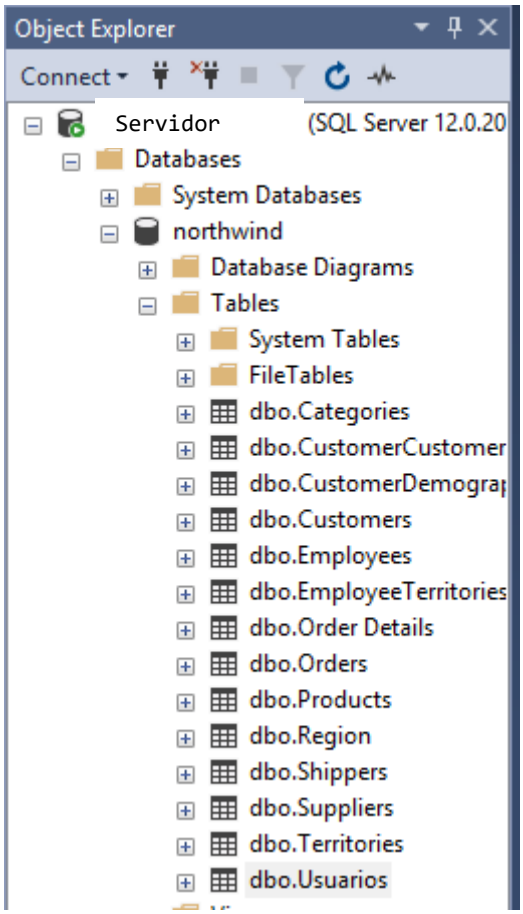
Este adaptador, como vemos, tiene almacenado cómo debe hacer la consulta para poder generar el dataset de "usuarios". Como ves es la conexión entre los datos físicos y nuestro Dataset con el resultado "Usuarios".

No hago más que contarte lo mismo desde varios puntos de vista y para conocer cómo ha hecho VB para generarnos esa pantalla con todo funcionando. No ha hecho nada raro: ha añadido una conexión, un DataAdapter con los comandos necesarios y un DataSet con los resultados.

Por tanto, cuando con el IDE añadimos un origen de datos, éste introduce las sentencias SQL en el DataAdapter y así poder funcionar.

6. Edición de tablas con el explorador de objetos

Ya vimos antes cómo podíamos editar las tablas para crear nuevos campos y así poder modificar su estructura. Veamos más detalles sobre eso. Recordemos que en Microsoft SQL Server Management Studio tenemos una ventana con el explorador de objetos:

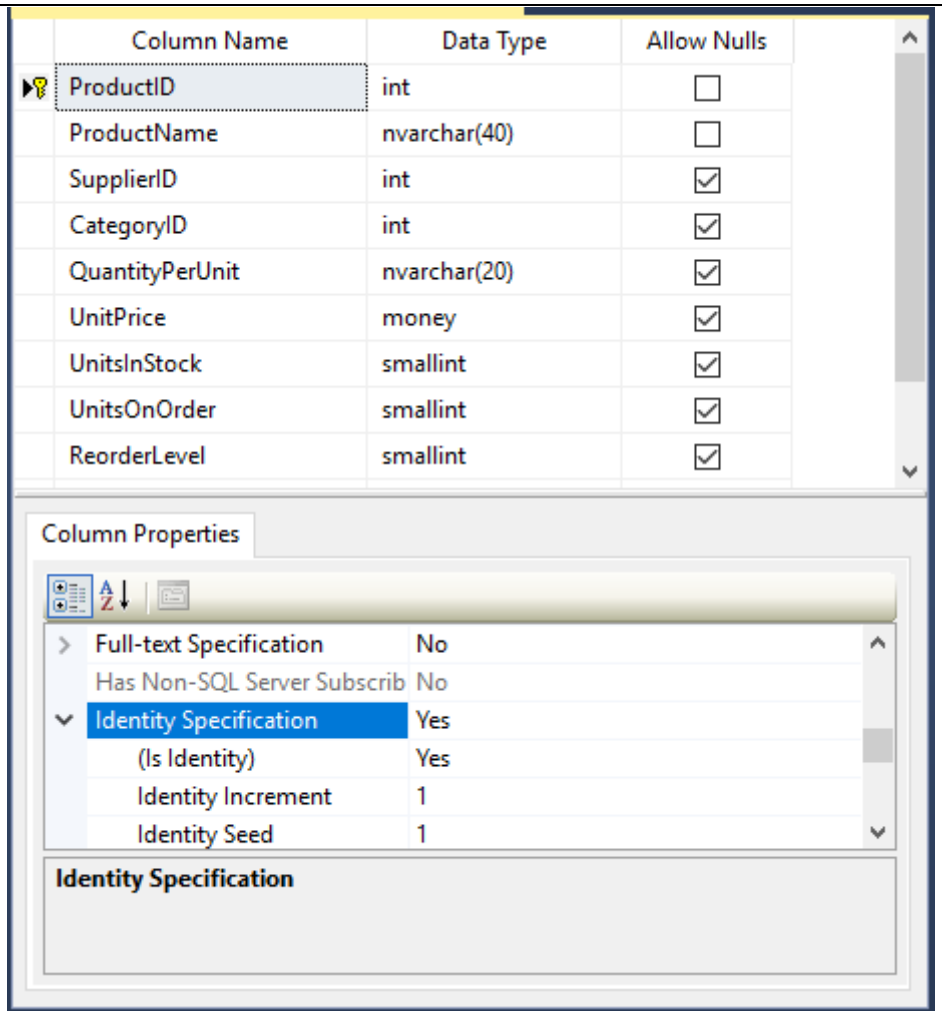


La primera ventana es un editor general de bases de datos. Simplemente es una herramienta para crear tablas, modificar su estructura, ...

Por otra parte, en Visual Studio tenemos la ventana, "Orígenes de datos", que son las fuentes de datos, llamadas "Dataset" que van a alimentar nuestra aplicación. Al enlazar el control de DataGridView nos solicitaba el nombre de un "dataset" ya definido que correspondía con la ventana de los orígenes de datos.

Ya tenemos claro pues que el explorador de objetos es un miniadministrador de bases de datos en el que podemos realizar muchas tareas de mantenimiento de éstas.

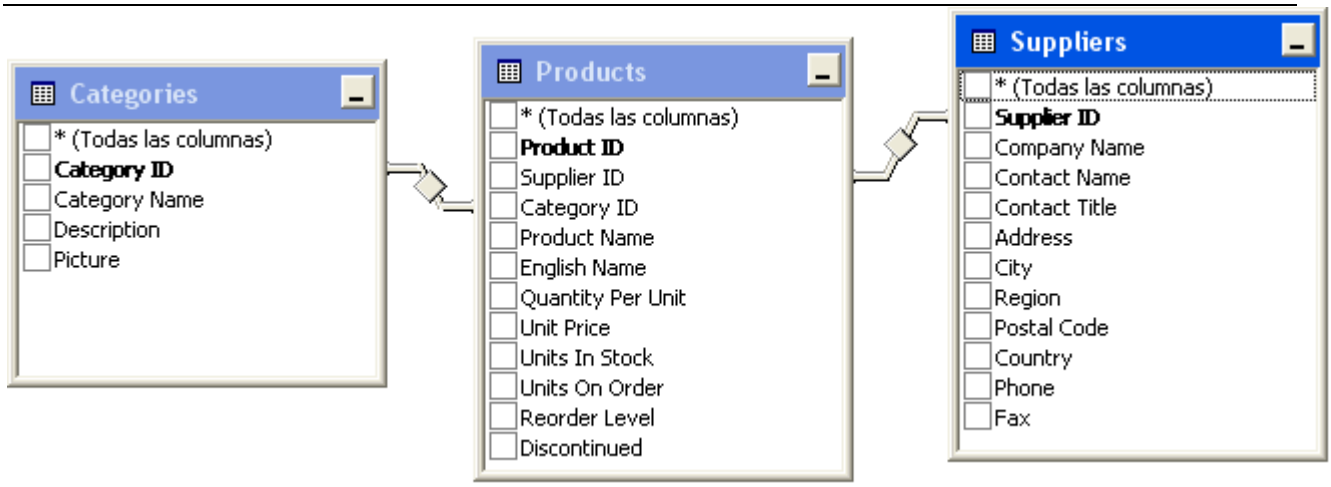
En el explorador de bases de objetos anterior vemos que hay una tabla llamada "Products", vamos a pulsar con el botón derecho para su "Diseño":



Vemos que tiene un campo con el identificador del producto que lo ha definido como hicimos antes, con el tipo de datos entero (int). Además tiene activada la propiedad de que sea un "identity" para así poder tener un valor único y referirnos a cada producto de forma única.

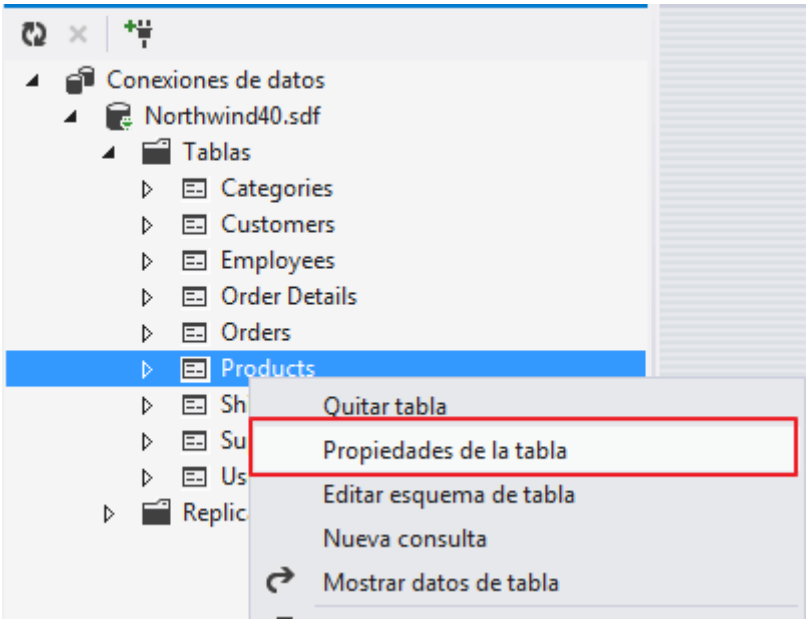
Vemos además como está relacionada con otras tablas, una de proveedores "Suppliers" (con el campo "SupplierID") y otra de categorías "Category" (con el campo CategoryID) donde almacena sólo el identificador de cada uno de estos, ya que con este dato quedan perfectamente conectadas las tablas.

Cuando se define una base de datos, además de crear las tablas y pensar que deben estar relacionadas, tenemos que crear esas relaciones, que en nuestro caso anterior sería:

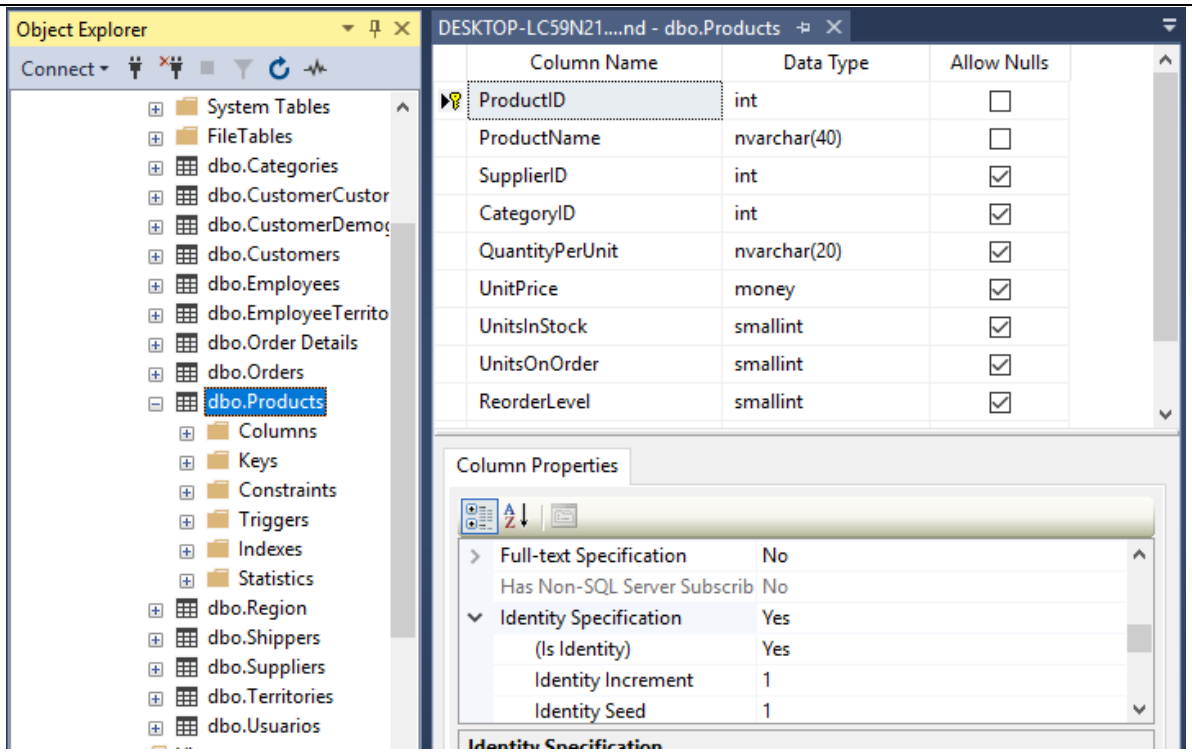


La tabla de productos está relacionada con la tabla de categorías a través del campo "Categoryid" que por supuesto son del mismo tipo de datos. La tabla Productos está relacionada también con la tabla de proveedores "Suppliers" a través del campo "SuplierID" que si te fijas, son los campos claves de sus respectivas tablas que son de tipo entero con la propiedad "identity", así no hay duda de que hacen referencia a ese producto.

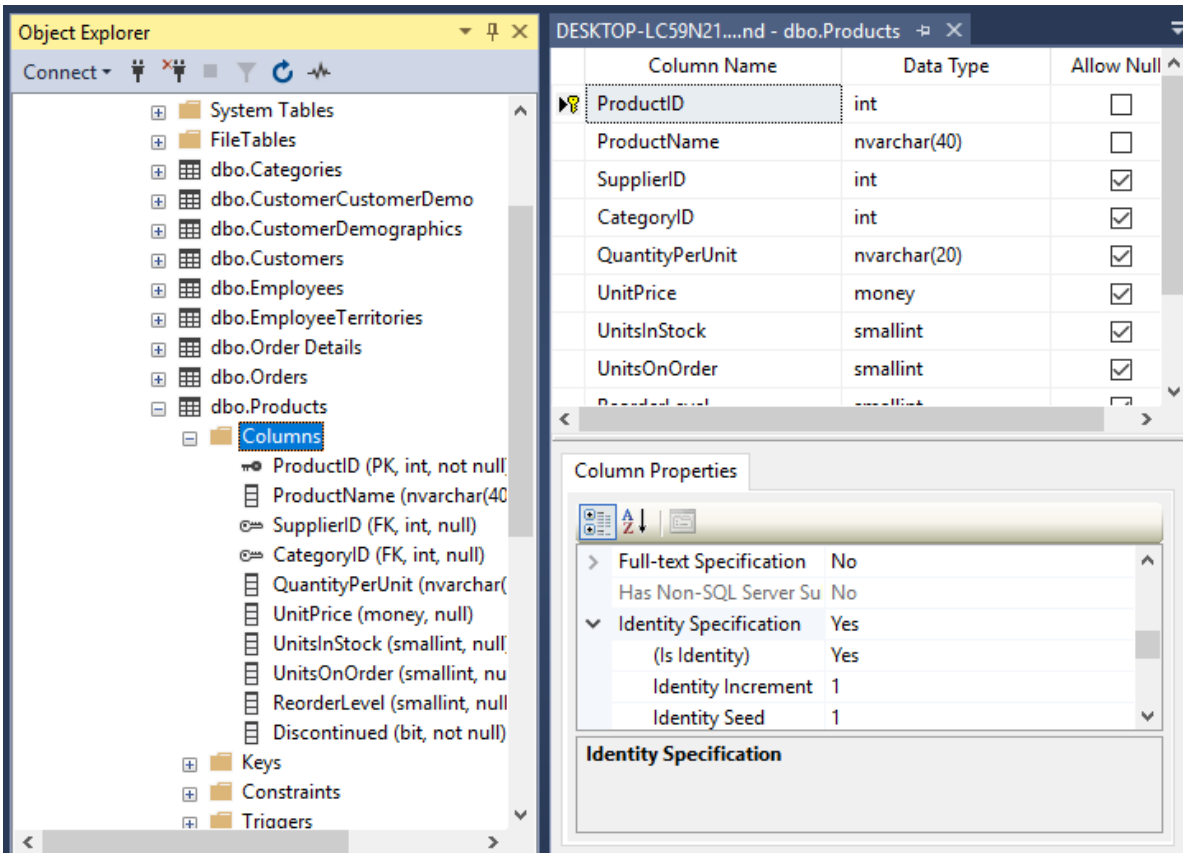
En nuestro ejemplo se producen dos relaciones de muchos a uno: Un proveedor puede tener muchos productos y lo mismo con una categoría que la pueden tener muchos productos. Vamos a pulsar con el botón derecho en la tabla de los pedidos categorías y le vamos a decir que:



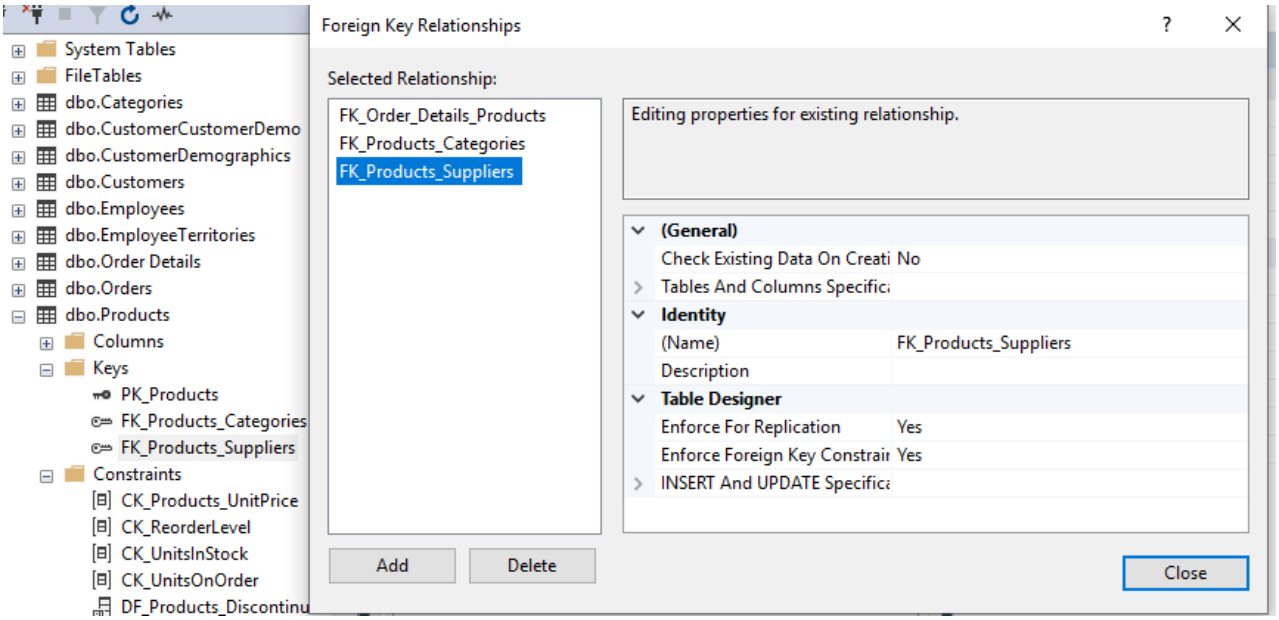
Queremos ver sus propiedades. No editarla sino sólo ver sus propiedades:



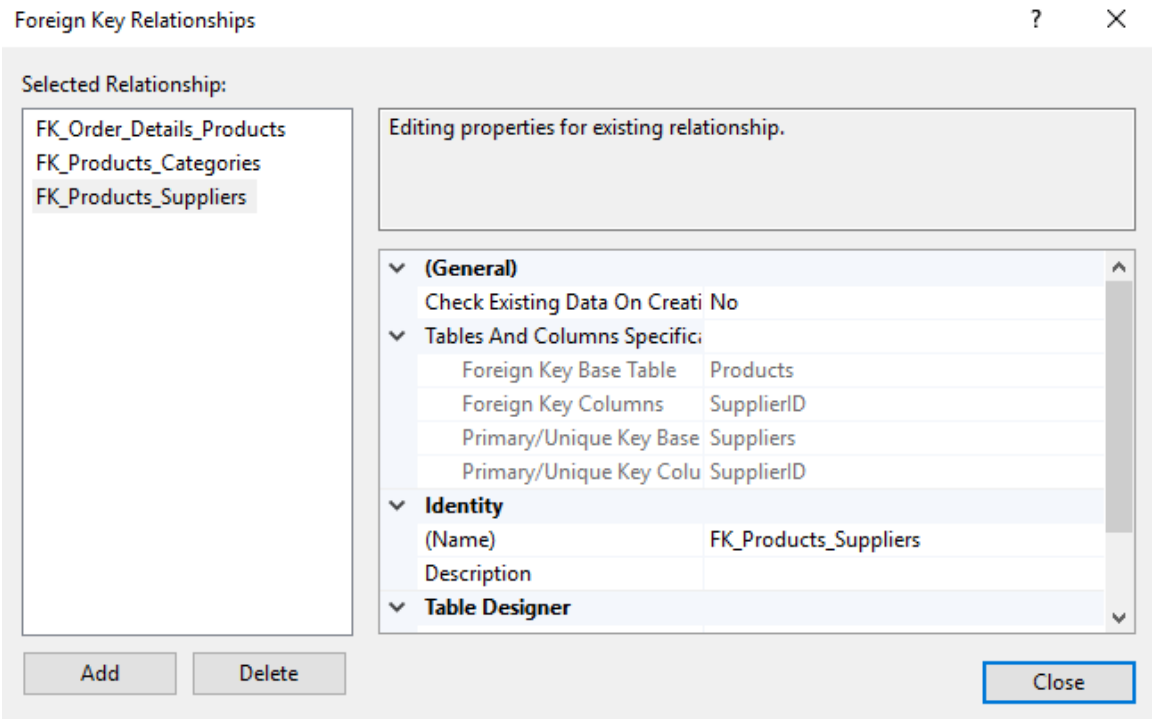
En la segunda opción de la izquierda, "Columna", podemos ver todos los detalles de la tabla. Fíjate que nos indica que campos son claves, el tipo de datos, si alguno de ellos es de tipo "identity" y algunos datos interesantes más.



Si pulsamos en "Constraints" y en “Keys” en las opciones de la izquierda, veremos:



Así es cómo se definen internamente las relaciones. Hay una relación creada llamada "FK_Products_Suppliers" que relaciona las tablas "Suppliers" y "Products", es decir la de proveedores y productos y cuyos campos de relación en las tablas con "SuplierID" que se llaman de igual forma en las dos tablas, así es como debe ser.

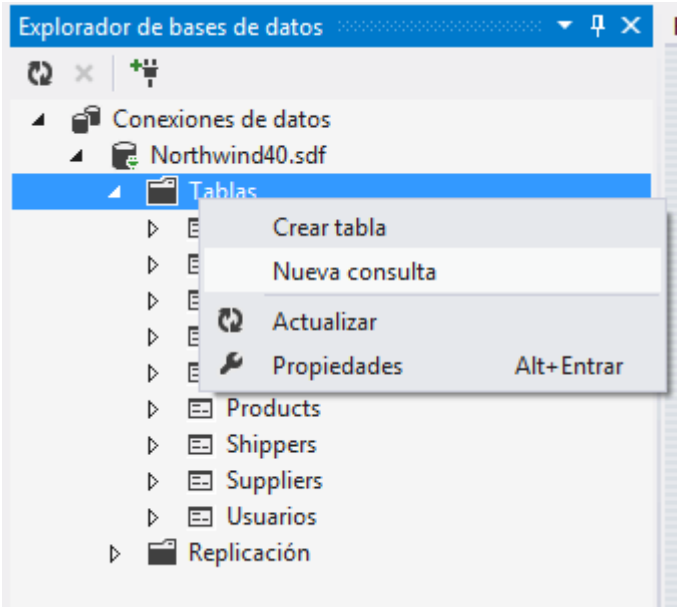


Si pulsamos en el desplegable, veremos la otra relación existente con la otra tabla.

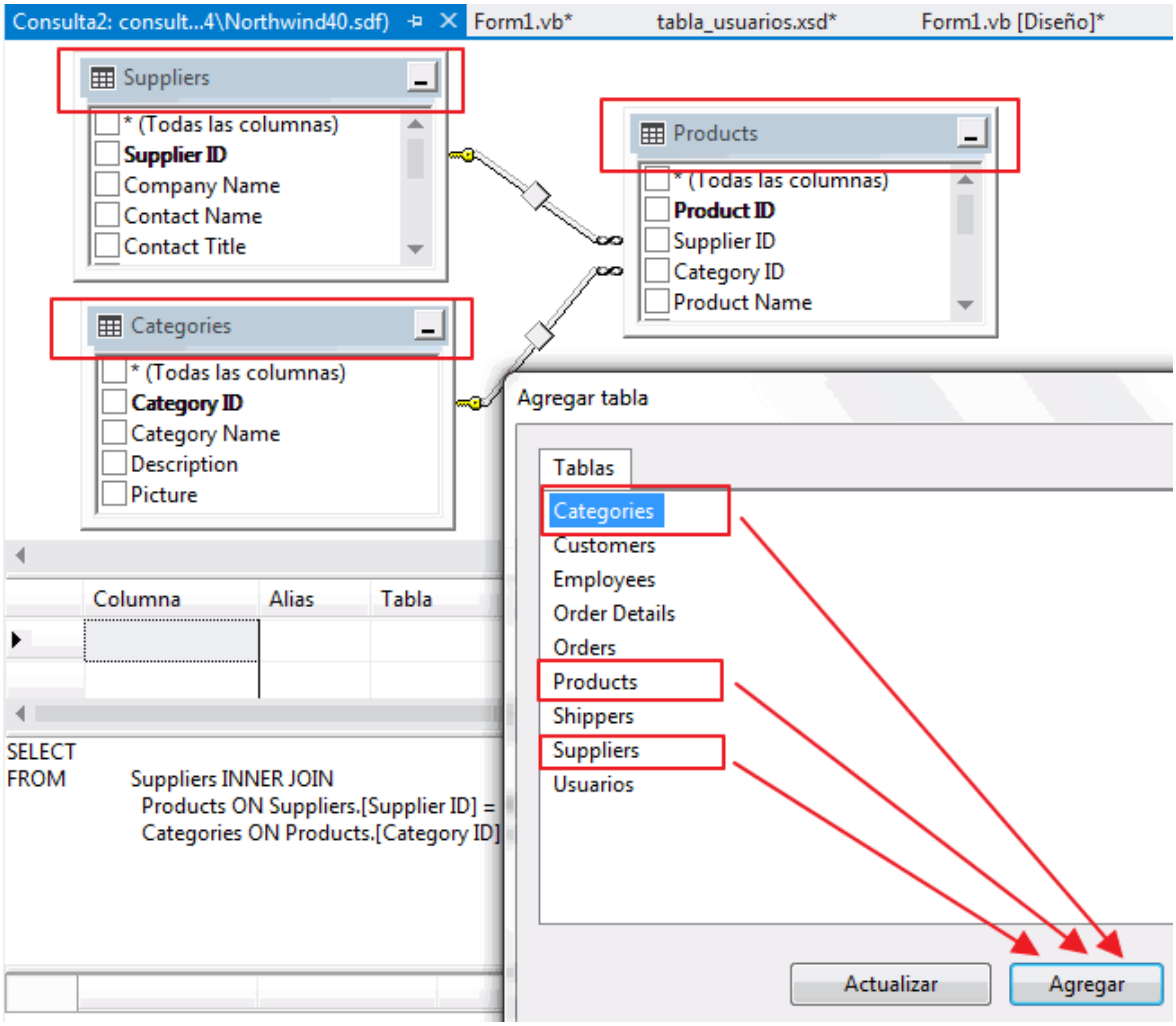
La opción de "Agregar" permitiría crea una nueva.

Definiríamos un nombre, unas reglas y las tablas y campos que intervienen en la relación.

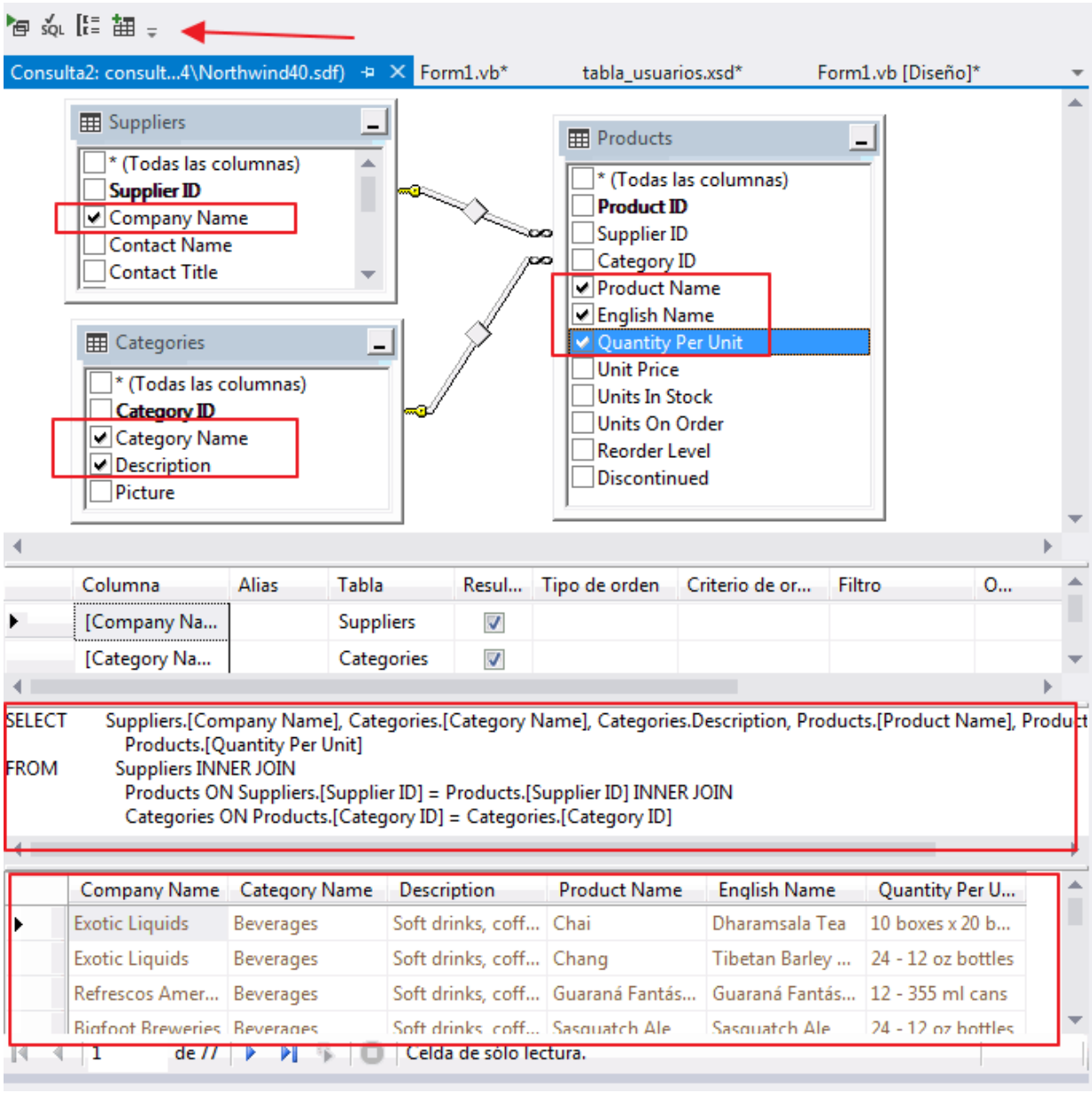
Vamos a realizar ahora una consulta cruzando estas tres tablas. Pulsamos con el botón derecho en nuestra base de datos o tablas para elegir una nueva consulta:



Seleccionamos ahora las tres tablas una a una y vamos pulsando en Agregar:



Cerramos la ventana y tendremos un gráfico con las tres tablas conectadas. Al tener las relaciones ya creadas, la pantalla nos muestra gráficamente con esos enlaces los campos que componen la relación. Ahora vamos a seleccionar unos cuantos campos de cada una:

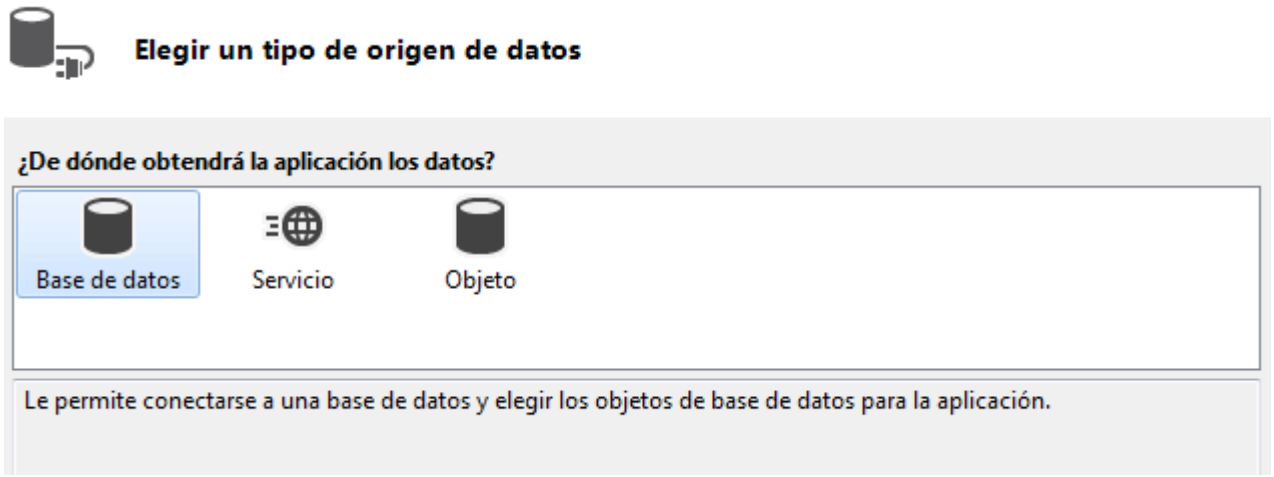


Hemos seleccionado unos campos de las tres tablas y después hemos pulsado en ejecutar la consulta en la barra de botones. En la parte intermedia podemos ver la sentencia SQL resultante de esa combinación de campos y en la parte inferior el conjunto de resultados, o Dataset:

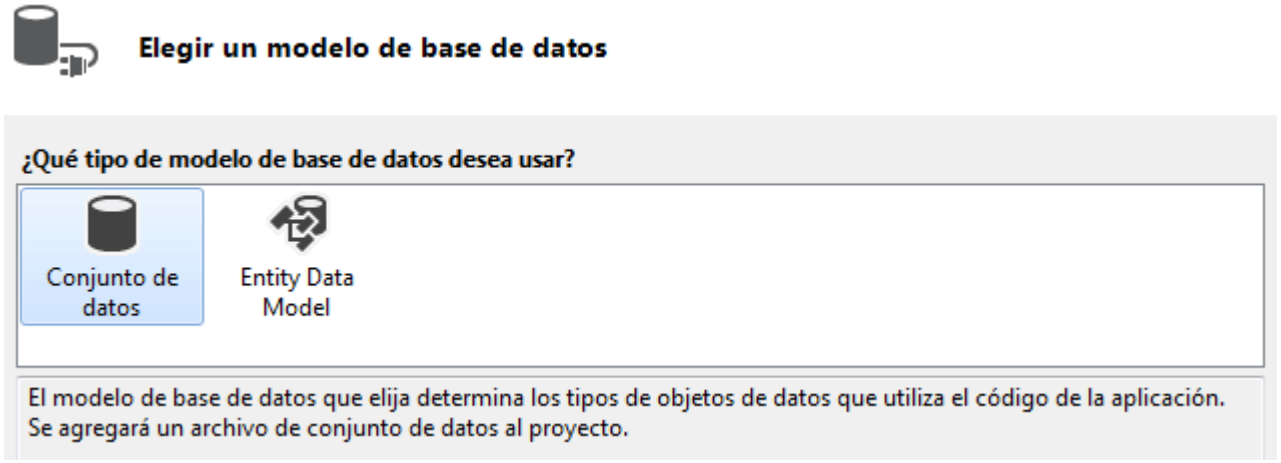
Al estar relacionadas las tablas podemos recuperar el nombre del proveedor y el nombre de la categoría del producto de sus respectivas tablas, gracias a su relación con los índices: "SupplierID" y "CategoryID". En el segundo panel podríamos poner restricciones, filtros y agrupaciones y en el siguiente tenemos la sentencia SQL equivalente a la selección que hemos hecho. Donde se ven las instrucciones "INNER JOIN" que construyen la consulta con la relación entre las tablas.

7. La ventana de orígenes de datos.

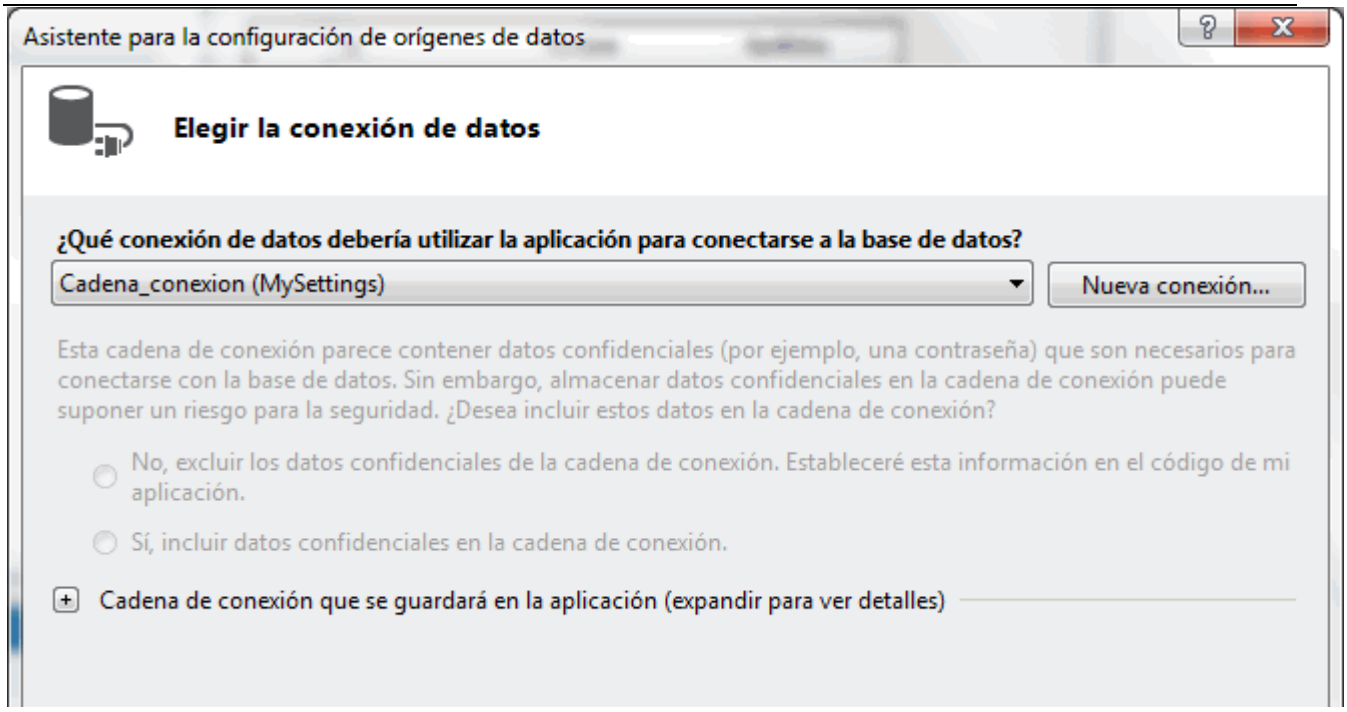
Volvemos ahora con la ventana de los orígenes de datos. Aquí tenemos temas muy distintos ya que como sabemos, se tratan de los datos que van a alimentar nuestra aplicación. Son, por tanto, conexiones activas con nuestra aplicación. Esta ventana tiene dos vistas, vamos primero a crear algún Dataset. Añadimos un nuevo origen de datos a nuestra aplicación: :



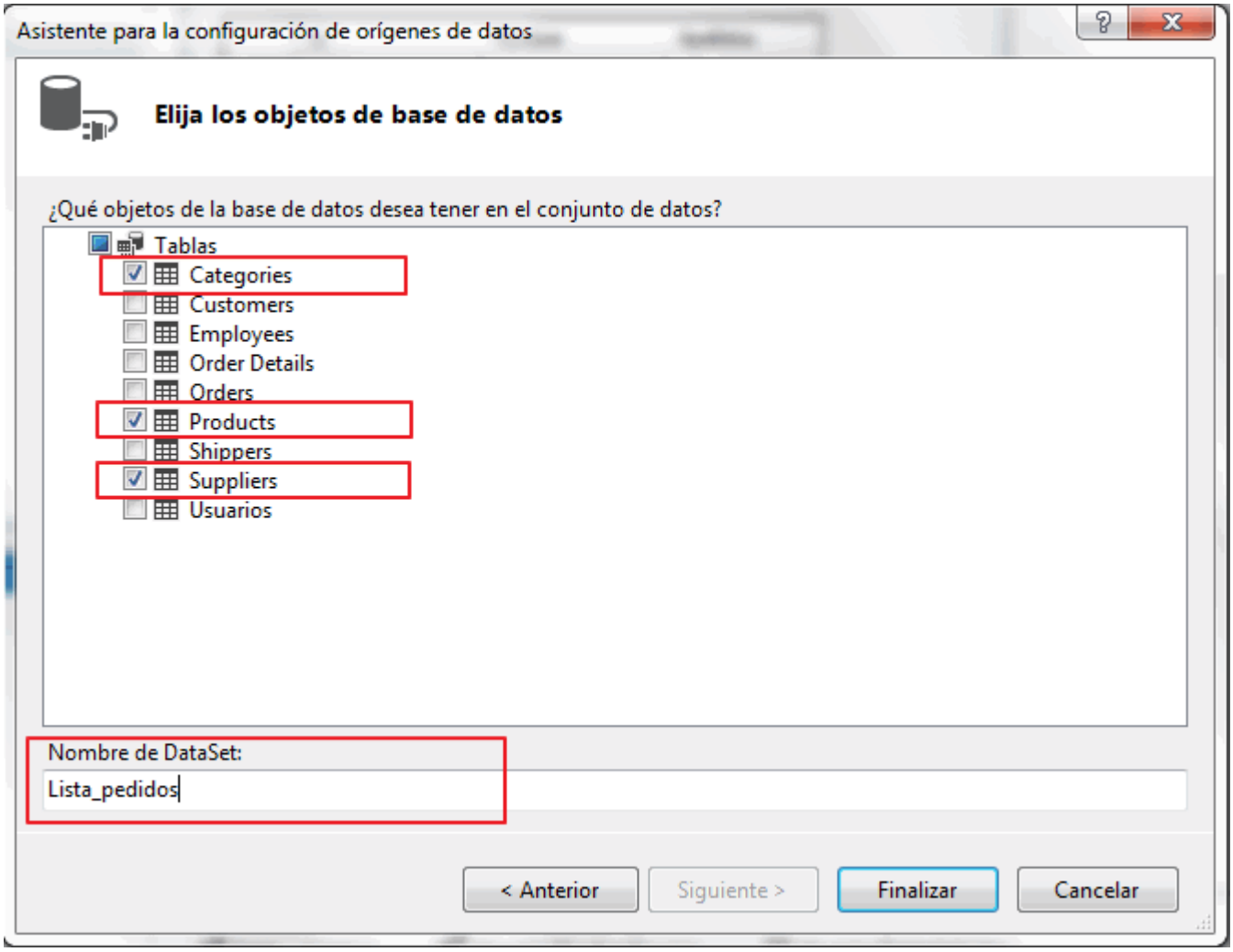
Luego le indicamos que sea de un "conjunto de datos":



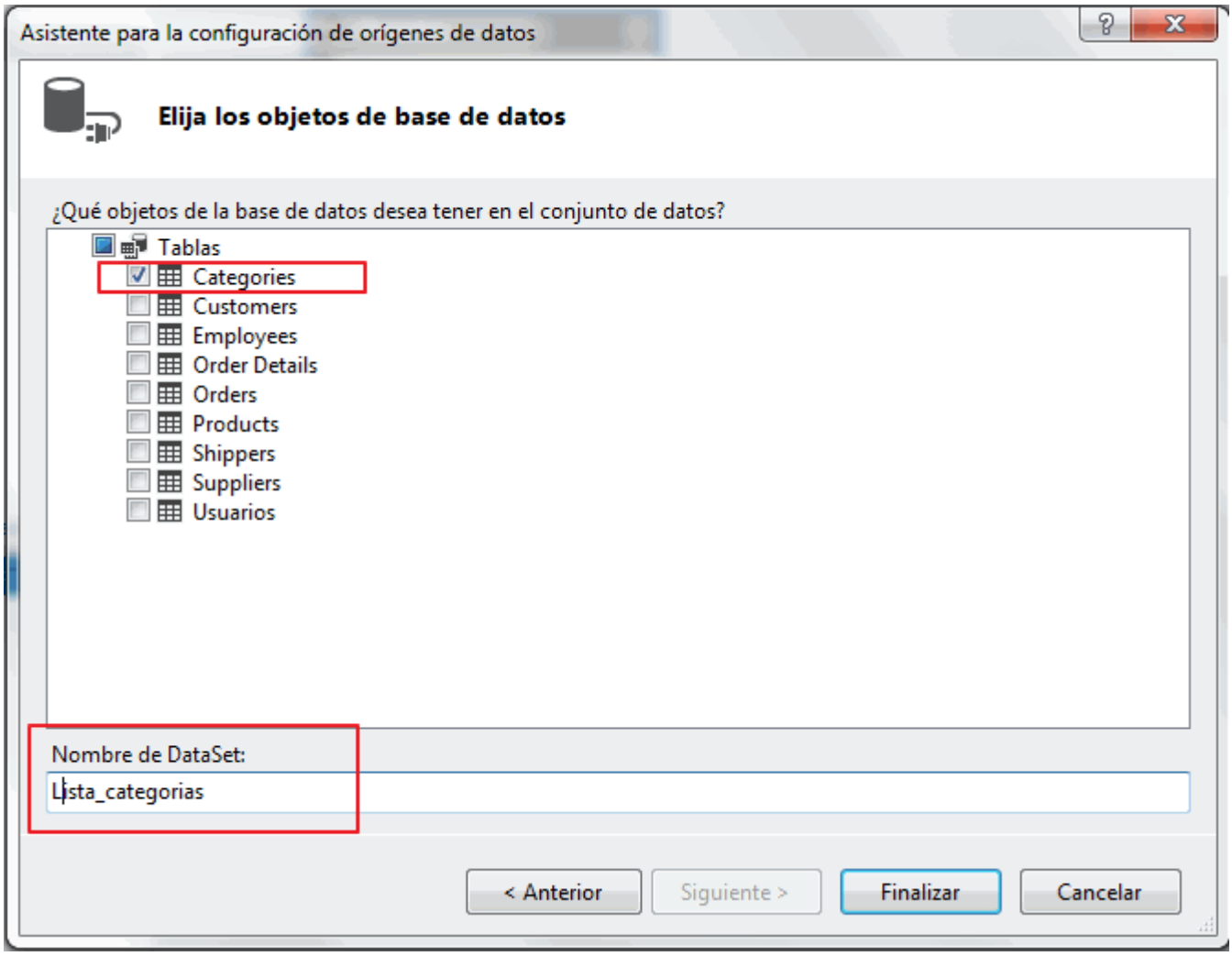
Seleccionamos posteriormente la conexión que está ya almacenada en nuestro proyecto:



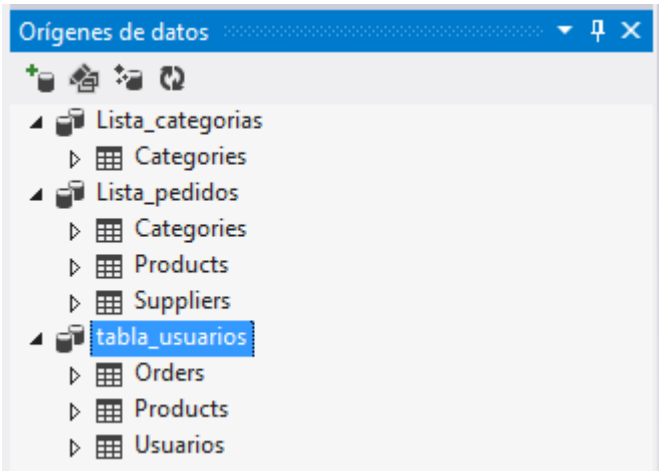
Y finalmente le decimos las tres tablas de antes:



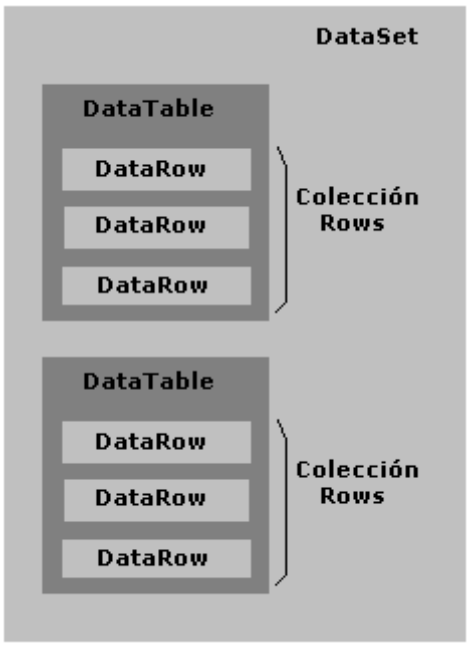
Le ponemos un nombre debajo, en el Dataset y terminamos. Ahora creamos uno más con las categorías:



Así que tendremos estos tres DataSet definidos:



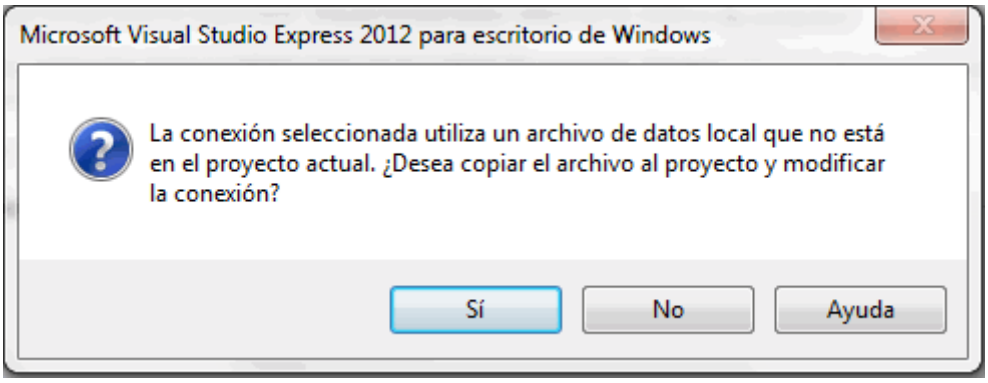
¿Recuerdas el gráfico de los objetos en los que se componía un Dataset? :



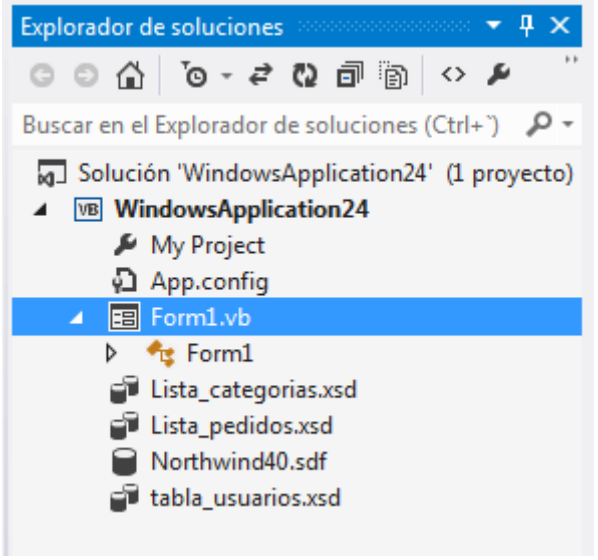
Ya ves que se parece bastante ya que hay dos que solo tienen una tabla, pero hay otro dataset que tiene tres tablas (DataTable). Y cada una de ellas los datos en colecciones de filas o "Rows".

7.1 Ubicación de la base de datos

Un detalle importante, cuando hicimos el primer vínculo con los Datasets, al crear la conexión con la base de datos, nos hizo esta pregunta:



Nosotros le dijimos que sí, por tanto se incluirá el fichero de bases de datos en nuestro proyecto. Lógicamente si se fuera a distribuir la aplicación deberíamos tener en cuenta esta y utilizar una cadena de conexión contra un fichero de un servidor o directamente contra un SQL Server. Fíjate en el explorador de soluciones como nos ha añadido el fichero de base de datos y otro más con la cadena de conexión:

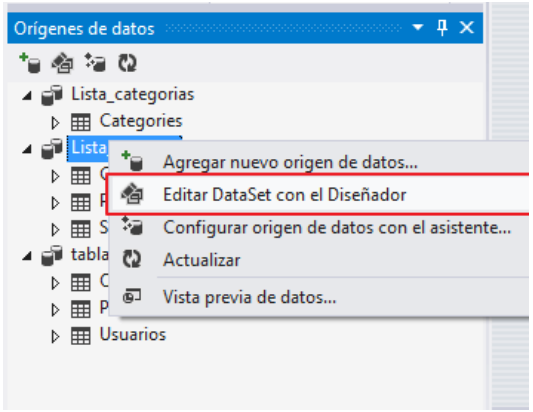


Ahora nuestro proyecto se compone de:

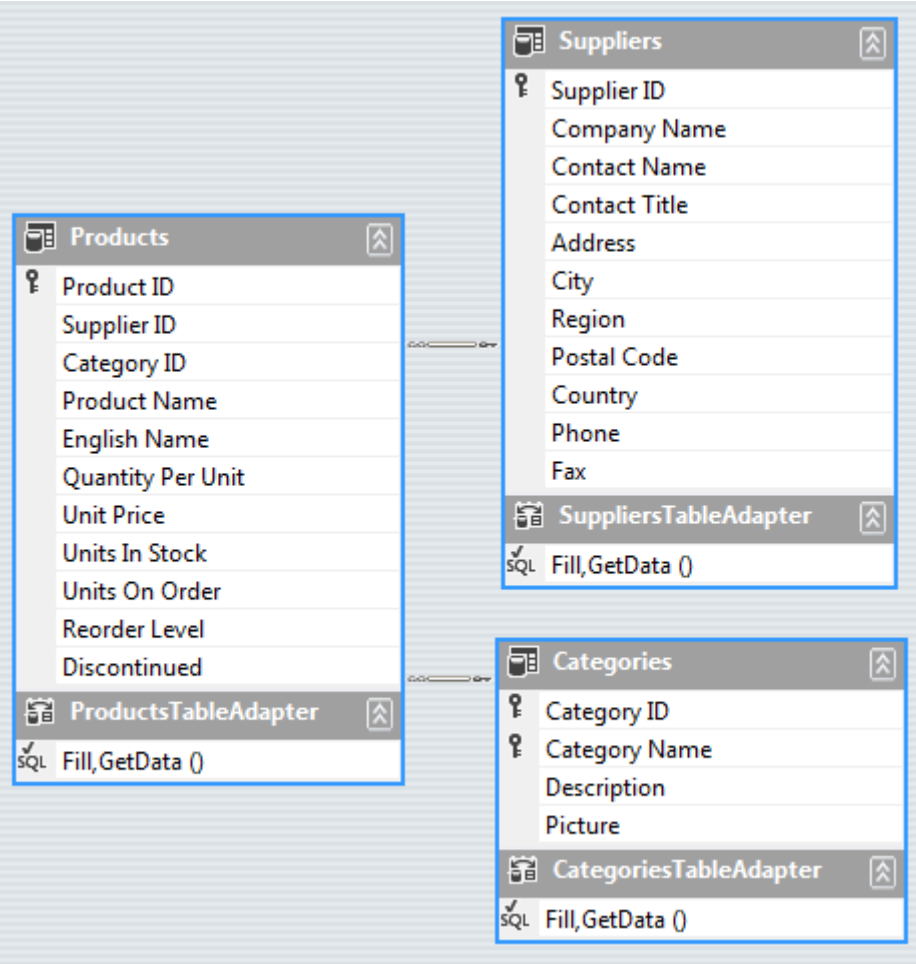
- "My Project". Fichero de configuración del entorno del proyecto: formulario de inicio, opciones como "Option Explicit", "Extrict", ...
- App.config. Fichero en formato XML con información adicional del proyecto: conexiones de bases de datos, ... si le das doble clic verás entre todas las etiquetas una con la cadena de conexión que hemos definido
- Ficheros .vb: formularios de nuestra aplicación Windows
- Ficheros .xsd: Dataset que alimentan de datos a nuestro programa
- Northwind.sdf. Fichero de base de datos que se adjunta automáticamente por haber contestado que sí en la pregunta anterior.

7.2. Utilizar los orígenes de datos

Ahora nos vamos a referir al esquema de las tablas. Editemos nuestro Dataset en el que hemos puesto las tres tablas desde el administrador de orígenes de datos:

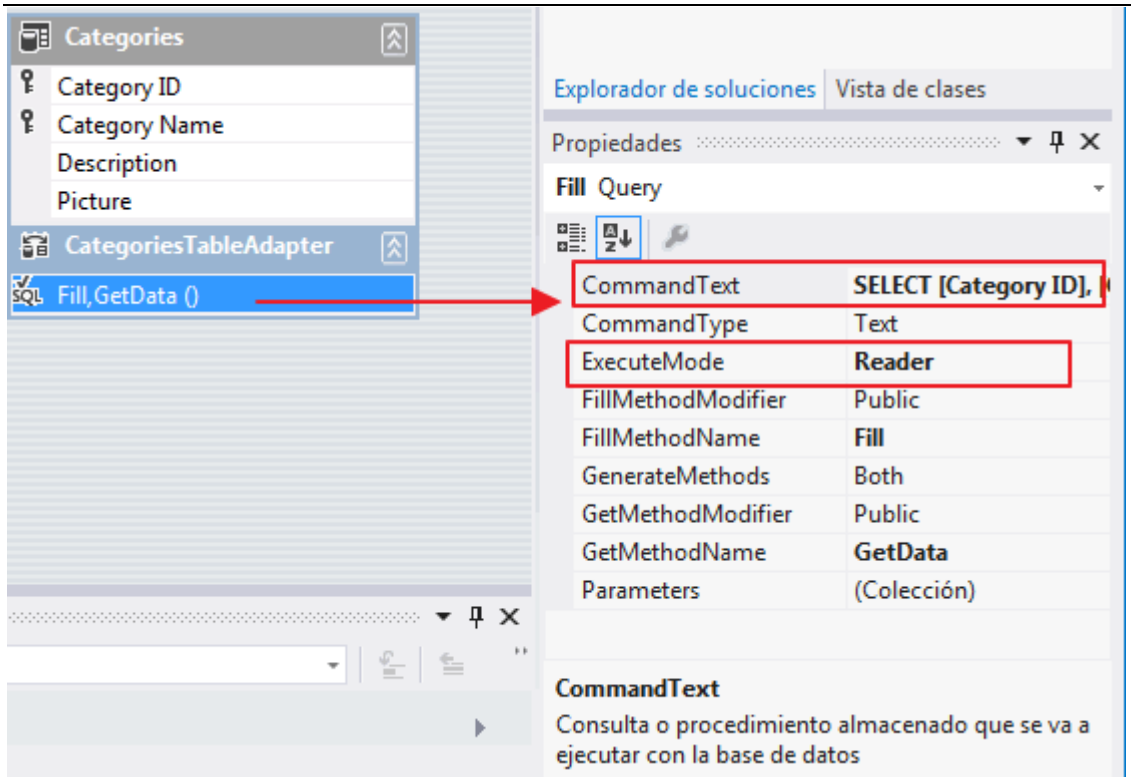


Vemos que el aspecto es parecido al del editor:

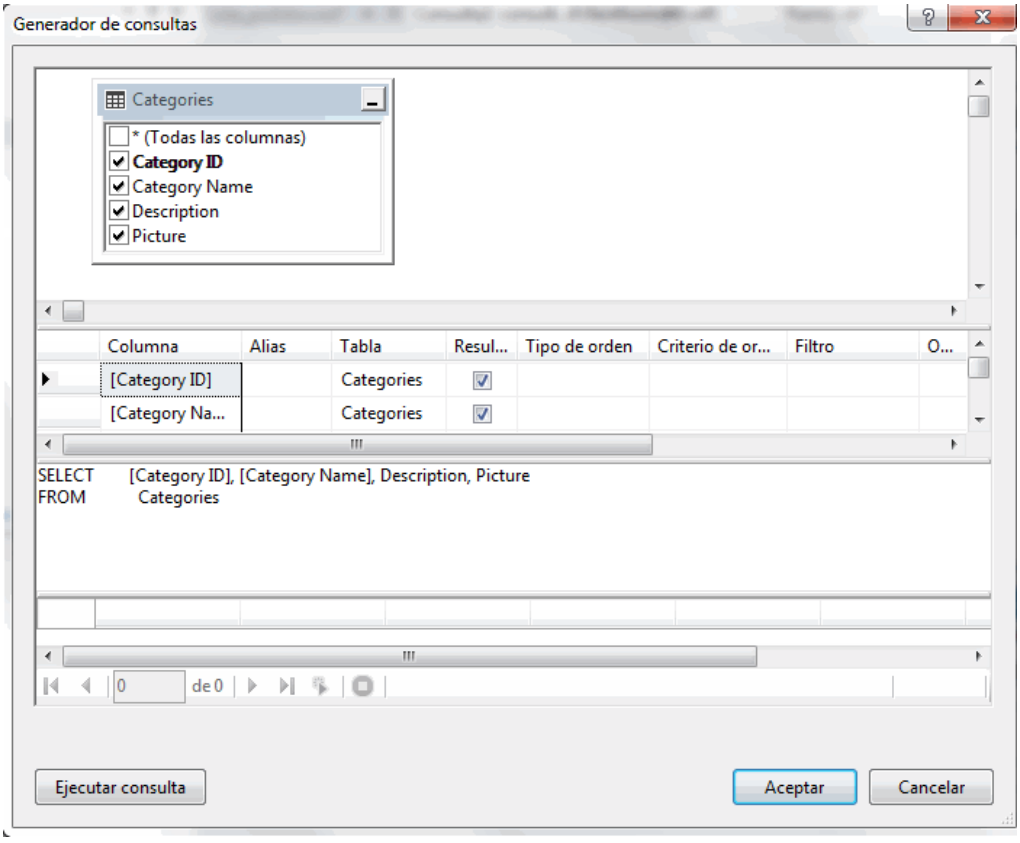


Pero no te confundas, aquí estamos editando los orígenes de datos en forma de Datasets, no las bases de datos con sus tablas. Son pues, cosas muy distintas. Fíjate que nos muestra también las relaciones y nos indica la dirección de la relación "1 a muchos" ya que aparece el símbolo de infinito en la tabla de productos. Eso significa que un proveedor puede tener muchos productos y que una categoría la pueden tener muchos productos.

Podemos ver debajo como se ha definido un adaptador para cada tabla que viene debajo como "ProductsTableAdapter", ... si hacemos clic en el icono "sql" de productos nos mostrará un "tip" con la sentencia SQL que corresponde pero también nos muestra en la ventana de propiedades:



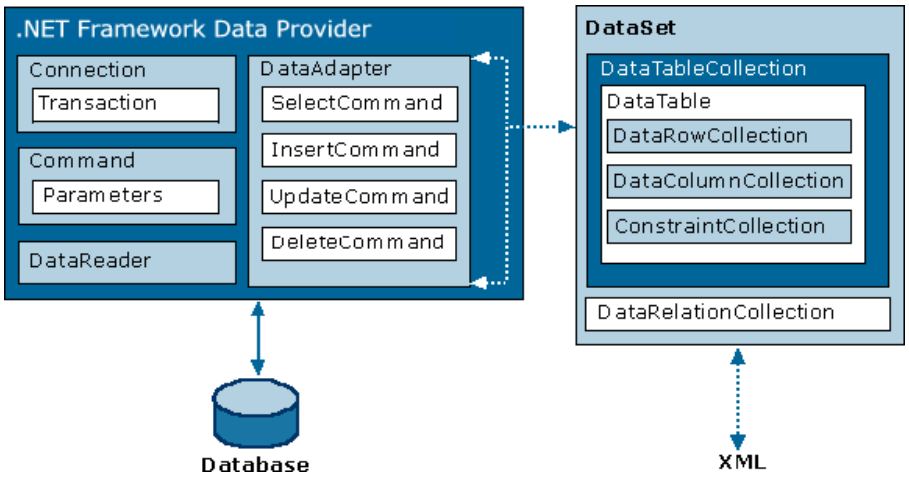
Es decir, este adaptador tiene ya definida un comando para consultar datos. Se ejecuta como un "reader", que veremos más adelante. En un CommandText tenemos la sentencia SQL asociada, que si pulsamos para ver los detalles de este campo de la ventana de propiedades nos mostrará:



Volvemos al generador de consultas que ya conocemos. Por tanto, ves que los Dataset tienen adosado imprescindiblemente un Adaptador "adapter" que es el que tiene los comandos para juntar los "dos mundos": El dataset que podemos enlazar con los controles en nuestros programas y la base de datos.

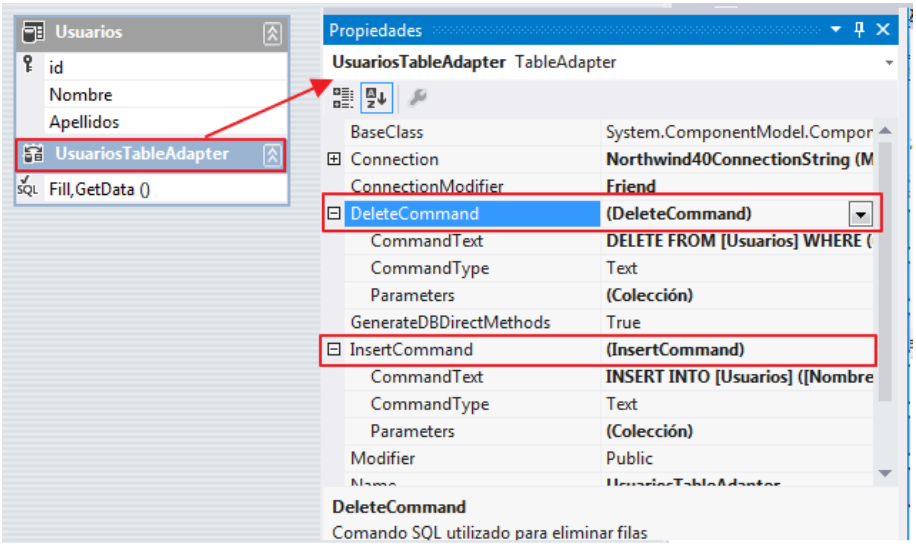
Pero aquí hay algo que falla, o por lo menos no entiendo bien... si el adaptador es el que contiene los comandos para conectar la base de datos con nuestro Dataset ¿por qué aparece sólo un comando para consultar? ¿no deberían aparecer los demás comandos de borrar (delete), modificar (update) e insertar (insert)? ¡PUES SÍ!

Veamos este gráfico:

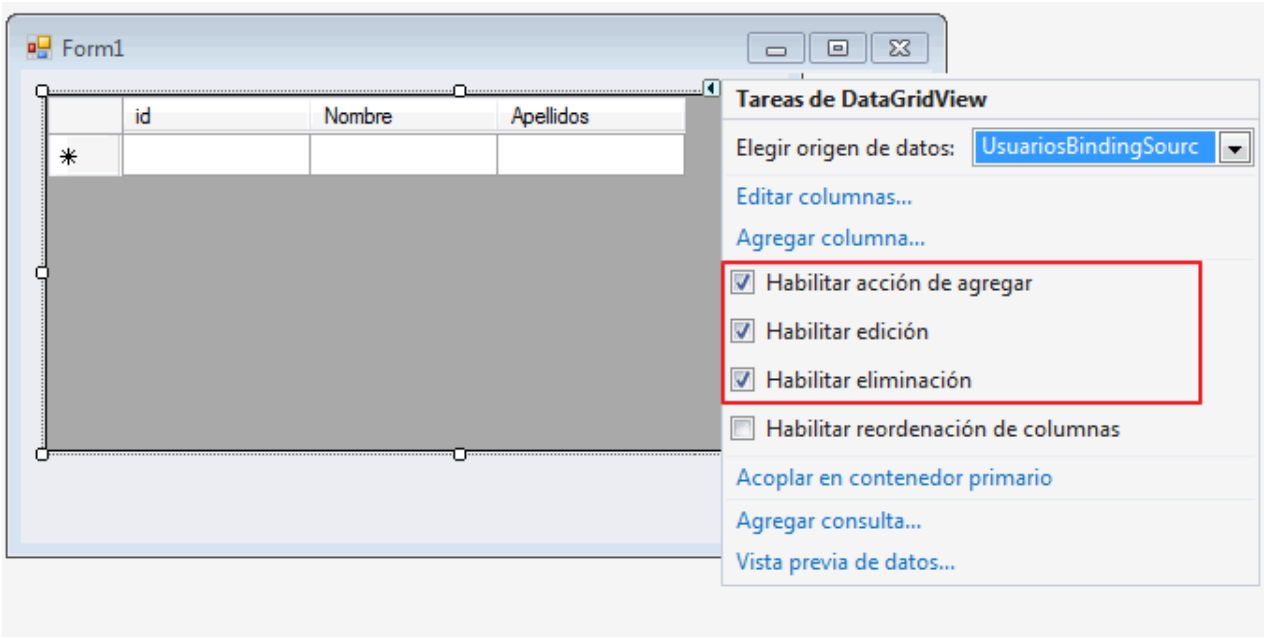


Que viene a ser lo mismo de siempre pero visto de otra forma. Vemos en la parte de la izquierda que tenemos una conexión y a su derecha el adaptador con los cuatro operaciones que puede contener y que de momento solo tenemos la de la consulta.

Si en el adaptador no tenemos todos comandos SQL correspondientes, no podremos hacer las operaciones. Recordemos el ejemplo que hicimos al principio con la tabla de usuarios. Vemos su Dataset y su adaptador:

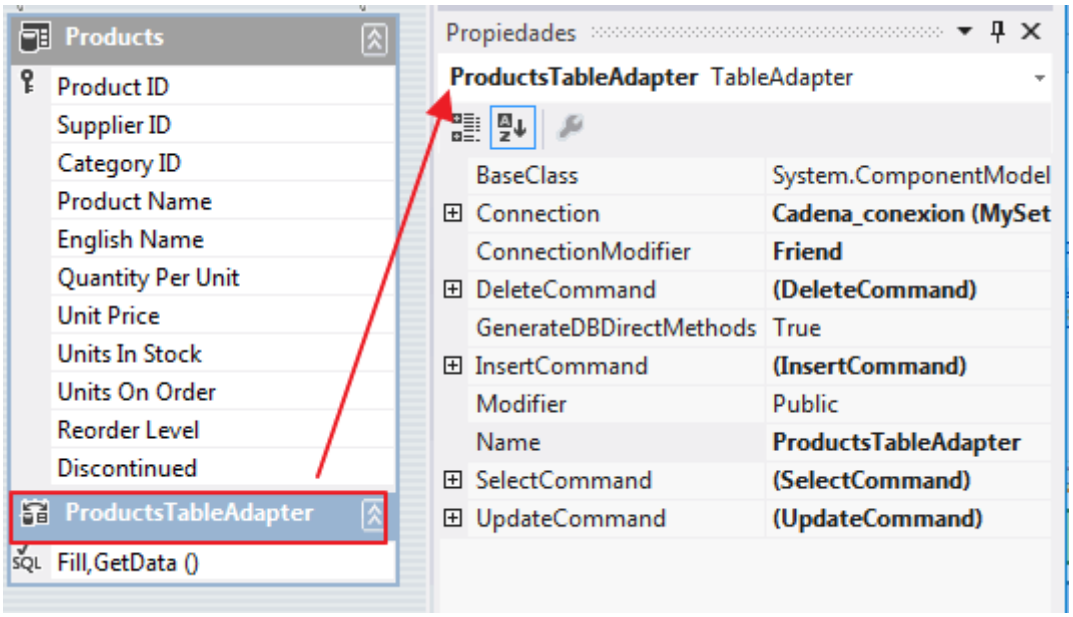


Vemos, haciendo clic en "UsuariosTableAdapter" que a la derecha nos muestra en las propiedades que, efectivamente están incluidas las instrucciones para hacer estas operaciones. ¿y por qué me las ha puesto? Pues porque cuando hicimos el enlace del origen de datos con el control de cuadrícula le indicamos que las pusiera:



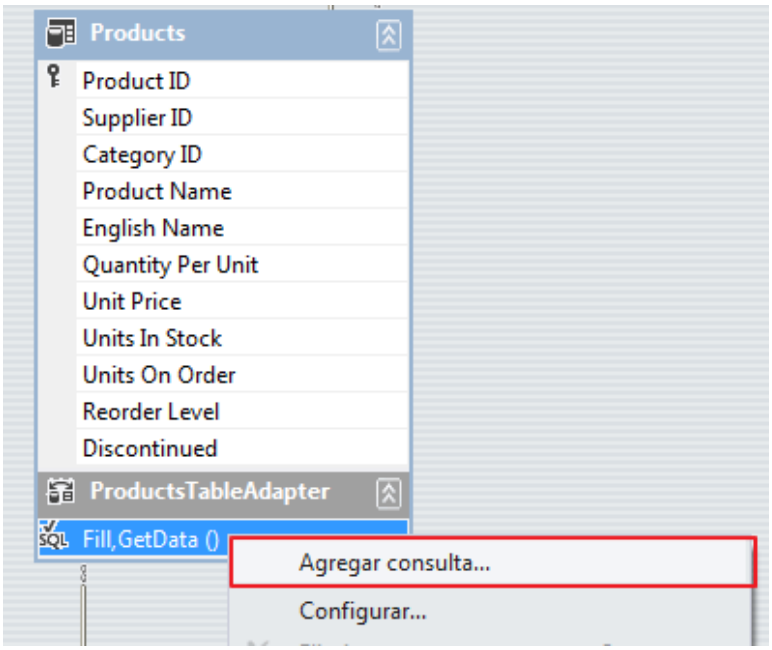
Recuerda que hemos asociado una tabla a un control DataGrid y además en el IDE le hemos dicho que se pueda editar. El precio de todo esto es crear un dataset y un adaptador con los comandos correspondiente.

Sigamos con nuestro ejemplo anterior y veamos el adaptador de la tabla de productos:

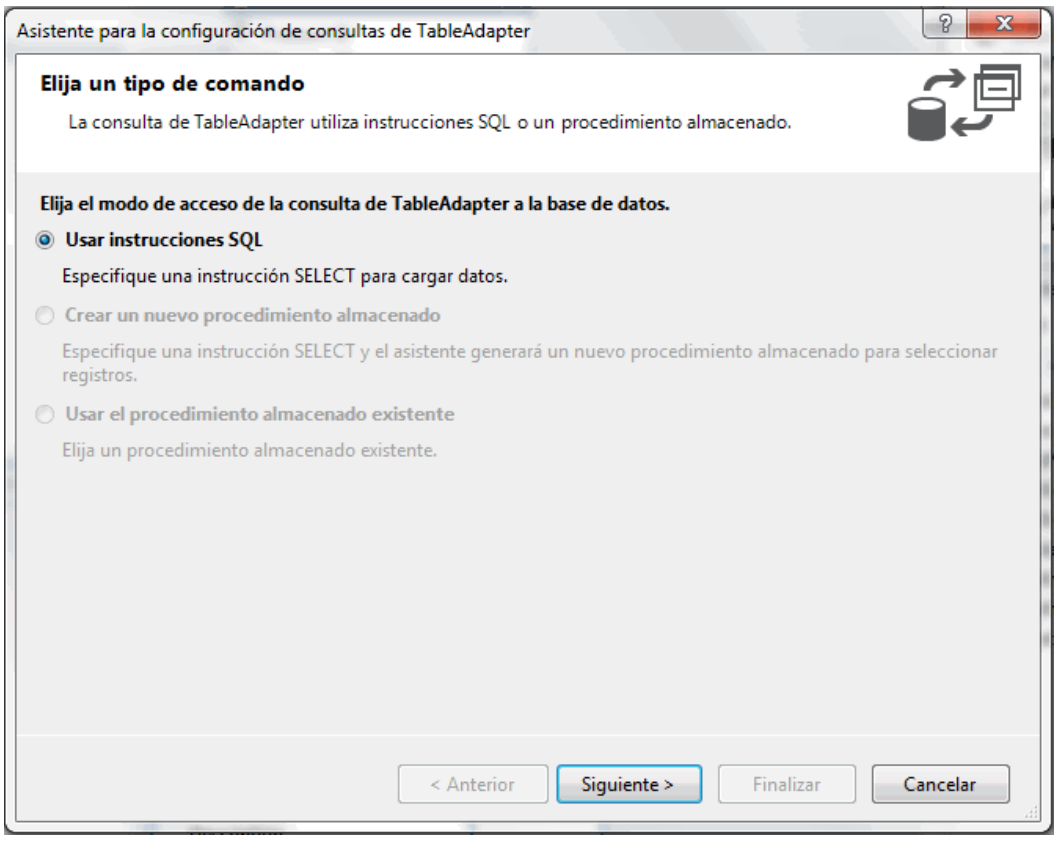


Bien, vemos de nuevo los comandos de mantenimiento de la tabla: Insert, Delete, Update y Select.

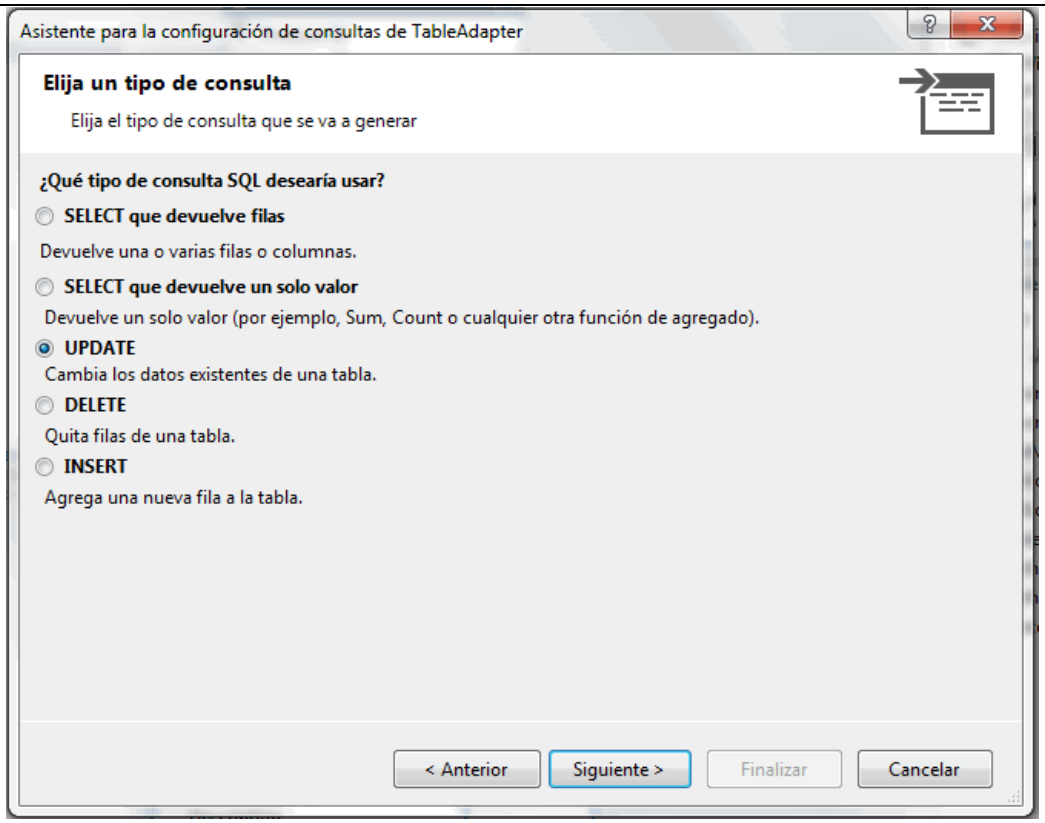
Aunque están puestos ya de forma automática, vamos a ver cómo añadirlos. Seleccionamos "Agregar consulta":



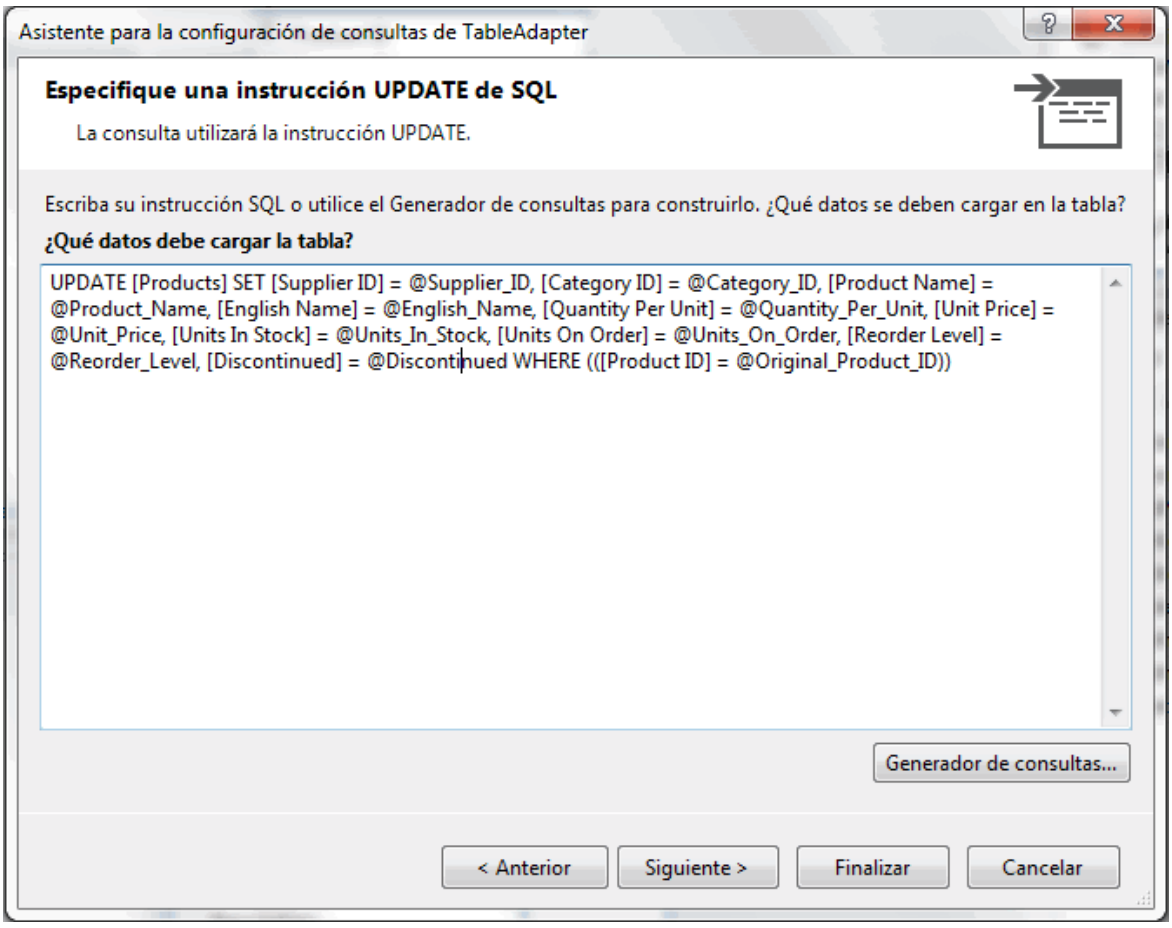
Se pone en marcha un asistente para la creación de la consulta:



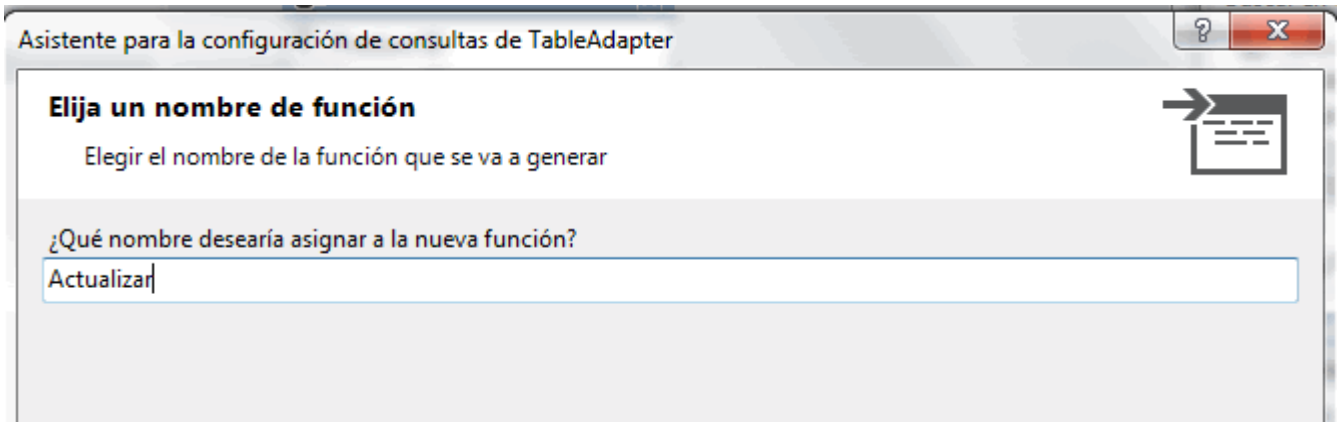
Solo nos dejar hacer esa operación. Las otras dos dependen del origen de datos, en un SQL Server completo podríamos seleccionar las otras para ejecutar un procedimiento almacenado. Pulsamos en siguiente:



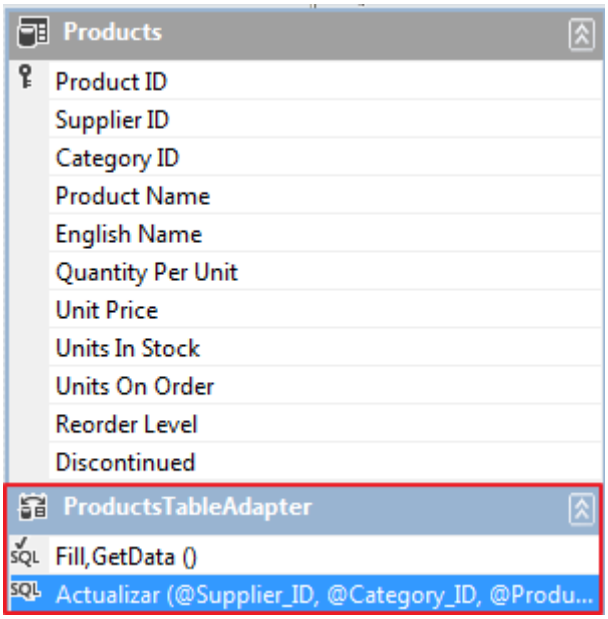
Y queremos añadir una instrucción para actualizar (Update):



Esta es la instrucción SQL para poder hacer una actualización de los datos de la tabla. Dejamos la actualización que nos pone por defecto y pulsamos para ponerle un nombre:



Finalmente veremos que ha añadido el comando a nuestro adaptador del dataset:

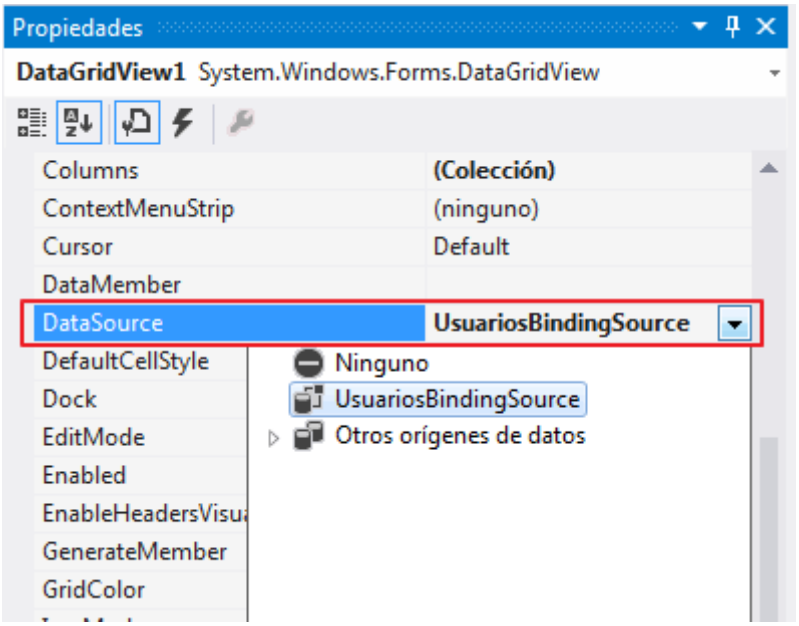


Los comandos (altas, bajas, consultas y modificaciones) ya están añadidos en el adaptador por defecto al utilizar este asistente. Lo que podemos hacer con esto último es añadir más comandos para aplicarse unos u otros en el código.

7.3. Controles enlazados a datos

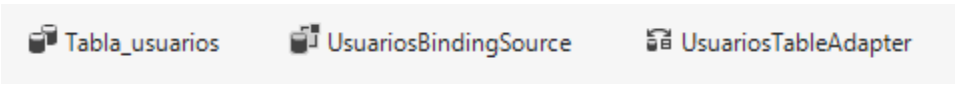
En prácticamente todos los proyectos de aplicaciones de Windows se utilizan los controles enlazados a datos. El ejemplo más sencillo lo hemos visto antes cuando con dos clics asociamos una tabla a un origen de datos. Su uso es realmente sencillo porque el "trabajo sucio" ya está realizado al tener los datos ya preparados en nuestros proyectos gracias a los "datasets".

Para realizar este enlace utilizaremos la propiedad "DataSource" del control. Podremos seleccionar luego lo que queremos enlazar...; por ejemplo, en una cuadrícula como la que ya vimos tenemos la propiedad:



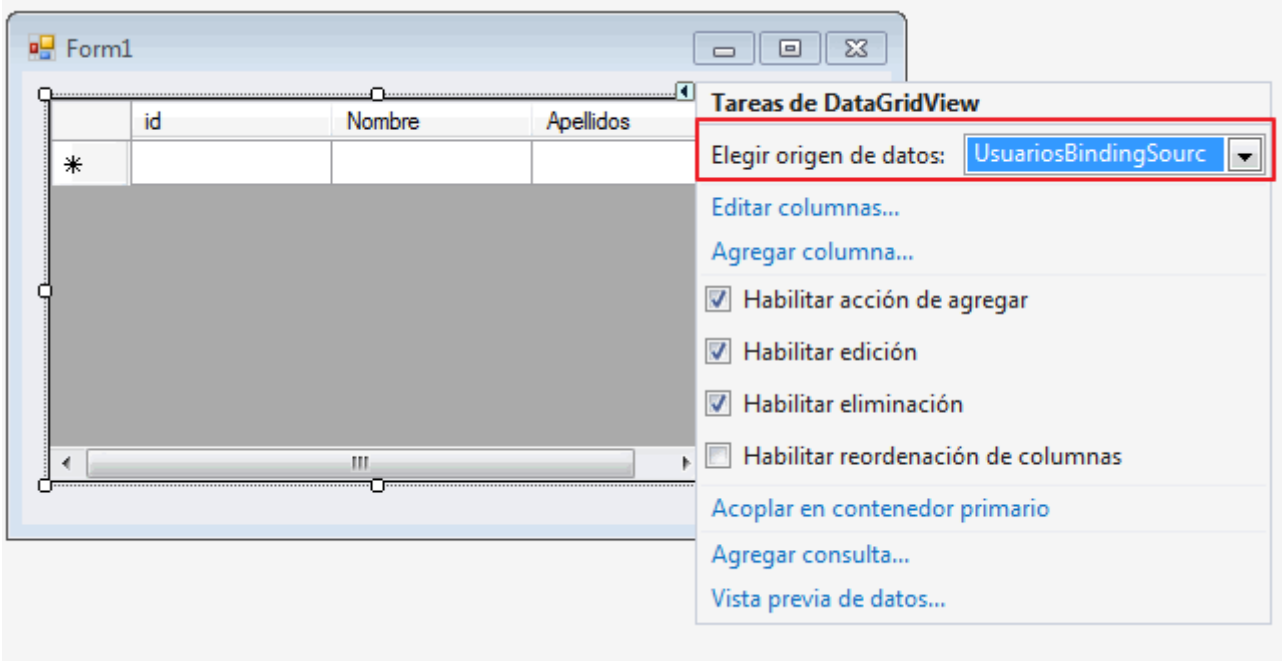
Es decir, se trata de la conocida propiedad "DataSource", a la que se puede acceder desde el menú rápido que aparece junto al control o desde la ventana de propiedades. Como vemos nos indica que al seleccionar un origen de datos se crea una instancia en el proyecto y se enlaza a través de un nuevo "BindingSource". Por eso vimos antes una serie de controles añadidos al programa para poder hacer funcionar todo.

Visual Basic crea entonces automáticamente un componente "BindingSource" y lo añade al formulario, que verás coloca en la lista de controles no visibles:



Esto es uno de los grandes avances ya que antes había que realizar varios pasos antes de tener nuestro enlace de datos preparado.

El principal control enlazado a datos es el ya conocido DataGridView que es nuevo. En versiones anteriores teníamos el DataView y el DataGrid que eran inferiores en prestaciones. Las opciones más importantes las controlaremos desde el menú rápido que ya conoces:

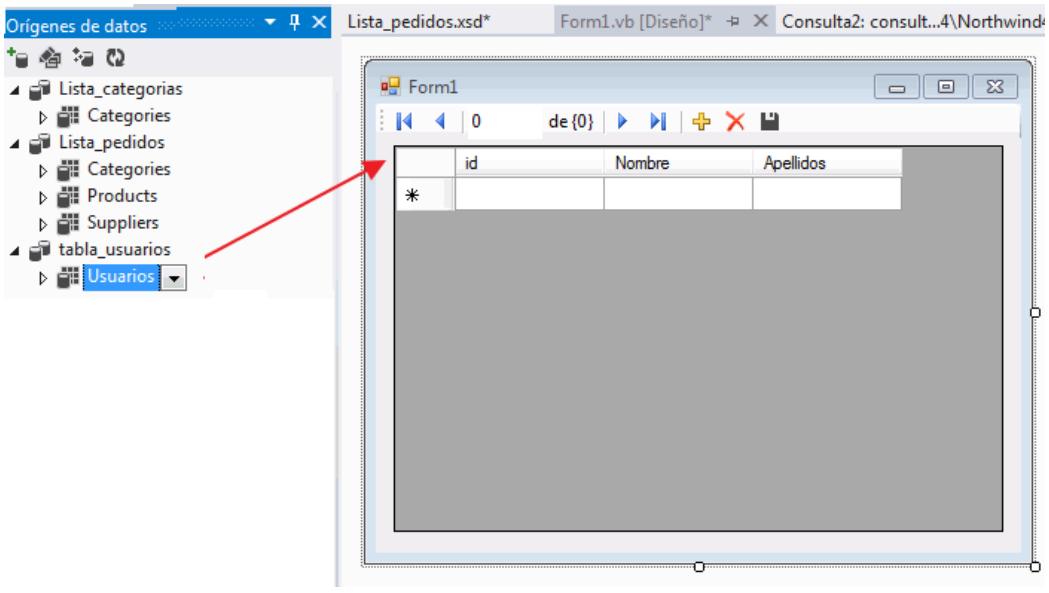


Podemos decirle las funcionalidades que queremos añadir para poderse editar y añadir filas.

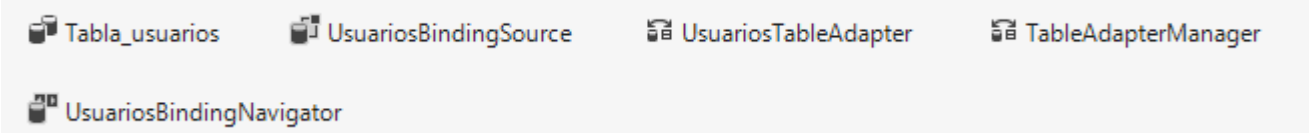
Más sobre el enlace de datos

A pesar de que hemos visto que es realmente fácil enlazar los datos con el sistema anterior, hay todavía una forma más sencilla. La segunda forma es utilizar unos formularios ya existentes para la explotación de los datos. Solo necesitamos arrastrar una tabla o un campo a nuestro diseñador. Visual Basic generará automáticamente un control basado en el tipo especificado en la ventana de orígenes de datos. Automáticamente hará los enlaces con el Dataset y TableBindingSource creándolos si es necesario.

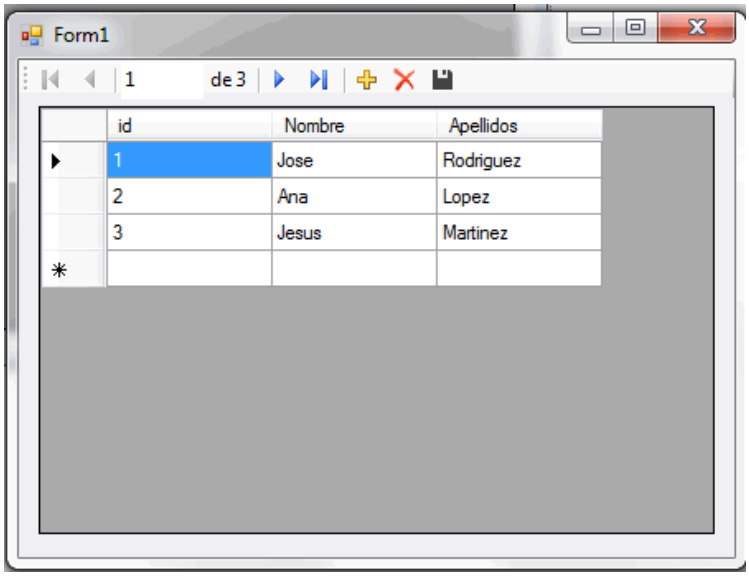
Coge un "dataset" cualquiera de los que tenemos en la lista de orígenes de datos y arrástralo a un formulario vacío. Veremos que nos coloca todo esto:



Ha colocado como unos controles de navegación en la parte superior y además ha puesto todos estos controles ocultos:



Ejecutamos el proyecto y podemos ver que tenemos todas las funciones activas y podemos realizar cualquier operación con la base de datos:



Bueno, parece que todo coincide con lo que hemos visto en la teoría. Ha creado el adaptador y todos los comandos para poder hacer todas estas operaciones. La novedad la tenemos en el control de navegación que aparece arriba y que es de tipo "BindingNavigator": "UsuariosBindingNavigator"

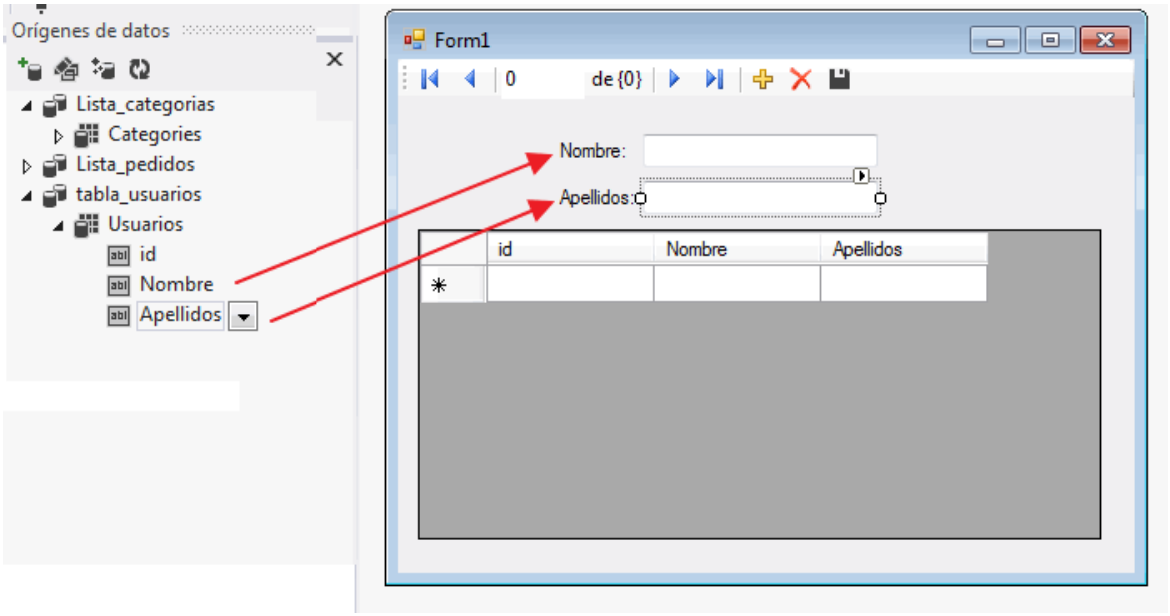
Veamos el código que ha generado dentro del formulario:

```
public Form1()
{
    InitializeComponent();
}
1 referencia
private void usuariosBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    this.Validate();
    this.usuariosBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.usuariosDataSet);

}
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    // TODO: esta línea de código carga datos en la tabla
    // 'usuariosDataSet.Usuarios' Puede moverla o quitarla según sea necesario.
    this.usuariosTableAdapter.Fill(this.usuariosDataSet.Usuarios);

}
```

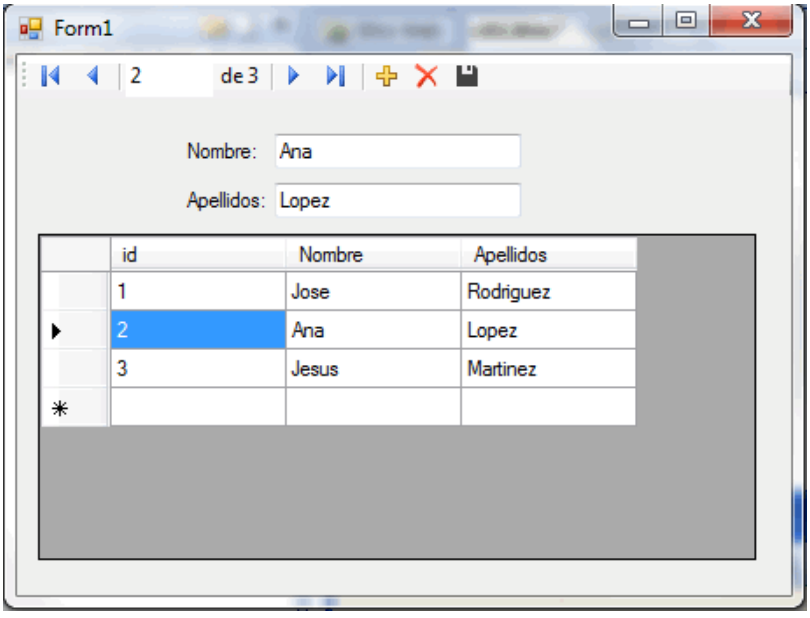
Un procedimiento para rellenar los datos (Load) y otro para almacenar los datos al pulsar el botón de guardar (UsuariosBindingNavigationSaveItem). Luego como vemos, los Dataset cumplen la definición de que están desconectados, podemos modificarlos, añadir, borrar pero hasta que no hagamos un "Update" del adaptador no se enviarán los datos al servidor.

Si aun así no estamos contentos con lo que nos ha generado podemos cambiar los controles que ha utilizado para mostrar los datos. La gran novedad que tenemos en este formulario es que nos ha puesto un control nuevo en la parte superior que es del tipo "BindingNavigator" que proporciona herramientas para edición y navegación de datos.

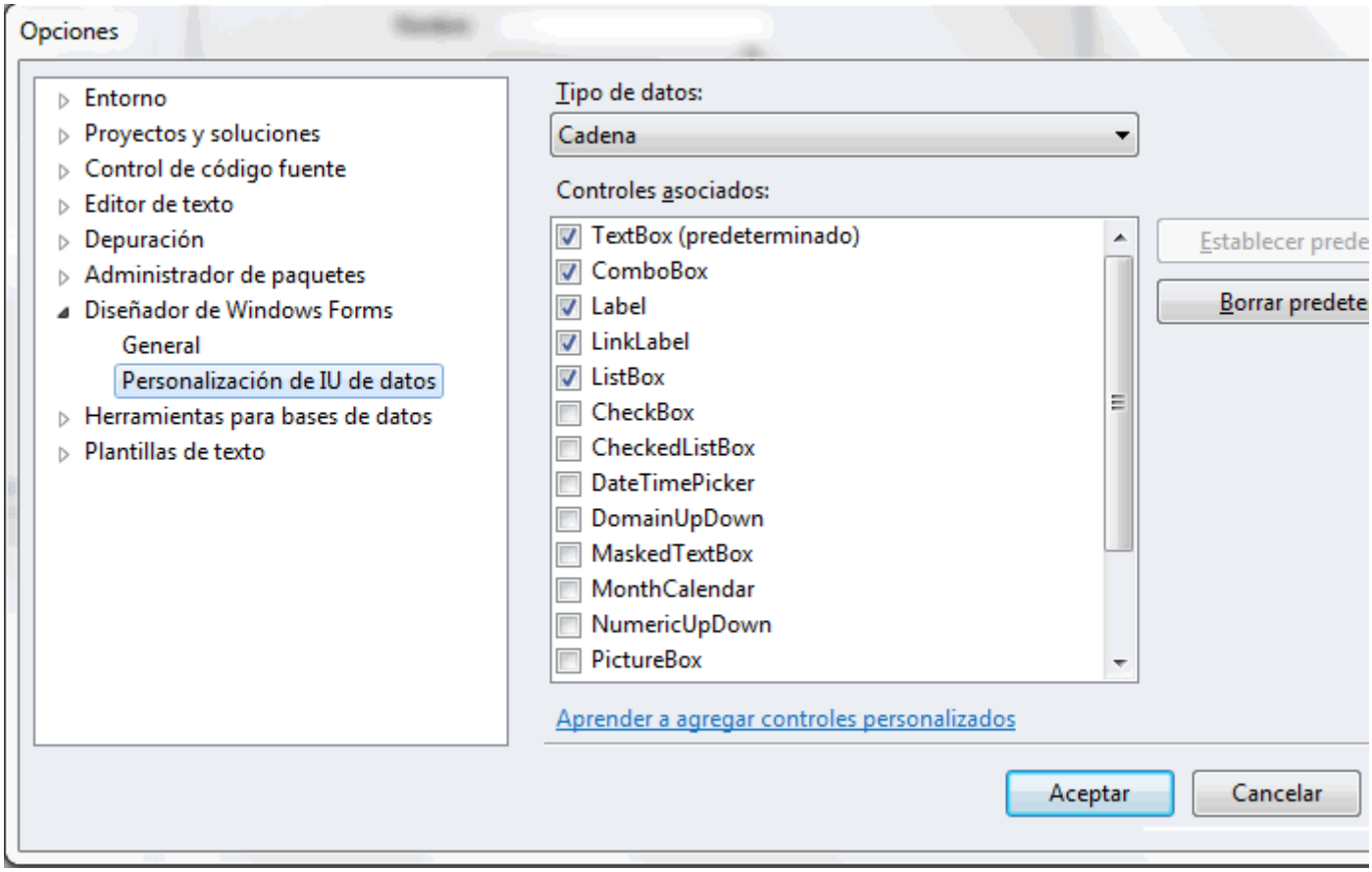
Veamos otro ejemplo. Sobre lo que ya tenemos, arrastra individualmente los dos campos de nombre y apellidos:



Así que si ejecutamos ahora:

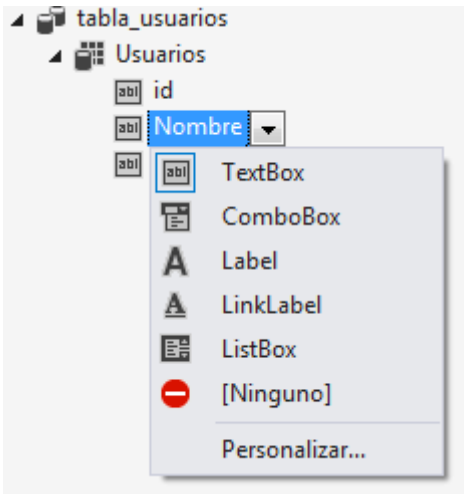


Vemos que todo está perfectamente sincronizado. Pero aún tenemos cosas más curiosas todavía, si nos vamos al menú de opciones del IDE tenemos:

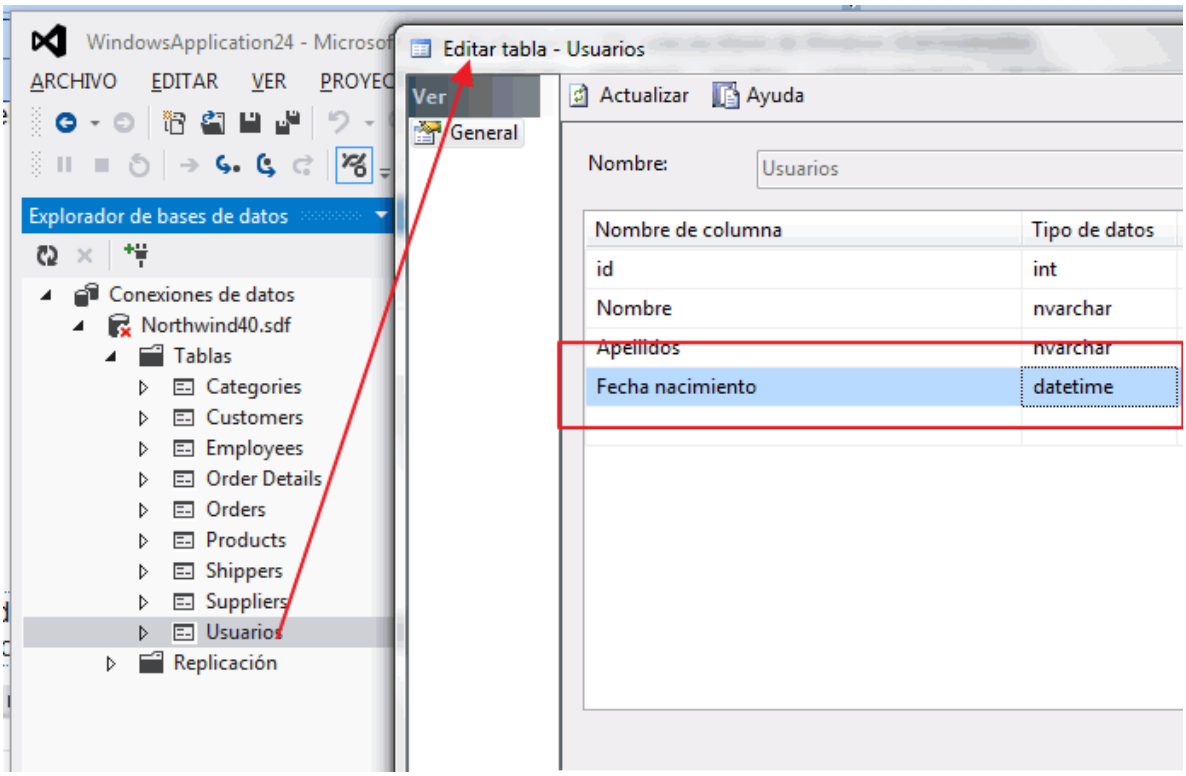


En esta pantalla podemos definir los controles que nos va a poner automáticamente dependiendo del tipo de datos que tengamos. Esto es, si en la base de datos el campo es de tipo fecha, al arrastrar el campo, lo podemos asociar con un control de tipo fecha, por ejemplo.

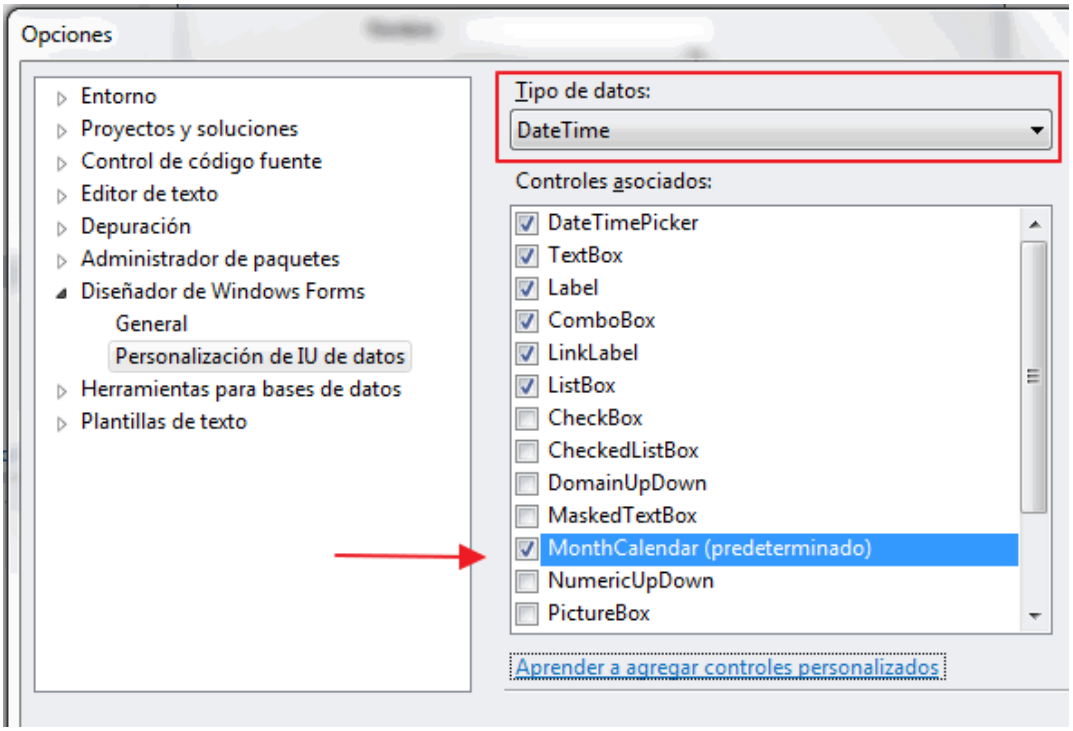
Antes de arrastrar los campos podemos indicarle el control que queremos utilizar para la edición, seleccionándolo de:



Es decir, en el cuadro desplegable le podemos indicar el control para el campo según el tipo de datos, y estos controles son los que podemos definir. Por ejemplo, vamos a añadir un campo de tipo fecha a nuestra tabla desde el "Explorador de bases de datos" de Microsoft SQL Server Management Studio, no desde el cuadro de las conexiones:

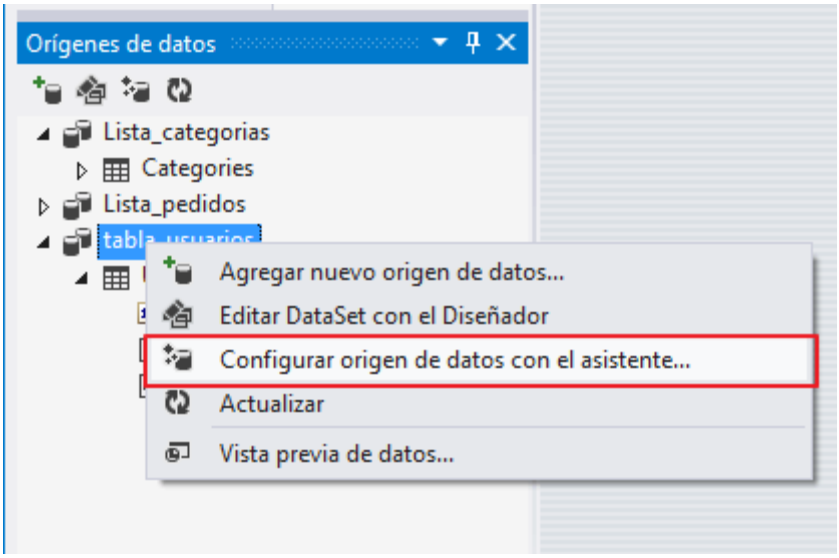


Ahora en las opciones de los campos, de vuelta a Visual Studio, veamos que tenemos para los campos de tipo fecha:

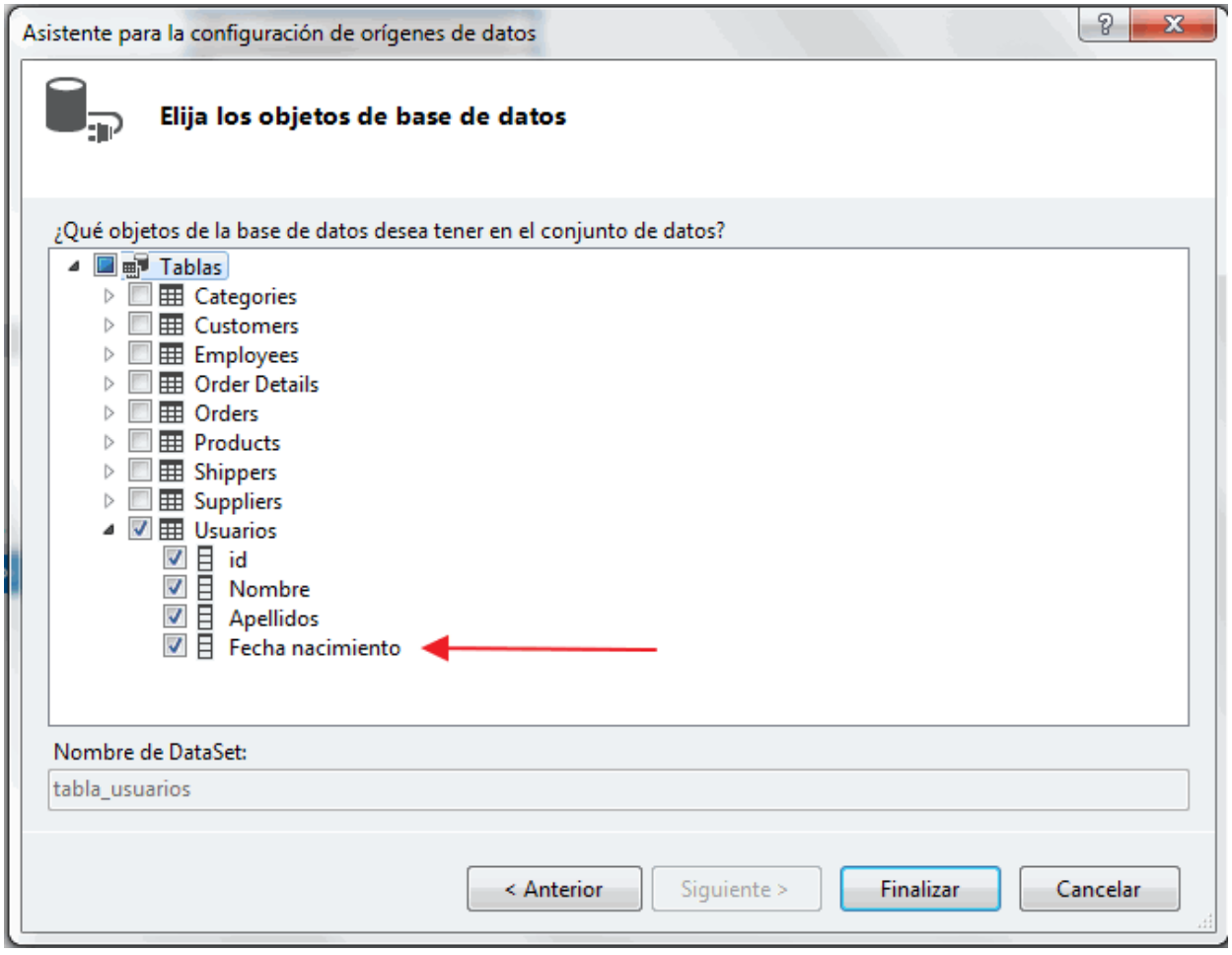


Bien, teníamos como predeterminado el "DateTimePicker" que es el de calendario desplegable. Es el mejor para estos casos pero le vamos a poner también el de la vista del mes completo y lo marcamos como predeterminado.

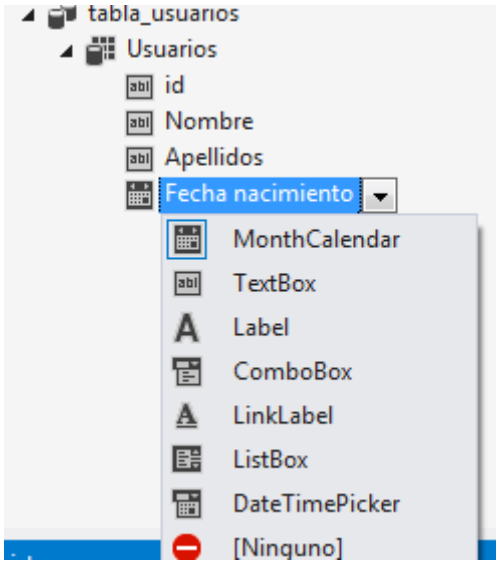
Como hemos añadido un campo a la tabla debemos actualizar el Dataset:



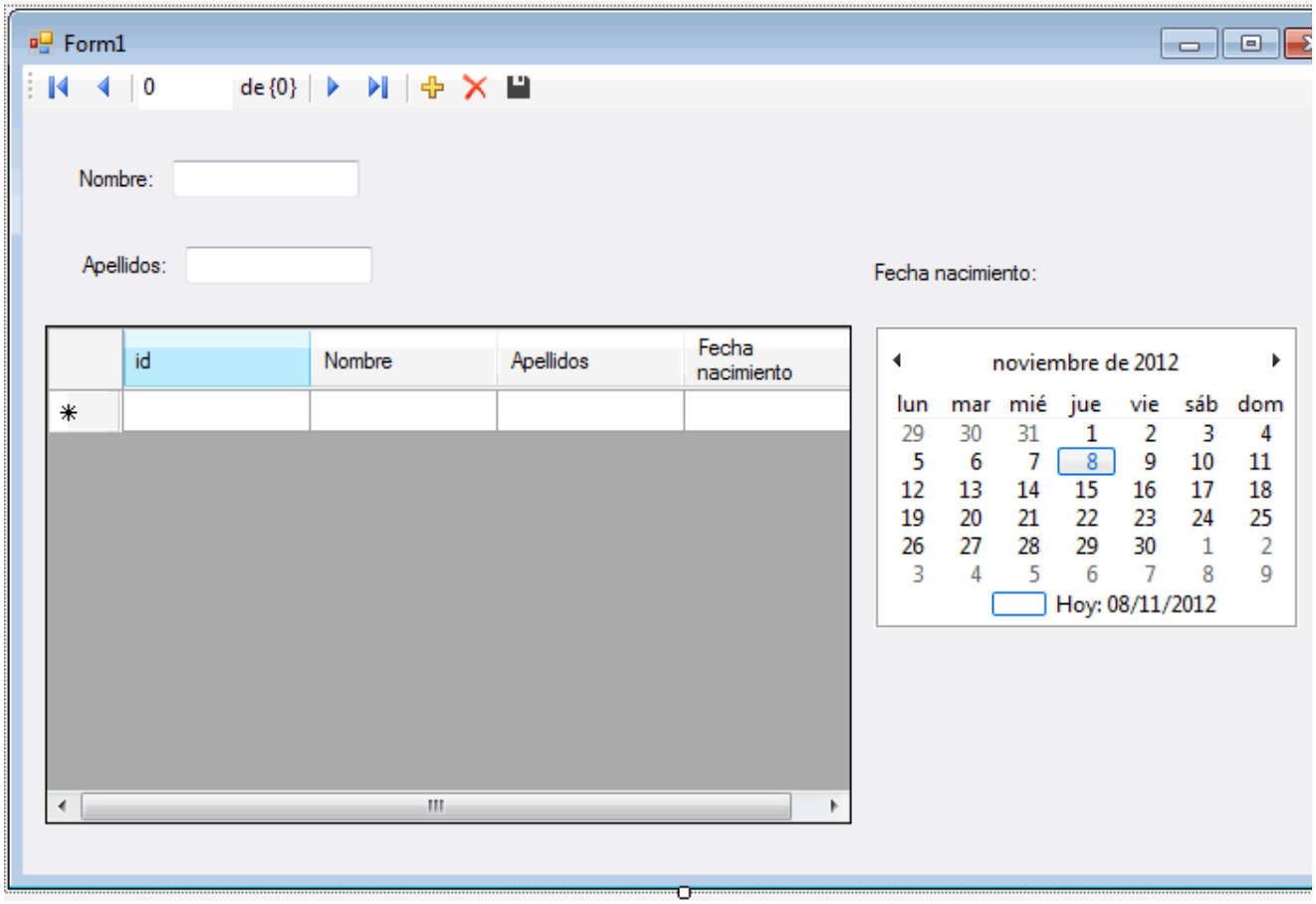
Expandimos y marcamos el nuevo campo para que lo incluya en el método "Fill" del "Dataset":



Donde podemos seleccionar la tabla con el nuevo campo, que arrastraremos

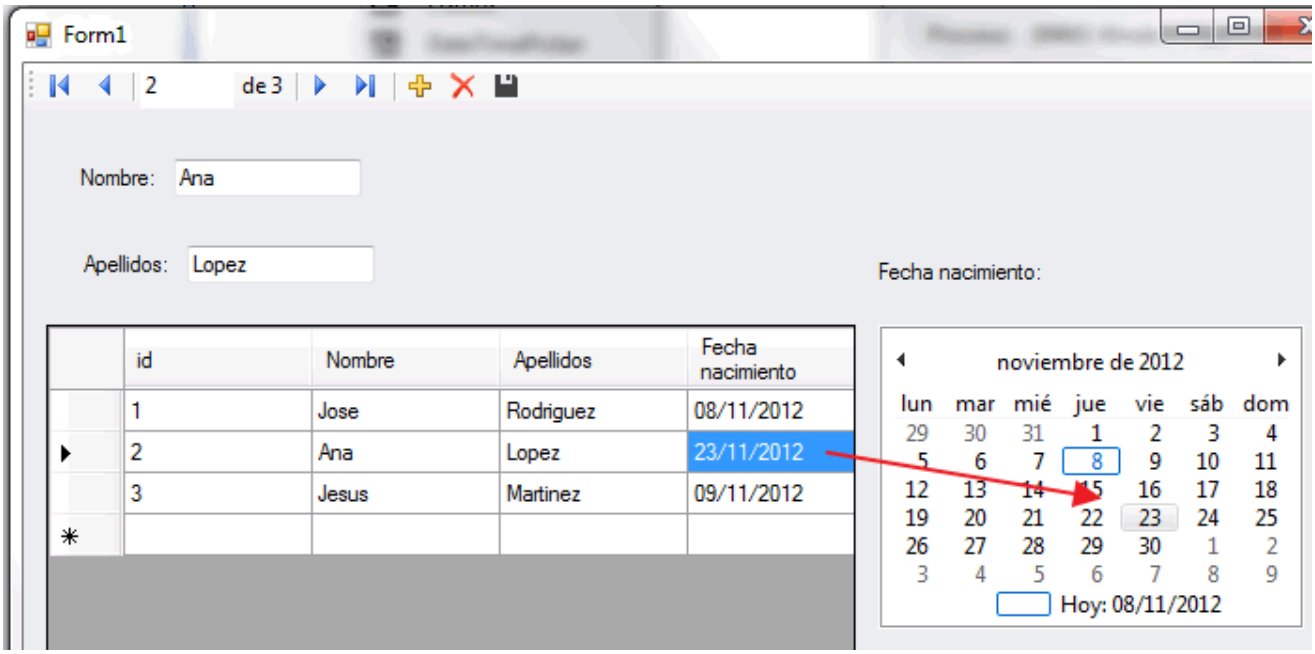


Hasta la posición que queramos. Por ejemplo:

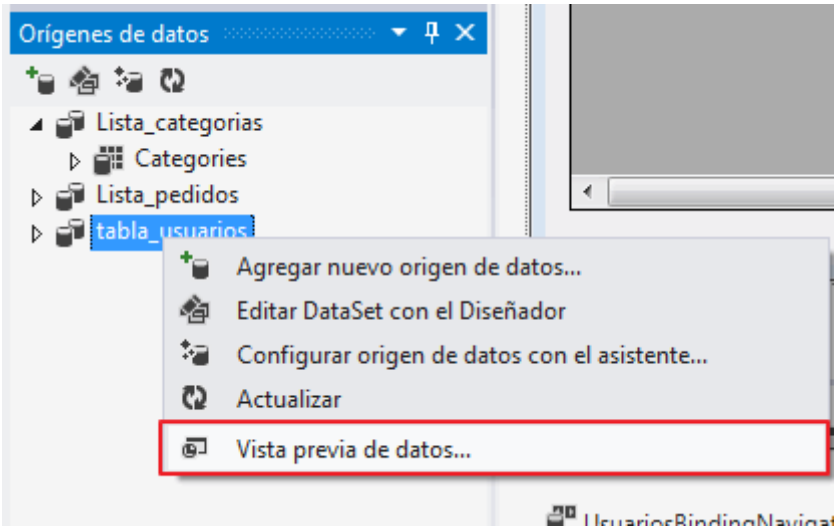


Como hemos variado los campos, lo mejor es volver a colocar los controles de un momento, eliminando también los controles ocultos de enlaces a datos. Ahora funcionará con el nuevo

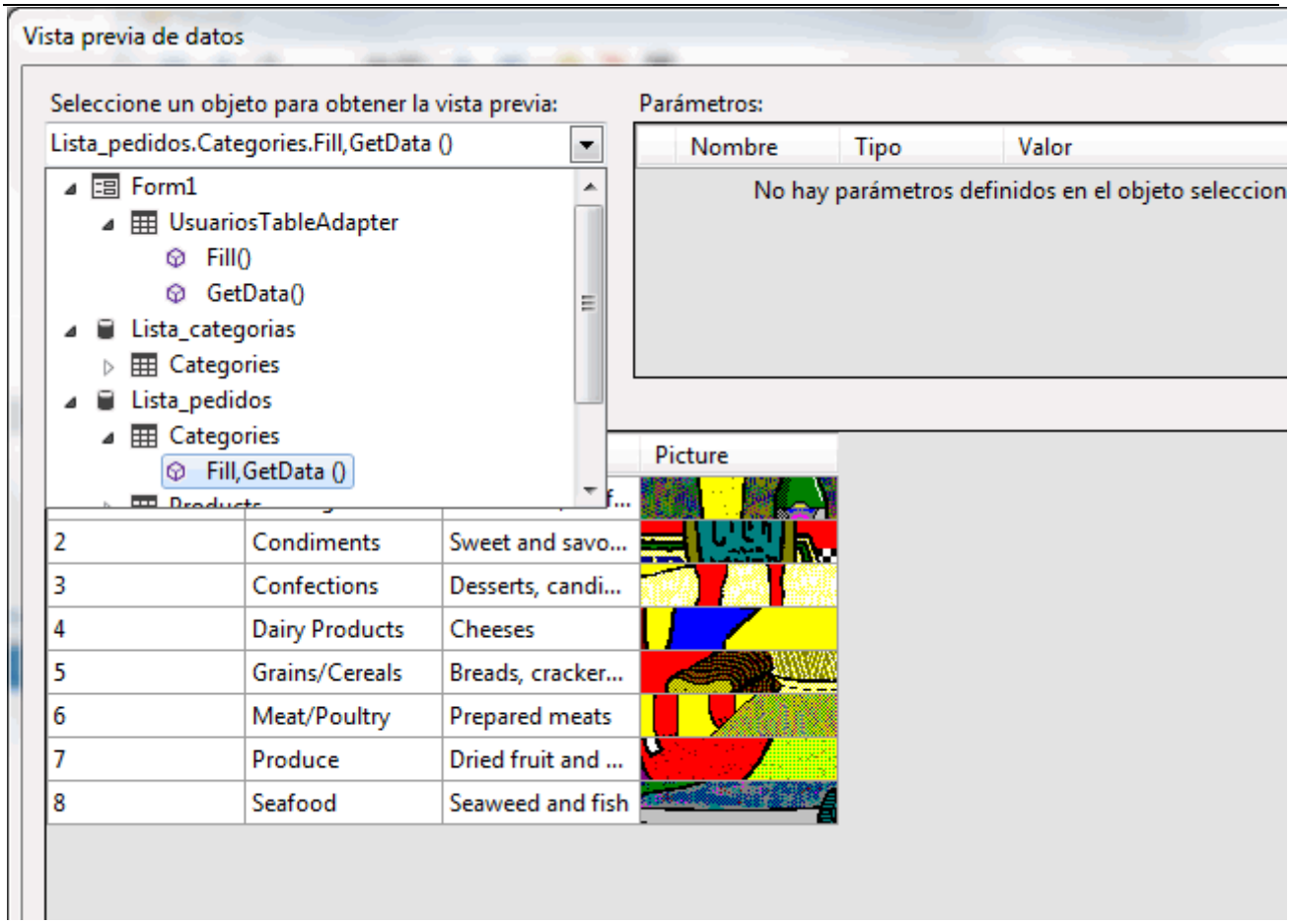
control y su tipo de datos de fecha totalmente operativo. Si seleccionamos una fecha en un registro y le damos al botón superior de grabar, se almacenarán los datos:



Como ayuda, tanto en el explorador de la base de datos como de los "Dataset", disponemos de una opción para hacer una vista previa de los datos, es decir, una simulación de la ejecución de la sentencia SQL del adaptador y el método "Fill" en una tabla de ejemplo:



Donde como ves:



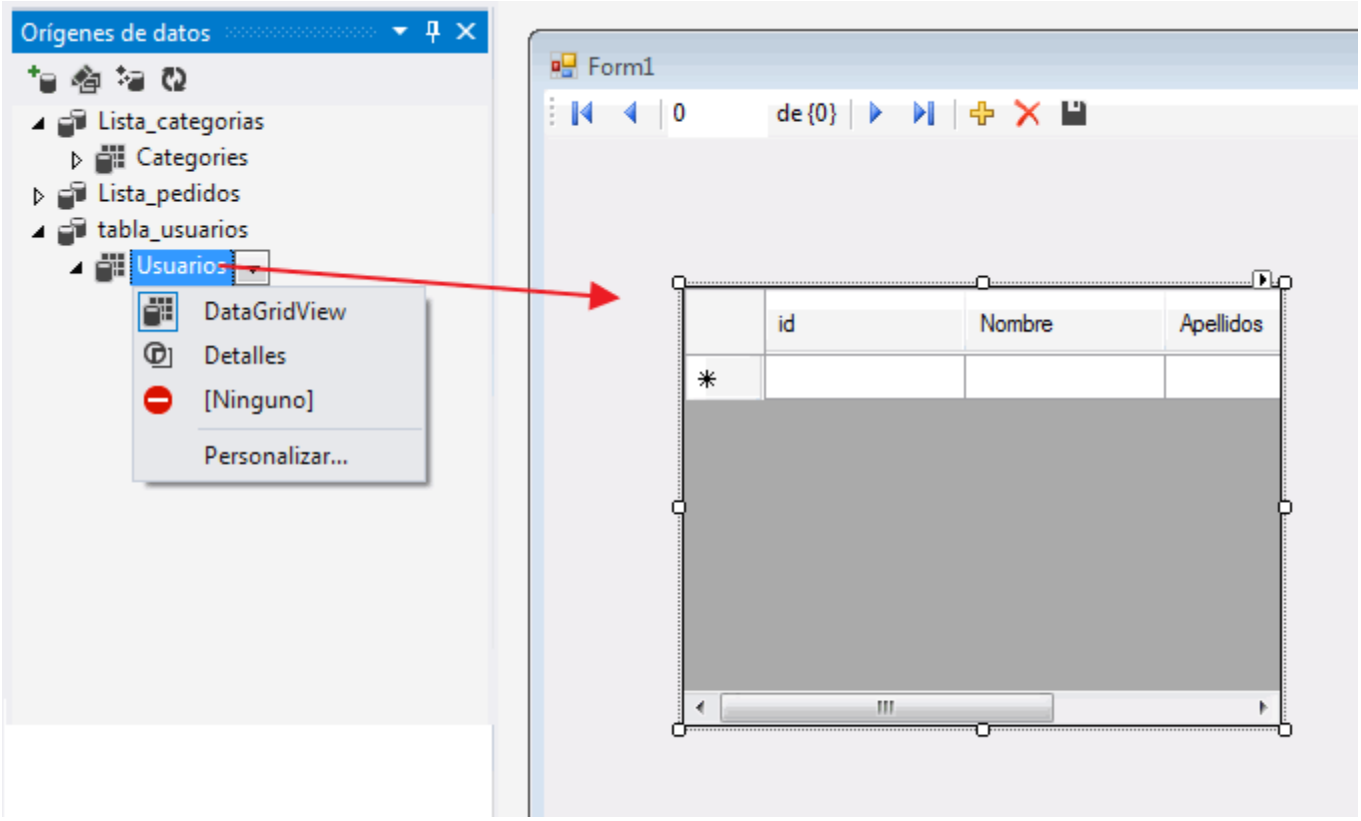
Se ejecuta el método Fill para rellenar esa cuadrícula de prueba. En ese ejemplo, la tabla de categorías tiene un campo gráfico, así que le asocia la vista previa con un "picturebox" para que muestre el contenido.

Con lo que hemos visto aquí podemos realizar multitud de operaciones automáticas pero debemos aprender el código para poder personalizar más nuestras interfaces.

Enlace con dos controles

Acabamos de hacer un enlace de la tabla completa con un DataGridView y con unos campos personalizados. Vamos a realizar lo mismo pero de una forma más automática. Creamos de momento un formulario nuevo y seleccionamos en el origen de datos el DataGridView como control para el formulario:

Una vez seleccionado el DataGridView seleccionamos la tabla "usuarios" y la arrastramos al formulario, donde nos creará todo lo necesario para que este Dataset sea operativo:



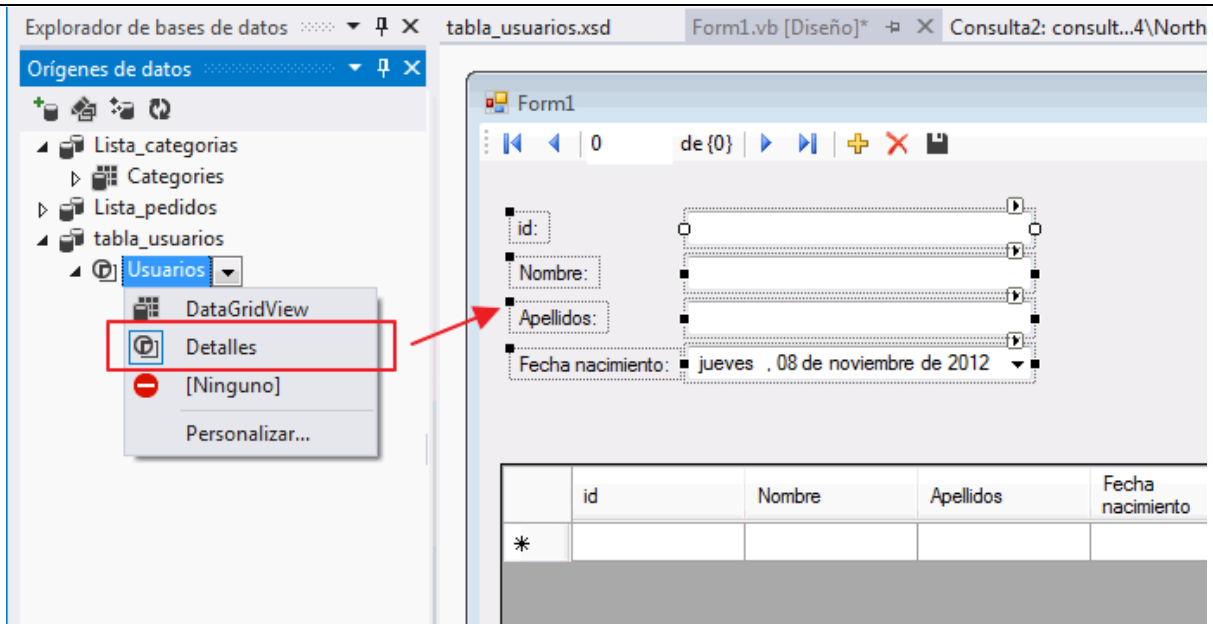
Y con las dos instrucciones, primero para cargar del servidor de base de datos el Dataset a nuestro equipo local:

```
private void Form1_Load(object sender, EventArgs e)
{
    // TODO: esta línea de código carga datos en la tabla
    // 'usuariosDataSet.Usuarios' Puede moverla o quitarla según sea necesario.
    this.usuariosTableAdapter.Fill(this.usuariosDataSet.Usuarios);

}
```

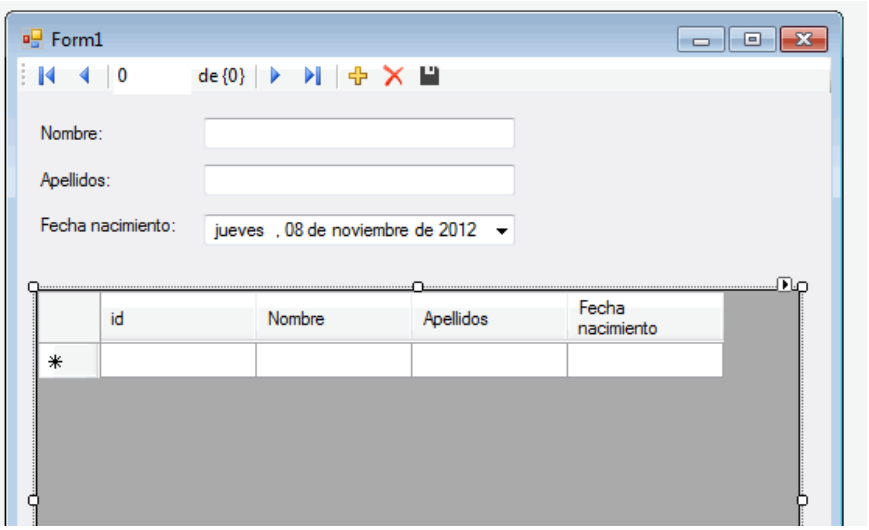
Y dentro del botón de grabar la actualización del Dataset local con el servidor:

```
private void usuariosBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    this.Validate();
    this.usuariosBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.usuariosDataSet);

}
```

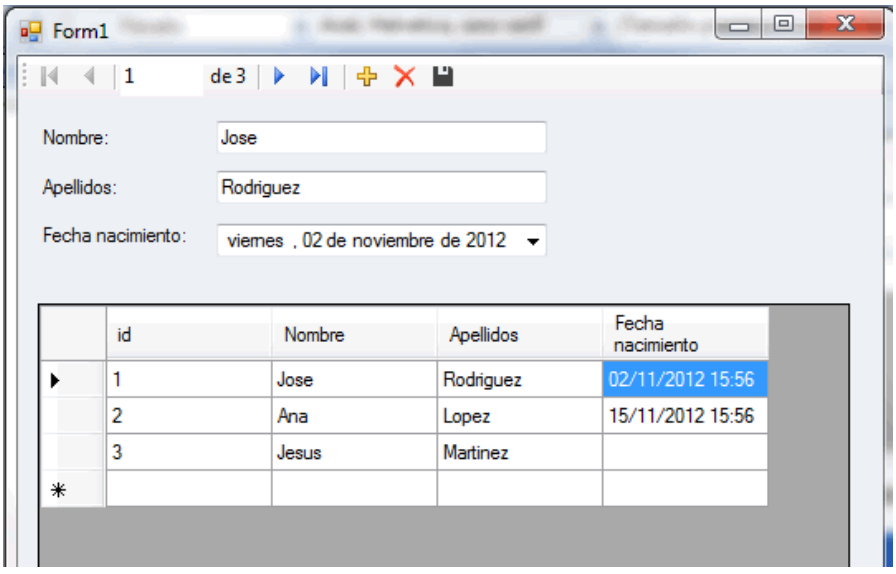
Ahora vamos a seleccionar que el control para explorar los datos va a ser un "Detalles" que es una vista de los campos individual. Volvemos a dejar en el campo de fecha el control "DatePicker" para que quede más simplificada la pantalla. Volvemos a hacer clic en "usuarios" y lo arrastramos a nuestro formulario:



Vamos a eliminar el campo "Id" que es de lectura y no queremos que entre en nuestra edición:



Ejecutamos y tenemos la vista que hicimos antes de forma manual:



Podemos seguir explorando las opciones que nos da el control de cuadrícula para mejorar el aspecto. También podemos variar las opciones que le ponemos al usuario, por ejemplo no añadir más registros.

En definitiva, podemos comprobar que la versatilidad es bastante grande. Ahora pasemos a aspecto un poco más avanzado de las bases de datos.

A partir de aquí vamos a ponernos a trabajar a fondo con ADO.NET. Pero esta vez no abusando de todo lo que el IDE hace automáticamente por nosotros sino haciendo las cosas a mano ya que muchas veces lo que queremos mostrar u ofrecer al usuario no se podrá hacer mediante los asistentes.

8. Los objetos de ADO.NET

Comencemos por ver de forma ordenada los objetos que componen el mundo de ADO.NET, no son muchos y nos servirá de referencia para siempre. Tenemos dos mundos en ADO.NET, el mundo de los objetos conectados y el de los desconectados. La única excepción la tiene el objeto "DataAdapter" que es la unión entre los dos mundos. Veamos los objetos:

8.1. Objetos conectados

Estos objetos son los que tiene una conexión abierta entre el usuario y la base de datos. Los objetos son:

- **Connection**. Es el objeto imprescindible que nos permite establecer la conexión con la base de datos. Dependiendo del proveedor de datos utilizado podemos conectarnos a distintas bases de datos. Ya lo vimos y son: OleDbConnection, SqlConnection y OracleConnection, por ejemplo.
- **Transaction**. Las transacciones se utilizan mucho en entornos avanzados y consisten simplemente en la ejecución de varios comandos en un lote. Nos permite agrupar operaciones bajo una sola descripción y mejora notablemente el rendimiento de las bases de datos. Tiene por supuesto su propia colección de objetos: OledbTransaction, SqlTransaction, ...
- **DataAdapter**. Este objeto es la pasarela entre los aspectos conectados y desconectados de la base de datos. Establece la conexión por nosotros y realiza las operaciones programadas en él. SqlDataAdapter es el objeto para las bases de datos de SQL Server.
- **Command**. Este objeto representa un comando ejecutable y puede o no devolver datos. Echábamos de menos la posibilidad de ejecutar una sentencia SQL para realizar alguna operación, pues bien, este es el objeto para estas operaciones. Las de consulta devuelven

datos y las que modifican datos no devuelven nada. SqlCommand es el comando para SQL Server, OledbCommand, ...

- **Parameter**. Es un comando para tratar con parámetros. Es muy útil para poder poner parámetros y construir sentencias SQL correctas.
- **DataReader**. Es un objeto de solo lectura y que solo se puede recorrer hacia adelante para recuperar datos. Es decir, podemos hacer una consulta rápida con este objeto pero no podremos tener opciones avanzadas como la ordenación o paginación de resultados. Vienen muy bien por ejemplo, para rellenar un cuadro de lista con unos valores. Aun con sus defectos tiene como ventaja que es muy rápido.

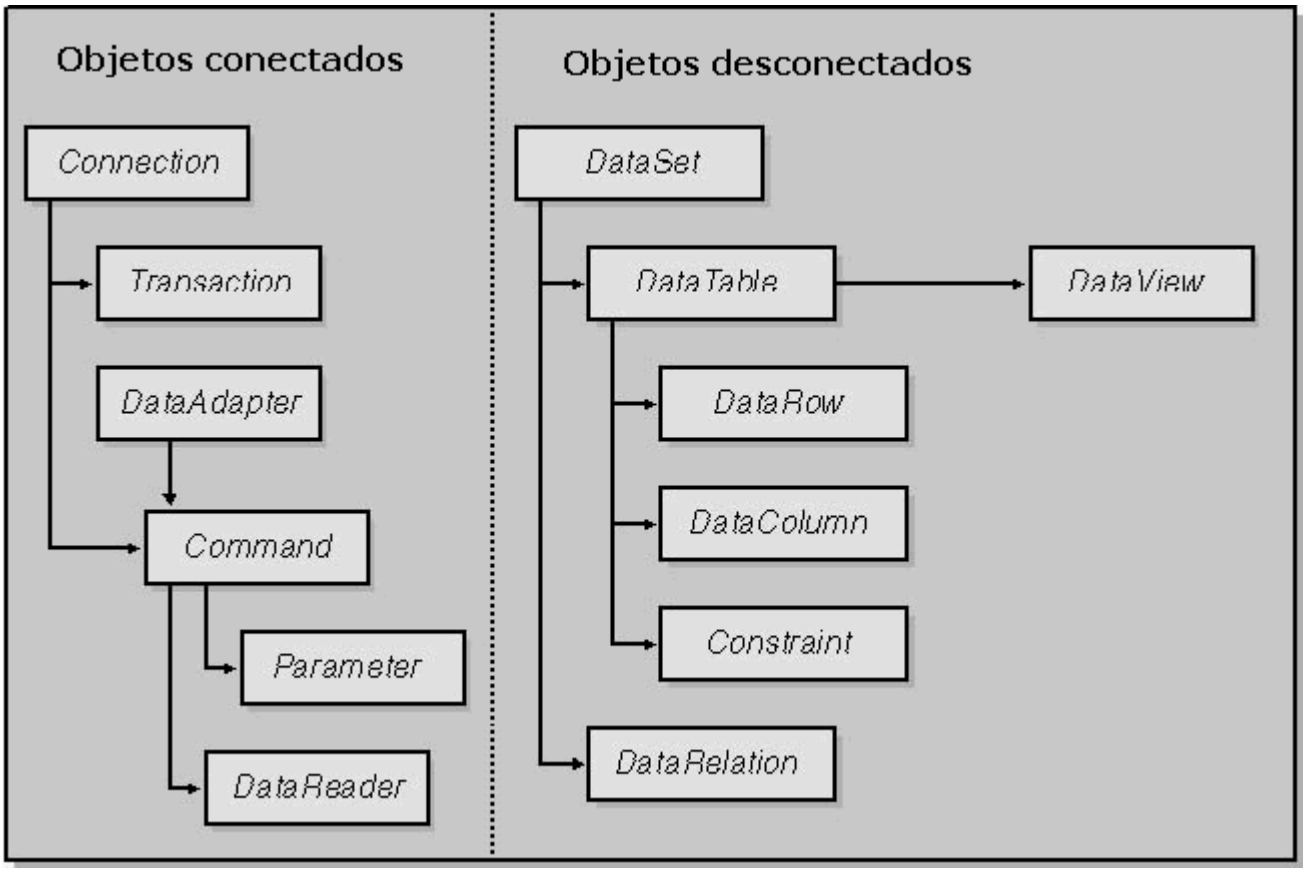
8.2. Objetos desconectados

Es la forma con la que hemos trabajado hasta ahora. Después de muchos estudios se llegó a la conclusión de que mantener conexiones y consultas permanentemente abiertas afectaba mucho al rendimiento de la aplicación así que se diseñaron este tipo de objetos. Se conectan lo justo para actualizar los datos y se vuelven a desconectar. Los objetos son:

- **Dataset.** Ya lo conocemos y sabemos de sobra que es desconectado ya que si no llamábamos a un método para que a través del adaptador ejecutara un comando de actualización, no se modificaban los datos. Es el núcleo central de este tipo de objetos y representa un miniservidor de base de datos en sí mismo, ya que si recuerdas podía tener dentro varias tablas, además relaciones entre ellas, ... Este objeto se compone a su vez de una colección de DataTables y DataRelations
- **DataTable.** Es lo más parecido a una tabla de una base de datos ya que se compone de una colección de filas y columnas.
- **DataRow.** Una de las propiedades de DataTable es Rows del tipo DataRowCollection, que representa a una colección de filas o DataRow. Es lo equivalente a una fila de una tabla (DataRow) o una colección de filas (DataRowCollection)
- **DataColumn.** Un datatable contiene además una propiedad Columns de tipo DataColumnCollectios. Representa la estructura de la tabla ya que las columnas es como los campos que componen esa tabla.
- **DataView.** Es una vista de una base de datos. Permite crear una vista de los datos que puede ser subconjunto de ellos aplicando, por ejemplo, un filtro (filter) y además con criterios de ordenación Sort. Un DataTable puede contener múltiples vistas definidas.
- **Constraint.** Es el objeto que se dedica a, mediante las claves de las tablas, las relaciones entre ellas.

- **DataRelation.** Un Dataset como ya hemos dicho que se puede comportar como una mini base de datos, puede contener varias tablas. Mediante un objeto de este tipo podemos especificar las relaciones entre las tablas. La diferencia con el objeto anterior es que en este además se validan los datos proporcionándonos mecanismos para explorar los datos maestros y los relacionados.

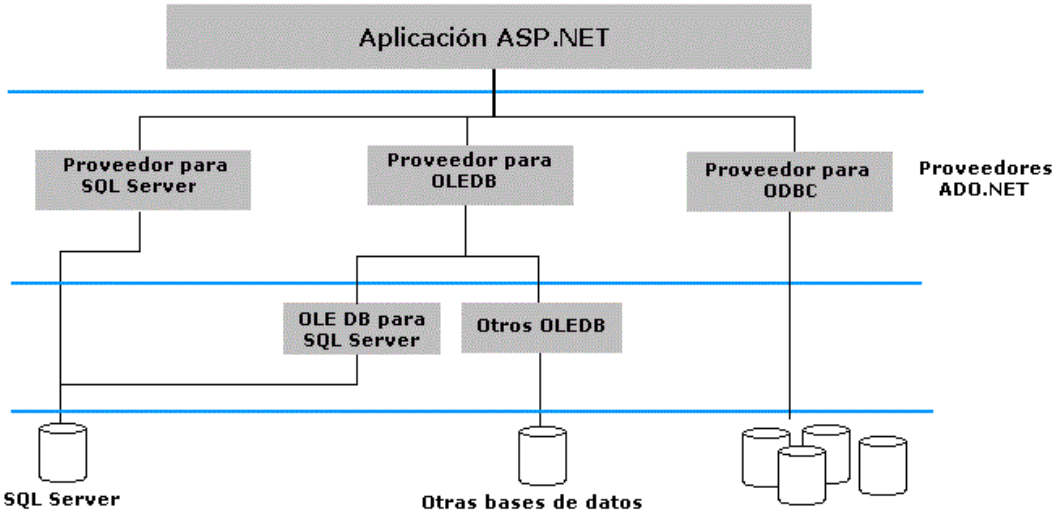
En este cuadro podemos ver los dos grupos de objetos:



Este esquema es muy útil para situarnos en la estructura y saber qué objetos están en un mundo y en otro.

8.3. Proveedores de .NET

Antes de conectar a una base de datos debemos por supuesto tener claro el formato o servidor de base de datos que vamos a utilizar en nuestra aplicación. Puesto que cada base de datos tiene sus propias particularidades se han escrito distintos proveedores de datos. Uno solo sería impensable porque no es lo mismo manejar un .mdb de Access ejecutándose en mi equipo que acceder a un superservidor con Oracle y Unix como sistema operativo. Por tanto, debemos utilizar el proveedor adecuado. Recordemos este gráfico:



Por un lado, tenemos los servidores de bases de datos de Microsoft, SQL Server. Por otro lado, tenemos un conector "universal" para todas las bases de datos que soporten "OleDB" que es una evolución de los "ODBC" que también tienen su sitio. Para Access utilizaremos la conexión OleDB y, por ejemplo, para la conocida base de datos de Linux podemos utilizar también el controlador OleDB que nos proporciona el fabricante.

Los espacios de nombres que ves aquí son los que nos van a permitir conectarnos con las bases de datos, Cada grupo de objetos de los proveedores está con su espacio de nombres, luego deberemos importar a nuestro proyecto el que vayamos a utilizar:

Namespace	Función
System.Data	Contiene las clases fundamentales del núcleo de ADO.NET. Incluye los objetos "DataSet" y "DataRelation" para modificar la estructura relacional de los datos. Estas clases son independientes del tipo de controlador que se utilice.
System.Data.Common	No se utilizan en nuestro código. Las utilizan otras clases proveedoras de datos.
System.Data.Oledb	Contiene las clases necesarias para conectarnos a orígenes de datos OLE DB, y la ejecución de comandos: "OleDbConnection" y "OleDbCommand"
System.Data.SqlClient	Contiene las clases para conectarnos y ejecutar comandos en un servidor de base de datos SQL Server. Las propiedades y métodos de "SqlConnection" y "SqlCommand" son las mismas que cuando la bbdd es de tipo OLEDB con los objetos anteriores.
System.Data.SqlTypes	Contiene estructuras para tipos de datos específicos de SQL Server, como SqlMoney y SqlDateTime. De esta forma no hay que convertir tipos de datos.
System.Data.OracleClient	Contiene las clases para conectarnos y ejecutar comandos en un servidor de base de datos Oracle: "OracleConnection" y "OracleCommand
System.Data.ODBC	Contiene las clases para conectarnos y ejecutar comandos a través de un controlador ODBC: "OdbcConnection" y "OdbcCommand

La utilización del proveedor adecuado es mucho más importante de lo que parece porque aunque SQL Server y Oracle pueden conectarse con OleDB ya que traen controladores de este tipo, es

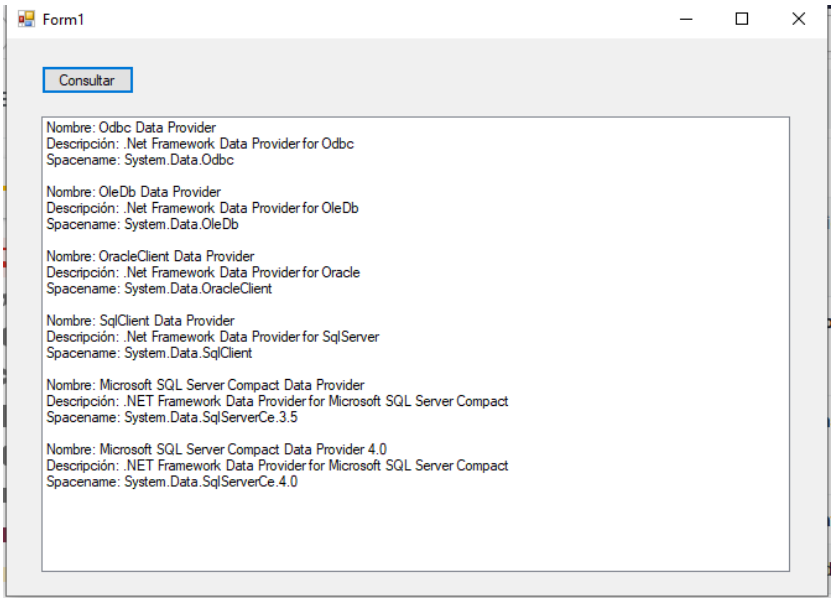
mejor utilizar los específicos. Además de tener mucho mejor rendimiento utilizan los mismos tipos de datos. Imagina que un entero "int" con OleDB debe traducirse a su tipo de datos equivalente según la base de datos que sea, esto hace que sea mucho más lento.

Vamos a hacer un sencillo ejemplo para ver los proveedores de datos que tenemos instalados. El código crea un objeto de tipo tabla "DataTable" e introduce con un bucle la colección de elementos de "DbProviderFactories.GetFactoryClasses". Esa colección nos mostrará los proveedores que tenemos instalados en nuestro equipo.

Creamos una aplicación nueva de Windows, ponemos un cuadro de lista y un botón. Importamos el espacio de nombres de "System.Data.Common " y ponemos el siguiente código en el evento clic del botón:

```
using System.Data.Common;

private void btnConsultar_Click(object sender, EventArgs e)
{
    DataTable listaClases = new DataTable();
    listaClases = DbProviderFactories.GetFactoryClasses();
    foreach (DataRow clase in listaClases.Rows)
    {
        listBox1.Items.Add("Nombre: " + clase["Name"]);
        listBox1.Items.Add("Descripción: " + clase["Description"]);
        listBox1.Items.Add("Spacename: " + clase["InvariantName"]);
        listBox1.Items.Add(" ");
    }
}
```

Lo ejecutamos y obtendremos:

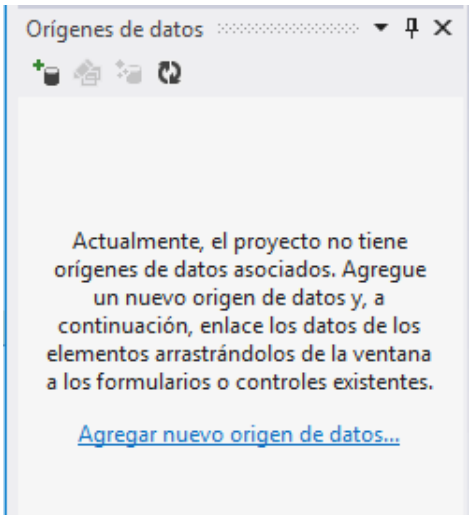


Una interesante lista de los proveedores. Si nos fijamos en los tres últimos, el primero es para un servidor SQL estándar y los otros dos son para el mini servidor SQL que nos ha instalado con Visual Basic. Vamos ya a conectarnos a bases de datos y a realizar sencillas operaciones.

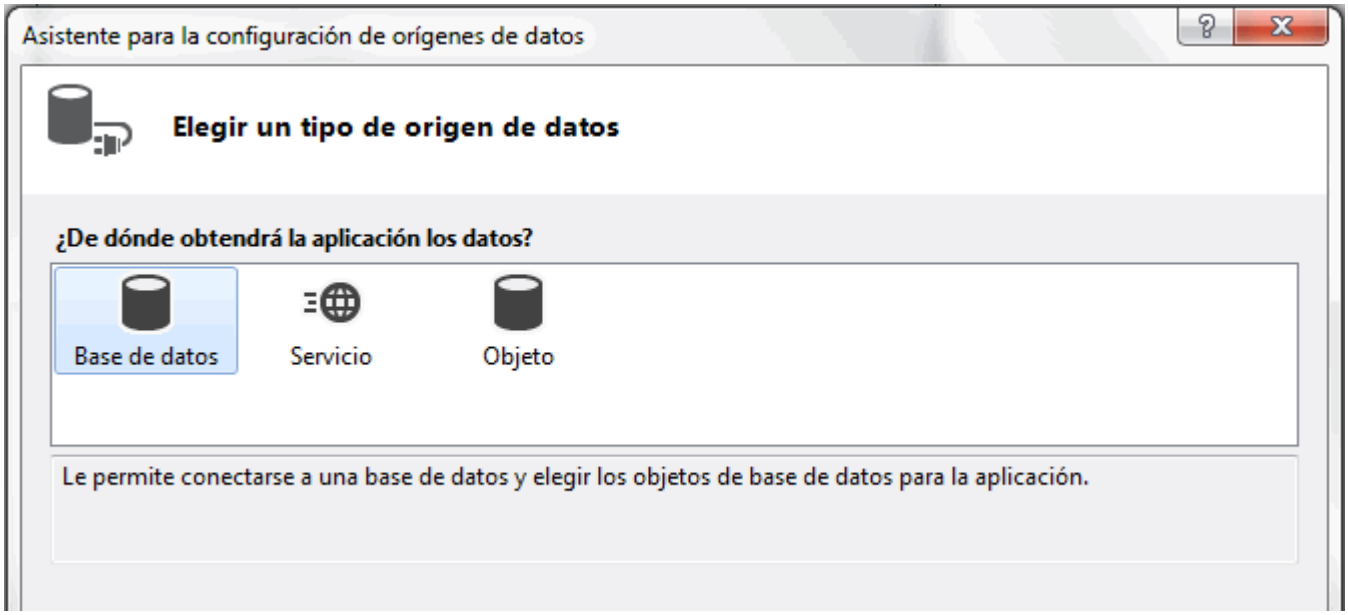
9. Consultas a la bbdd, del código automático al manual

Ya sabemos con qué base de datos nos vamos a conectar y sabemos el proveedor que utilizaremos. Ahora haremos ejemplos con éstos. Para realizar la conexión tendremos que proporcionar una serie de datos que dependen del proveedor indicado. En SQL Server debemos indicar el servidor con su usuario y contraseña de acceso, en SQL Server Compact un archivo de base de datos, con Access el fichero .mdb, ...

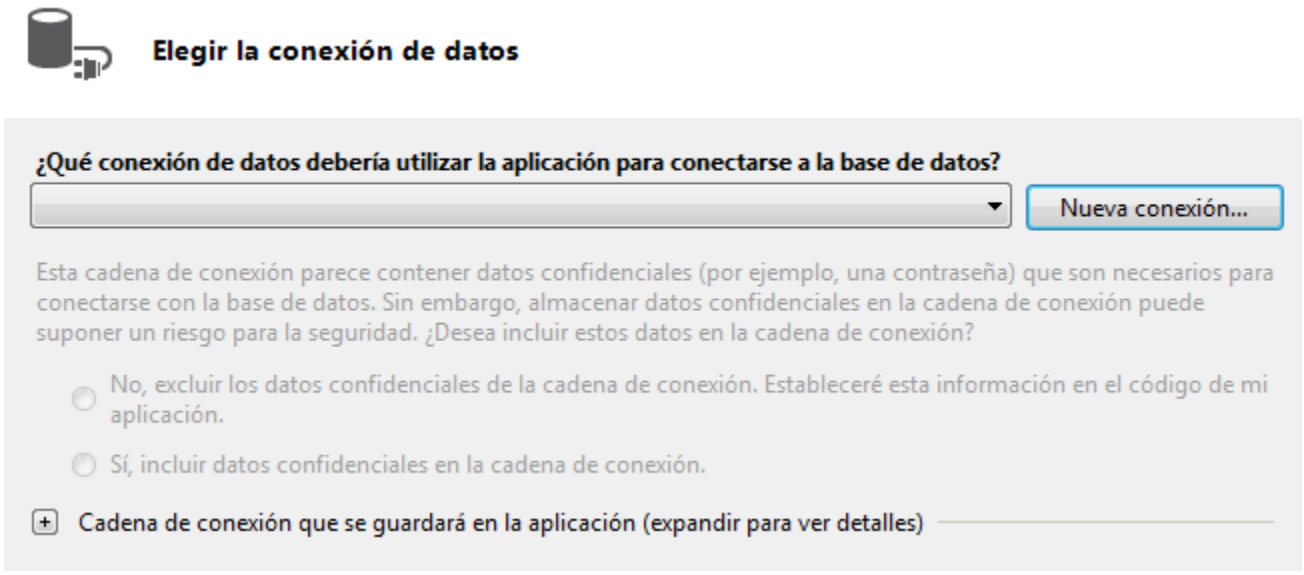
Vamos a crear una nueva aplicación de Windows y mostraremos la ventana de los orígenes de datos:



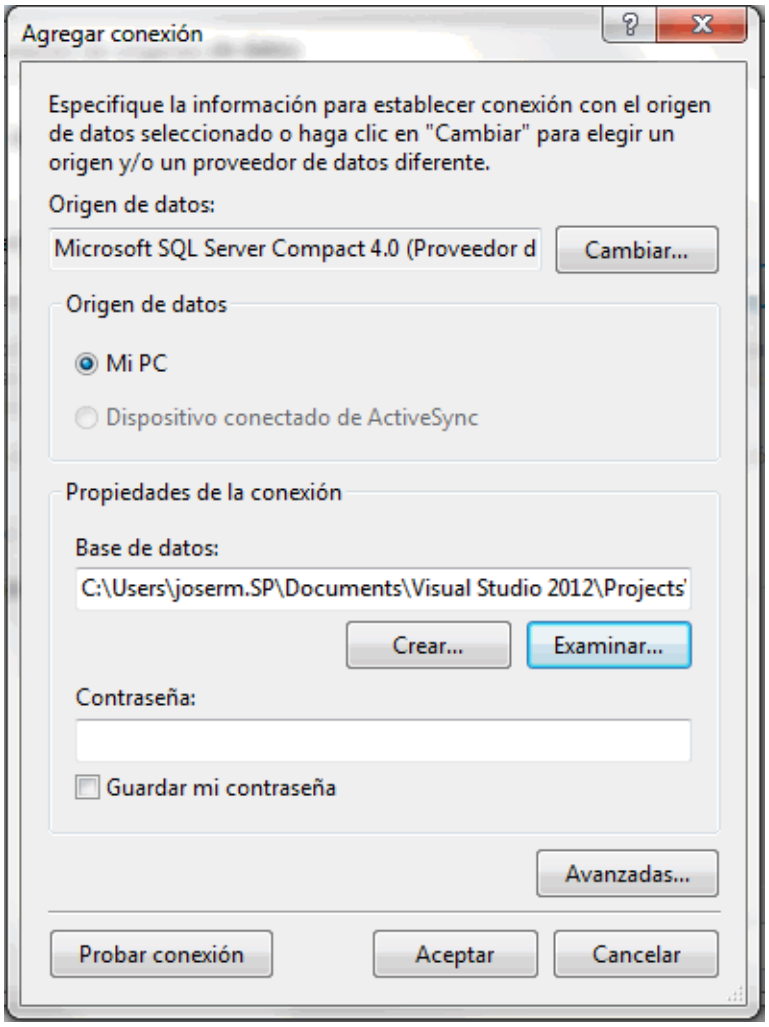
Añadimos un nuevo origen de datos:



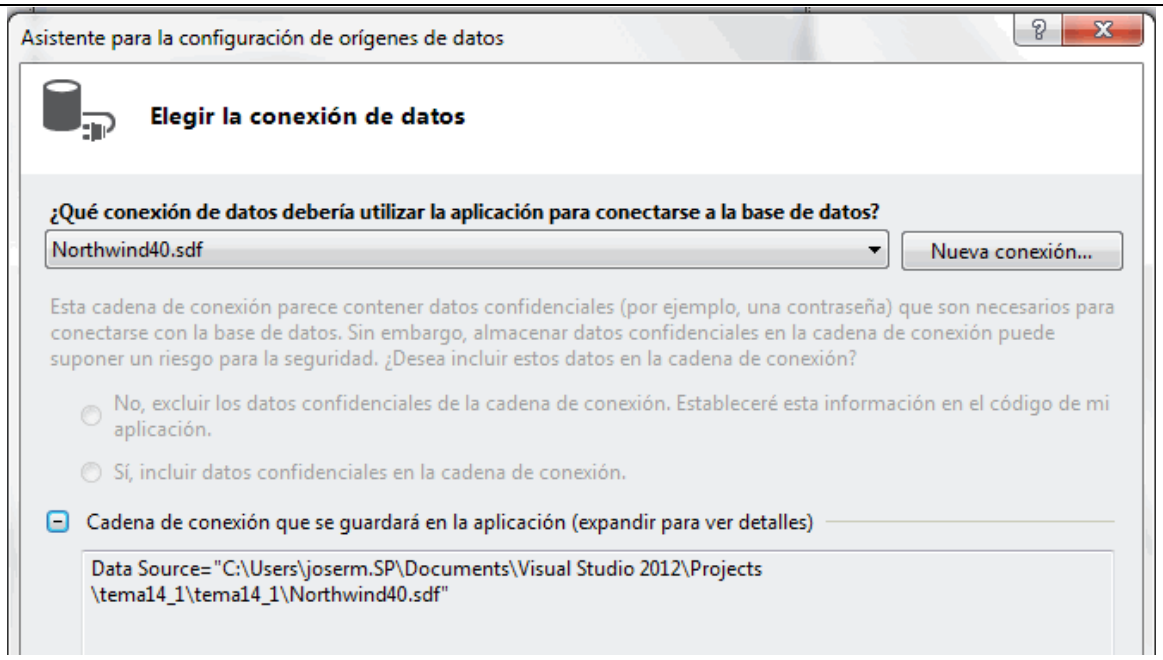
No tenemos ninguna conexión. Como es un proyecto nuevo no hay ninguna definida, así que pulsaremos en nueva conexión



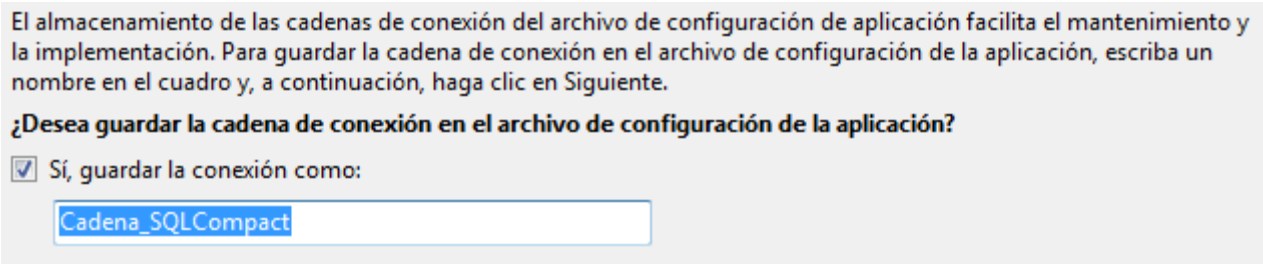
Volveremos a seleccionar el proveedor de "Sql Server" y seleccionaremos la base de datos de antes:



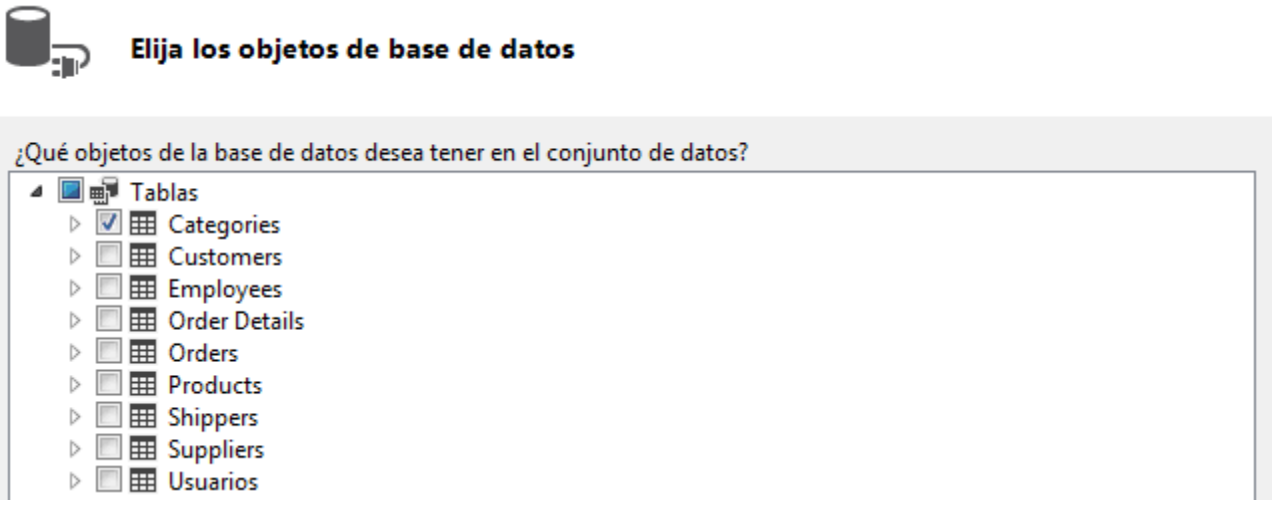
Finalmente pulsaremos en probar conexión para ver si está todo correcto y pulsamos en terminar:



Atento a la cadena de conexión que tenemos debajo. Esta cadena es distinta según el proveedor de datos. Para SQL Server ésta es la necesaria. Al continuar nos preguntará por almacenar la base de datos en nuestro proyecto y si queremos almacenar la cadena de conexión en un fichero de configuración del proyecto:



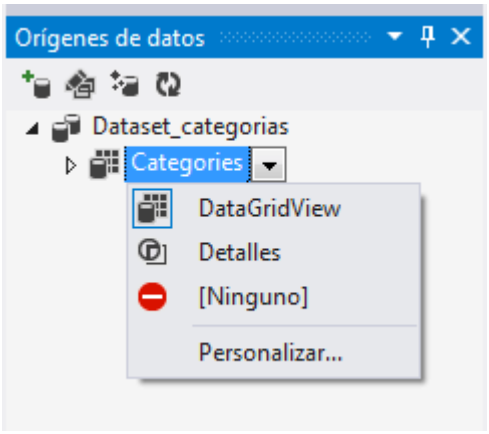
Le ponemos un nombre y le diremos que sí. Finalmente le diremos que queremos incluir la tabla de las categorías:



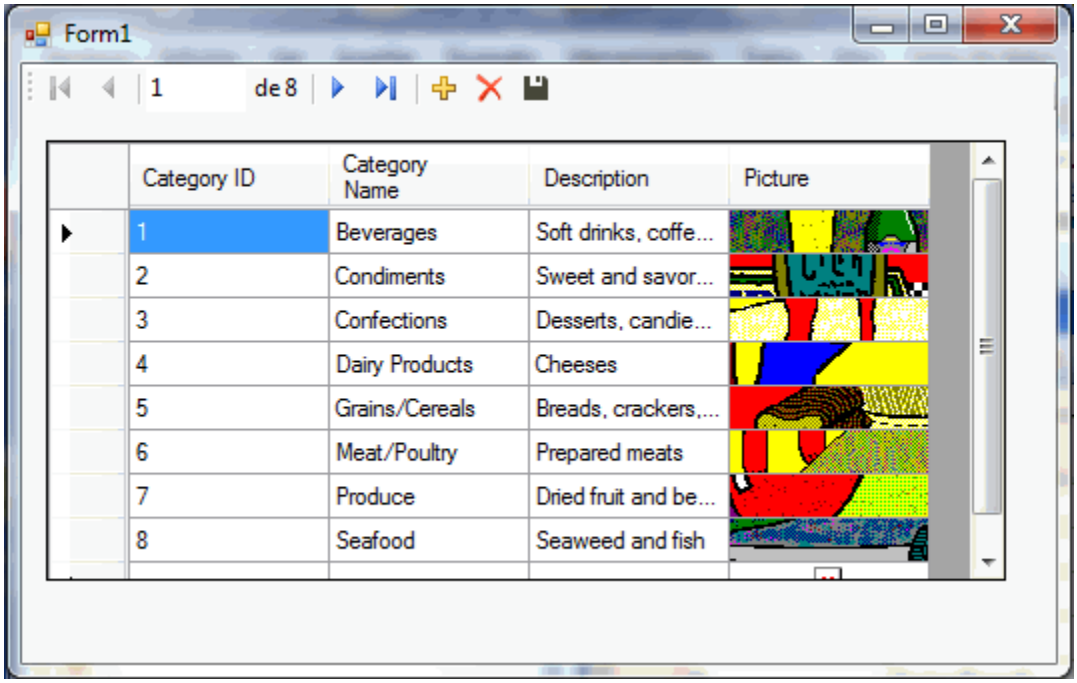
9.1. Consultar la base de datos con el IDE

Tenemos el fichero de base de datos, un fichero de configuración donde ha almacenado la cadena de conexión y un "dataset" con los datos de la tabla de categorías que pusimos antes.

Volvemos a los orígenes de datos y hacemos clic junto al dataset:



Anteriormente seleccionábamos los campos individualmente, ahora hemos seleccionado el dataset completo y, por tanto, en lugar de pedirnos un control sencillo nos ofrece controles capaces de mostrar datos con muchos campos, como es una tabla. Comprobamos que tenemos la cuadrícula "DataGridView" seleccionada. Arrastramos al formulario el nombre de la tabla "categories" y ejecutamos el programa:



Incluso como uno de los campos es de tipo binario que incluye una imagen, el control es capaz de mostrárnoslas. Vamos ahora a la cadena de conexión que se ha escrito en el fichero de configuración que se ha incluido en el proyecto y que se llama "app.config":

```
<connectionStrings>
    <add name="Ejemplo6___T12.Properties.Settings.NorthwindConnectionString"
        connectionString="Data Source=SERVER\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Y ahora veamos el código autogenerado en el programa:

```
private void Form1_Load(object sender, EventArgs e)
{
    // TODO: esta línea de código carga datos en la tabla 'categoriasDataSet.Categories'
    // Puede moverla o quitarla según sea necesario.
    this.categoriesTableAdapter.Fill(this.categoriasDataSet.Categories);
}

private void CategoriesBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    this.Validate();
    this.categoriesBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.categoriasDataSet);

}
```

Que es, todo lo que necesitamos... ¿todo? Pues no, hace falta más código que no está visible y que está en un fichero del proyecto adjunto y que veremos después.

Parece una maravilla esto de arrastrar la tabla para poder editarse... ¿seguro? Pues va a ser que casi que no. Me explico, nuestras aplicaciones van a ser cada vez más completas y complejas... y no sería nada raro que las bases de datos tuvieran varias o decenas de tablas. Incluso las aplicaciones complejas tienen cientos de tablas. ¿Sería un buen diseño el estar visualmente arrastrando las tablas? Pues evidentemente no. Esta técnica nos puede ayudar en cosas sencillas pero necesitaremos del código para poder manejar las tablas, podremos hacer los mismos vínculos por código y será mucho más fácil de mantener.

9.2. Realizar una consulta con el código

En este ejemplo vamos a hacer todos los pasos manuales. Es la versión más didáctica porque veremos todos los pasos más claros. Además en muchísimas ocasiones necesitaremos ejecutar sentencias SQL aisladas sin tener que pensar cómo crear "DataSet" o vistas gráficas para poder ejecutarlas.

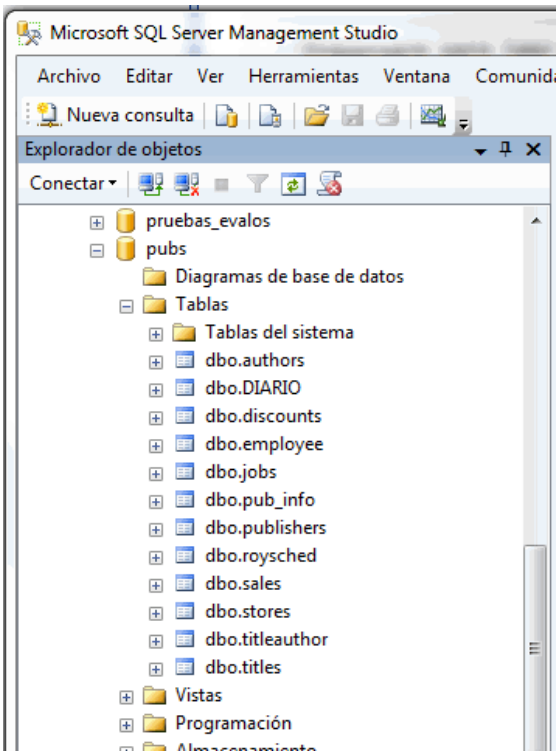
Creamos una nueva aplicación e importamos nuestro espacio de nombres para SQL:

```
using System.Data.SqlClient;
```

Esta vez vamos a conectarnos con un servidor SQL Server. Se trata de la versión SQL Server Express que puedes descargarte desde la página de Microsoft. Cuando lo hagas asegúrate de descargarte la versión completa, con la consola de administración para así poder realizar tareas de mantenimiento en las tablas de una forma más completa que la que nos ofrece el IDE.

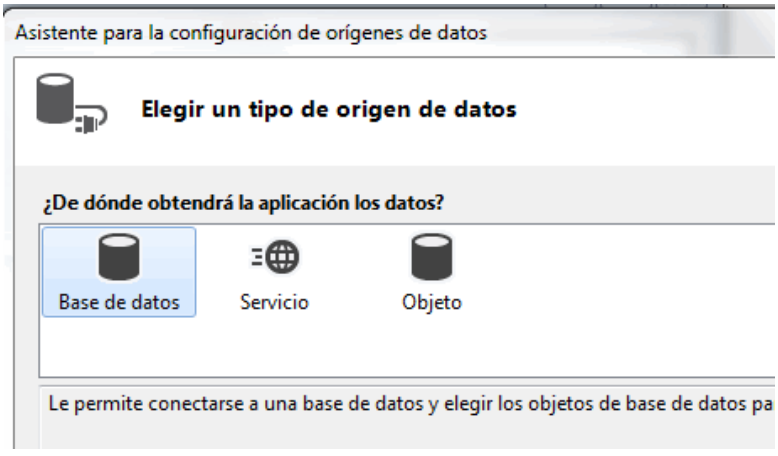
A continuación se realiza el ejemplo con la bbdd pubs. Nosotros trabajaremos en clase con la bbdd Northwind, que importamos al principio del tema.

En la instalación permite instalar una base de datos de prueba llamada "PUBS" y que tiene unas tablas distintas que las de SQL Compact. La idea es la misma, pocas tablas y relacionadas:

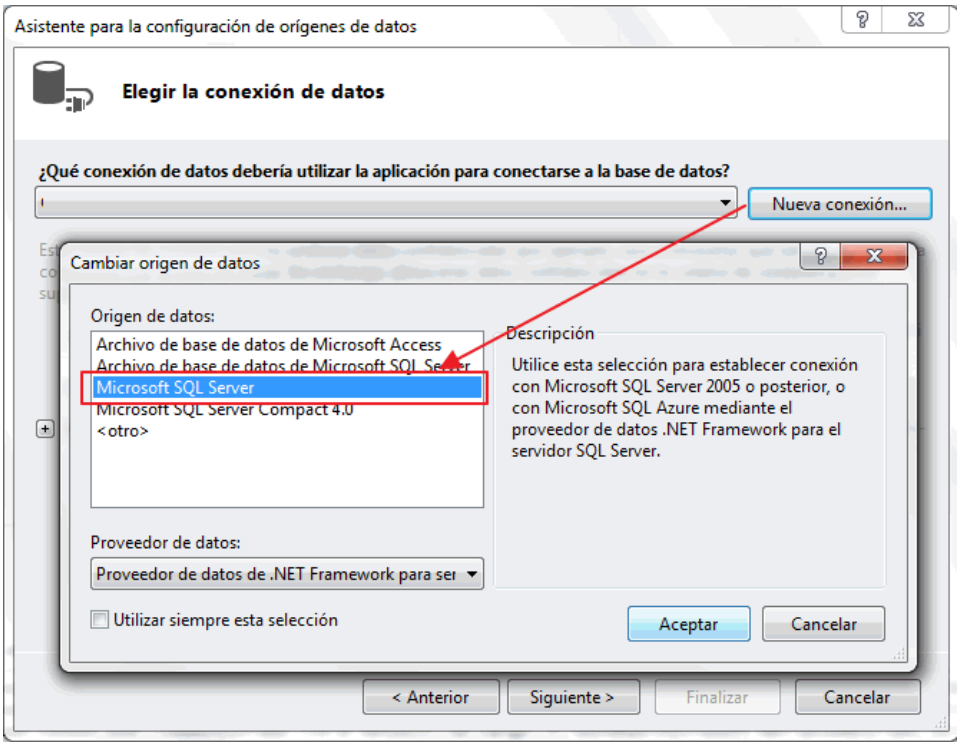


En la tabla "Authors" hay una lista de autores de libros, donde queremos extraer los campos de nombre y apellidos, pero todo de forma manual. Vamos con ello.

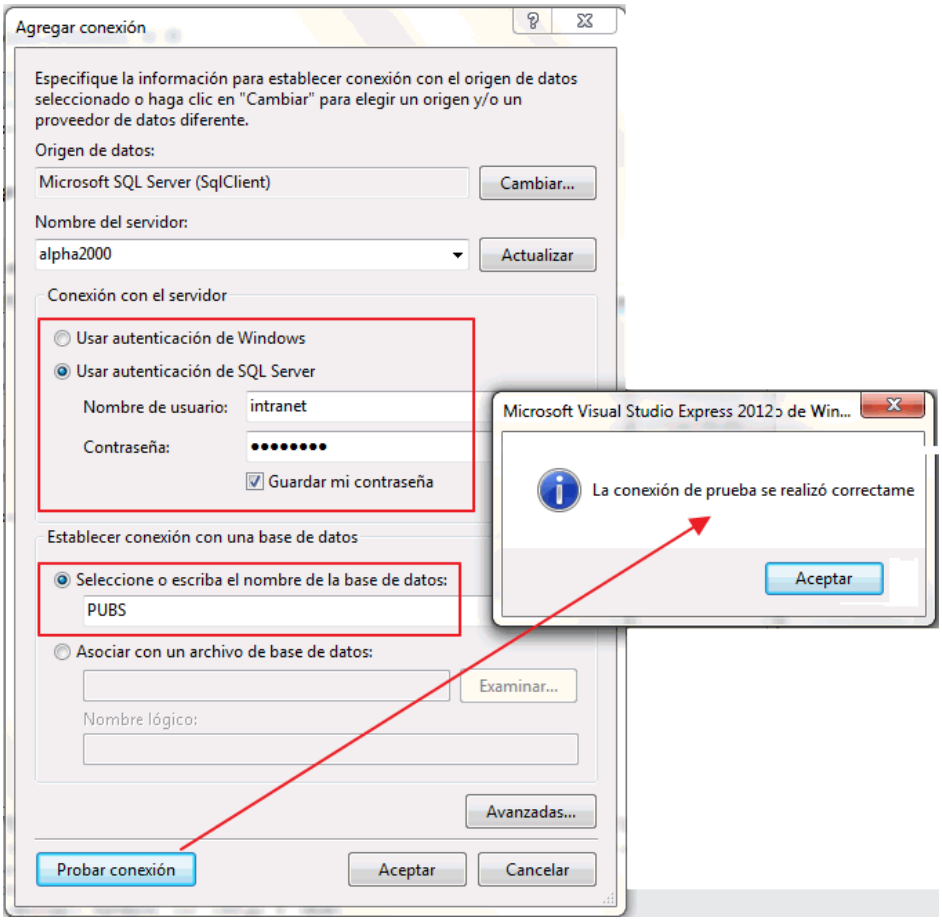
Creamos una aplicación Windows y esta vez no incluiremos ninguna conexión de base de datos, lo vamos a definir todo desde el código. Para empezar vamos a definir la cadena de conexión. No vamos a utilizar ningún objeto ni nada automático, haremos lo mismo que antes pero de forma manual. Para empezar tenemos que definir la cadena de conexión contra este nuevo servidor de base de datos. Para generar esta cadena vamos a utilizar un pequeño truco. Al crear la aplicación nos vamos a añadir un nuevo origen de datos para que nos muestre el asistente:



Pulsamos para seleccionar una nueva conexión:



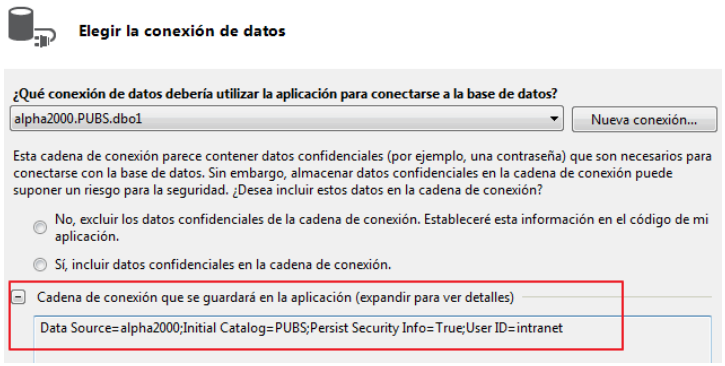
En este caso se trata de un servidor SQL Server, así que lo marcamos y continuamos para completar los datos de conexión:



Para una conexión a este tipo de servidores necesitamos estos datos:

- Nombre de servidor. En nuestro ejemplo se llama "alpha2000"
- Nombre de base de datos: PUBS
- Usuario: "intranet"
- Contraseña: "intranet"

Una vez está la cadena probada, podemos ver la cadena de conexión aquí:



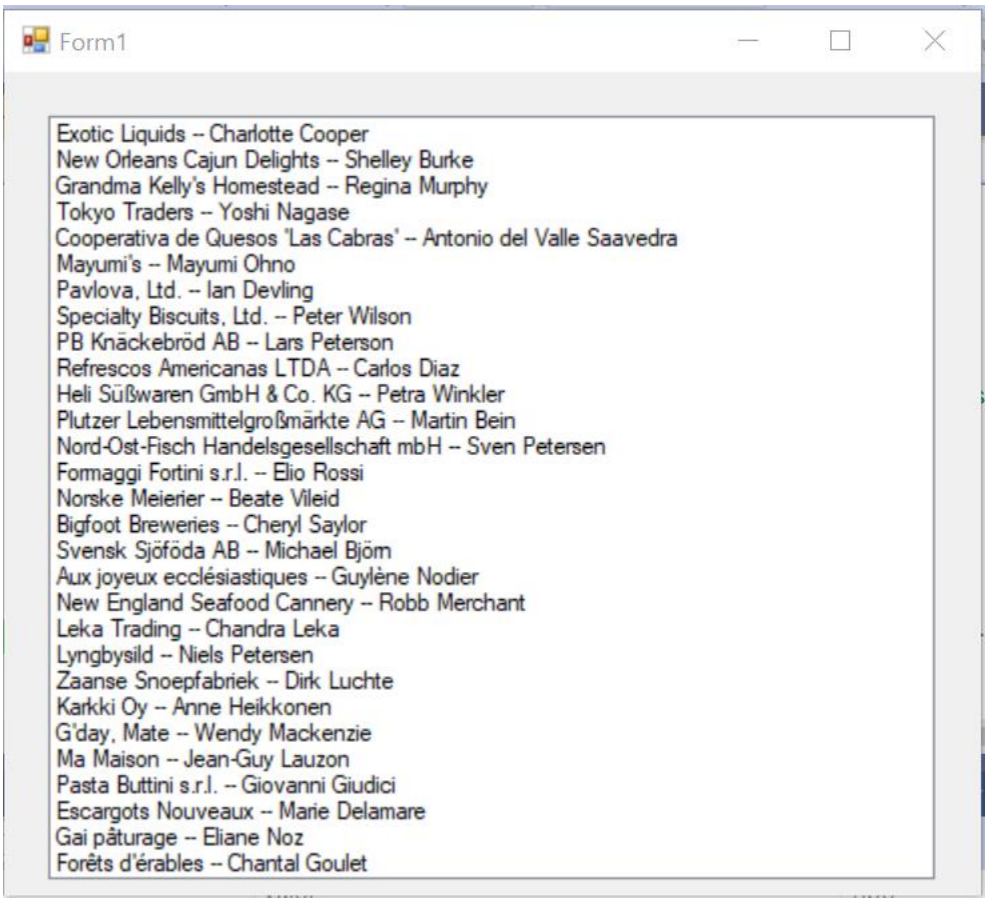
Esta es la cadena que nos interesa. Así que la copiamos y cancelamos el asistente. Esta cadena la vamos a pegar en un campo de nuestro programa:

```
// Cadena de conexión
string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog = Northwind; Integrated Security = True";
```

Y el código que tenemos tener al final es:

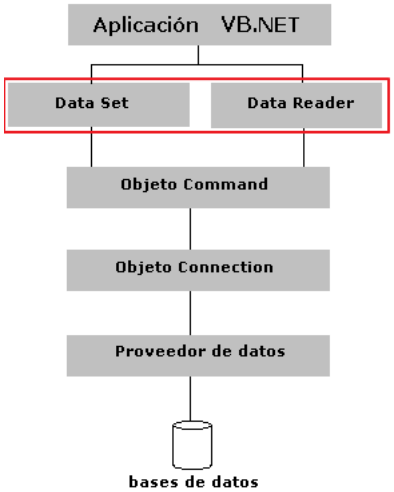
```
using System.Data.SqlClient;

public Form1()
{
    InitializeComponent();
}
// Cadena de conexión
string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog = Northwind; Integrated Security = True";

// Creamos el objeto conexión
SqlConnection conexion = new SqlConnection();

//Creamos el objeto comando
SqlCommand comando = new SqlCommand();

1 reference
private void Form1_Load(object sender, EventArgs e)
{
    conexion = new SqlConnection(cadenaConexion);

    // Asigmos al comando la consulta a realizar
    comando.CommandText = "Select * from Suppliers";

    comando.Connection = conexion;

    // Abrimos la conexión
    conexion.Open();

    // Creamos un objeto DataReader para almacenar el resultado de la consulta
    // Ejecutamos la consulta con ExecuteReader
    SqlDataReader resultado = comando.ExecuteReader();

    // Hacemos un bucle por el DataReader para escribir los datos en el listBox
    while (resultado.Read())
        listBox1.Items.Add(resultado["CompanyName"] + " -- " + resultado["ContactName"]);

    conexion.Close();
}
```

Hemos hecho todos los pasos: crear una conexión, un comando SQL, ejecutarlo, almacenar el resultado y luego mostrarlo. Hemos mostrado en un cuadro de lista el nombre de la compañía y el nombre del contacto de la tabla de Suppliers:



Bueno, hemos conseguido lo mismo que utilizando los controles del principio del tema. Vamos a ver los pasos que hemos realizado uno a uno.

Lo primero que vemos es que no hay ningún adaptador. No lo hay porque no hemos utilizado un "DataSet" sino un "DataReader" que para rellenar datos temporales es perfecto y más sencillo. Recordemos el grafico:



Tenemos que verlo desde abajo hasta arriba, desde la conexión hasta el enlace con el cuadro de lista. Para empezar, en lugar de utilizar un conjunto de datos DataSet como resultado, hemos utilizado el DataReader. Éste es un objeto muy sencillo y rápido que no permite ciertas acciones como la navegación adelante-atrás o la actualización de los datos. Pero es muy útil para acciones como esta que es la de rellenar ciertos controles para permitir selecciones al usuario. Los pasos han sido los siguientes, para empezar y ya como siempre, la creación de un objeto de conexión de acuerdo con el proveedor de datos utilizado. Este objeto es un "SqlConnection":

```
// Cadena de conexión
string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog = Northwind; Integrated Security = True";


// Creamos el objeto conexión
SqlConnection conexion = new SqlConnection();


private void Form1_Load(object sender, EventArgs e)
{
    conexion = new SqlConnection(cadenaConexion);
```

La cadena de conexión como presumiblemente la vamos a utilizar más veces en nuestro código la hemos puesto fuera del método para que esté al alcance de toda la clase. El siguiente paso es la creación de un comando para ejecutar con esa conexión:

```
//Creamos el objeto comando
SqlCommand comando = new SqlCommand();


// Asigmos al comando la consulta a realizar
comando.CommandText = "Select * from Suppliers";
```

Simplemente es un objeto de conexión con la sentencia SQL asociada. Ahora que tenemos la conexión y el comando vamos a abrir la conexión y a ejecutar la consulta:

```
// Abrimos la conexión
conexion.Open();

// Creamos un objeto DataReader para almacenar el resultado de la consulta
SqlDataReader resultado = comando.ExecuteReader();


// Abrimos la conexión
conexion.Open();

// Creamos un objeto DataReader para almacenar el resultado de la consulta
// Ejecutamos la consulta con ExecuteReader
SqlDataReader resultado = comando.ExecuteReader();
```

Con esta instrucción ejecutamos el método necesario para obtener un "DataReader".

Ya tenemos los datos para poder explotarlos así que haremos un bucle por todo ese "DataReader" e insertar los campos que queremos en el cuadro de lista:

```
// Hacemos un bucle por el DataReader para escribir los datos en el listBox
while (resultado.Read())
    listBox1.Items.Add(resultado["CompanyName"] + " -- " + resultado["ContactName"]);
```

Como vemos la ejecución controlada es muy interesante. Podemos hacer exactamente lo que queremos, en lugar del código automático que nos colocan los asistentes y que, si bien nos sirven para cosas sencillas, no nos permite la flexibilidad del código.

10. El objeto conexión

Como estamos en un enfoque avanzado de ADO.NET y pretendemos hacer por código todo lo necesario para explotar las bases de datos, vamos a ver de forma más avanzada la primera parte, que es la conexión a la base de datos.

La clase para la conexión con SQL Server se llama SqlConnection, recuerda los nombres de las clases según el tipo de proveedor que utilicemos en nuestro programa:

Proveedores -->	SQL Server	OleDB	Oracle	ODBC
Connection	SqlConnection	OleDBConnection	OracleConnection	OdbcConnection
Command	SqlCommand	OleDBCommand	OracleCommand	OdbcCommand
DataAdapter	SqlDataAdapter	OleDBDataAdapter	OracleDataAdapter	OdbcDataAdapter

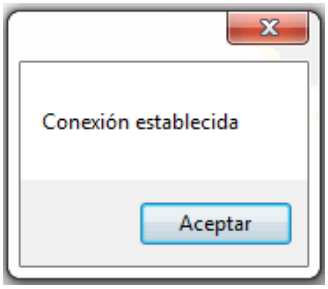
Para crear el objeto conexión lo realizábamos de esta forma:

```
SqlConnection conexion = new SqlConnection();

string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog =
Northwind; Integrated Security = True";

conexion = new SqlConnection(cadenaConexion);
```

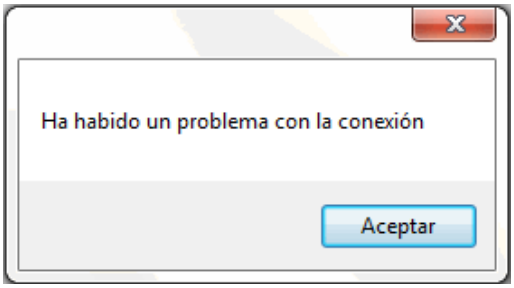
Este código crea el objeto para realizar la conexión pero todavía no la abre, esto lo haremos en un paso posterior. Vamos a hacer un ejemplo para ver detalles del estado de conexión y control de errores. Vamos con este código:

```
// Creamos la cadena de conexión
string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog = Northwind; Integrated Security = True";
// Creamos un objeto conexión
SqlConnection conexion = new SqlConnection(cadenaConexion);
try
{
    // Abrimos la conexión
    conexion.Open();
    if (conexion.State == ConnectionState.Open)
        MessageBox.Show("Conexión establecida.");
}
catch (SqlException ex)
{
    if (conexion.State != ConnectionState.Open)
        MessageBox.Show("Ha habido un problema con la conexión. \n" + ex.Message);
}
finally
{
    if (conexion.State == ConnectionState.Open)
        conexion.Close();
    conexion.Dispose();
}
```

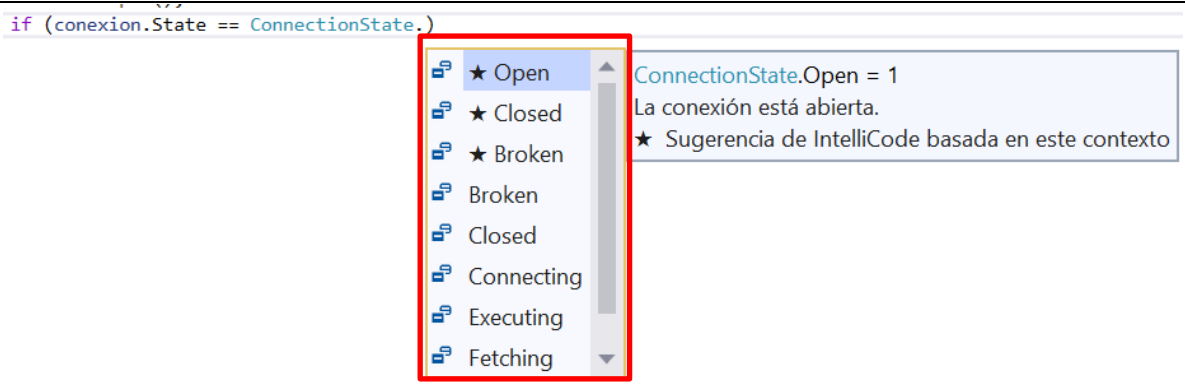
Hemos hecho una ejecución controlada de la conexión utilizando un controlador de errores "try...catch" que deberíamos utilizar siempre porque no sabemos si la base de datos destino está operativa. En el caso de que haya sido correcta la apertura obtendremos:



Si modificamos la cadena de conexión para poner un servidor inexistente y así forzar a ver si funciona esta interceptación de errores obtendremos:



Con lo cual hemos hecho un paso importante para no provocar un error en tiempo de ejecución en nuestro programa. Al editar el código hemos podido ver los distintos estados en los que se puede encontrar una conexión:



Aunque a nosotros solo nos van a interesar los de abrir y cerrar. Aun así podemos ver todos los estados en los que podemos controlar desde el tiempo en que se está conectando "connecting" hasta si se ha roto la conexión "broken". La lista de estados es pues:

Constante	Valor	Descripción
Broken	16	Se ha perdido la conexión con el origen de datos. Esto sólo puede ocurrir tras abrir la conexión. Una conexión en este estado se puede cerrar y volver a abrir. Este valor se reserva para versiones futuras del producto.
Closed	0	La conexión está cerrada.
Connecting	2	El objeto de conexión está conectando con el origen de datos. Este valor se reserva para versiones futuras del producto.
Executing	4	El objeto de conexión está ejecutando un comando. Este valor se reserva para versiones futuras del producto.
Fetching	8	El objeto de conexión está recuperando datos. Este valor se reserva para versiones futuras del producto.
Open	1	La conexión está abierta.

Un código perfecto estudiaría el estado de la conexión en cada operación de entrada/salida que realice con la base de datos. Veamos ahora los parámetros que podemos utilizar en la cadena de conexión con SqlServer.

```
string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog =
Northwind; Integrated Security = True";
```

Tenemos dos parámetros conocidos, el primero es el servidor y el segundo la base de datos. El tercer parámetro es la forma de autenticarse que tiene con el servidor SQL, en este caso utiliza la seguridad integrada de Windows. Es decir, no utiliza la gestión de usuarios de SQL Server sino que utiliza al usuario de Windows como credencial de autenticación.

SQL Server permite autenticarse de dos formas: contra un directorio activo de Windows y mixta, es decir teniendo SQL Server sus propios usuarios. Habitualmente ésta es la mejor forma, ya que no siempre vamos a tener en nuestra red un directorio activo que, si recuerdas, es la estructura de Microsoft para crear y administrar las cuentas de equipos y usuarios (y más cosas, pero básicamente eso).

En los ejemplos anteriores le dijimos que utilizara la autenticación integrada de Windows: Integrated Security = True. En este caso no hay que proporcionar usuario y contraseña porque se comprueba la del propio servicio. Veamos las diferencias:

- Con la autenticación de SQL Server. La base de datos mantiene sus propias cuentas de usuarios con los tipos de acceso que pueden tener a las bases de datos.
- Con la autenticación integrada de Windows. SQL Server utiliza la información de las cuentas de Windows y proporciona los tipos de acceso necesarios.

¿Podemos crear la cadena de conexión por código? Es decir, queremos poner esos literales del servidor base de datos y la forma de autenticación en el código. Aunque es algo poco habitual y para que veas la versatilidad de hacer todo por código, la construcción de la cadena sería de esta forma:

```
// Construimos la cadena de conexion por código
SqlConnectionStringBuilder cadenaConexion = new SqlConnectionStringBuilder
{
    DataSource = "SERVIDOR\\SQLEXPRESS",
    InitialCatalog = "Northwind",
    IntegratedSecurity = true
};

// Mostramos la cadena de conexión que hemos contruido
MessageBox.Show(cadenaConexion.ConnectionString);

// Creamos la conexión
SqlConnection conexion = new SqlConnection(cadenaConexion.ToString());

// Abrimos la conexión si es posible
try
{
    conexion.Open();
    if (conexion.State == ConnectionState.Open)
        MessageBox.Show("Conexión abierta correctamente.");
}
catch
{
    MessageBox.Show("Ha habido un problema con la conexión.");
}
```

Vemos en acción "SqlConnectionStringBuilder" construyendo la cadena de conexión. No lo utilizaremos mucho a menos que queramos en nuestra aplicación ofrecer la conexión a cualquier servidor que le pediríamos mediante un formulario.

Las propiedades nos servirán para indicarle de qué tipo es la conexión que queremos establecer y los métodos nos permitirán abrir esa conexión definida. Veamos las propiedades y métodos de este objeto:

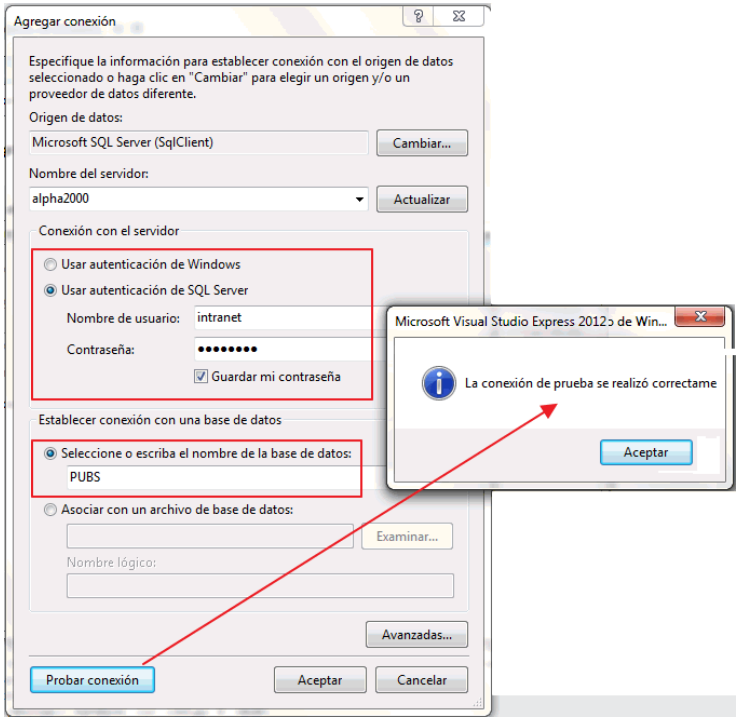
Propiedad	Descripción
ConnectionString	Cadena utilizada para conectar con la fuente de datos
ConnectionTimeOut	Número de segundos después de cual una conexión fallida de interrumpe. El valor predeterminado es de 15 segundos
Database	Devuelve el nombre de la base de datos especificada en la propiedad ConnectionString
DataSource	Devuelve el nombre del atributo DataSource especificado en la propiedad ConnectionString
ServerVersion	Devuelve la versión del servidor conectado o una cadena vacía si no se dispone de esa información.
State	Devuelve el estado actual de la base de datos.
Provider (sólo para proveedor Ole DB)	Devuelve el valor del atributo Provider especificado en la propiedad ConnectionString
PacketSize (sólo para SQL Server)	Devuelve el tamaño en bytes de los paquetes de red utilizados para comunicarse con SQL Server
WorkStationId	Devuelve una cadena que identifica al cliente, especificado en el atributo Workstation ID en la propiedad ConnectionString

Y los métodos más importantes disponibles

Método	Descripción
Open	Abre una conexión con los datos especificados en "ConnectionString"
Close	Cierra una conexión y libera todas las fuentes asociadas.

10.1. Utilizando el asistente

El asistente de conexión es desde luego la mejor forma de crear una cadena de conexión. Lo utilizaremos para generarla y luego la cancelaremos porque solo queremos extraer el texto de la cadena:



10.2. Almacenar la cadena de conexión

Las cadenas de conexión pueden variar a lo largo de la vida de nuestro programa, simplemente porque cambiamos de servidor de base de datos y queremos utilizar este nuevo más potente. Si hacemos esto obviamente nuestro programa perderá esa conexión al estar apuntando al antiguo. Esto lo podemos solucionar si utilizamos un fichero externo para almacenar la cadena de conexión.

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="Ejemplo_2___T12.Properties.Settings.NorthwindConnectionString"
            connectionString="Data Source=PROFESORC6\SQLEXPRESS01;Initial Catalog=Northwind;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>
</configuration>
```

Es el fichero XML que incorporan los proyectos y dónde el asistente de la cadena de conexión lo modificaba para incluirla.

Esta configuración es idéntica al fichero "web.config" de ASP.NET, la versión .NET para la generación de páginas web de servidor.

11. Objeto Command

Siguiendo con el modelo de objetos y una vez conectados a la base de datos el siguiente paso es trabajar con el objeto Command que nos permitirá realizar las operaciones habituales de las bases de datos. Permitirá ejecutar una consulta SQL o un procedimiento almacenado del servidor de base de datos. También podremos obtener un conjunto de resultados que trataremos más adelante.

Lógicamente el objeto Command lo utilizaremos a partir de una conexión existente y ejecutará una sentencia SQL sobre la conexión ya establecida. El lenguaje SQL es el lenguaje utilizado por las bases de datos para realizar operaciones con ellas: altas, bajas, consultas y modificaciones. Primero veamos las propiedades de este objeto:

Propiedad	Descripción
CommandText	Texto SQL de la consulta o de la operación con los datos
CommandType	Tipo de comando que se va a ejecutar. Puede ser: un procedimiento almacenado (StoredProcedure), obtener una tabla (TableDirect) o una sentencia SQL (Text)

CommandTimeout	Tiempo de espera para la ejecución del comando. El valor predeterminado es de 30 segundos.
Connection	Devuelve el objeto Connection asociado a este comando
Parameters	Colección de parámetros asociada a este comando

Pero más importantes quizás son los métodos que tenemos disponibles:

Método	Descripción
ExecuteNonQuery	Ejecuta la consulta de acción especificada en CommandText y devuelve el número de filas afectadas
CreateParameter	Crea un parámetro para el que después podemos definir una serie de características específicas como tipo de datos, valor, tamaño, ...
ExecuteReader	Ejecuta la consulta de acción especificada en CommandText y devuelve el objeto DataReader que permite acceder al conjunto de resultados obtenidos con la ejecución del comando.
ExecuteScalar	Este método se utiliza cuando deseamos obtener la primera columna de la primera fila del conjunto de registros, el resto de datos no se tendrá en cuenta. (por ejemplo para ejecuciones de sentencias SQL de tipo select count(*))
Prepare	Crea una versión compilada del comando en la fuente de datos.

11.1. Creación de un objeto Command

Como hemos dicho, las principales propiedades del objeto Command son CommandText (la sentencia SQL) y Connection (conexión sobre la que ejecutará el comando), así que por ejemplo podemos hacer:

```
// Abrimos la conexión
SqlConnection conexion = new SqlConnection(cadenaConexion);
conexion.Open();

// Definimos el comando para insertar una fila en la base de datos:
string sql = "insert into Employees (LastName,FirstName) values ('Wilde','Oscar')";

// Creamos un comando de ejecución:
SqlCommand comando = new SqlCommand();
comando.Connection = conexion;
comando.CommandText = sql;

// Ejecuto la consulta y se obtiene el número de registros afectados
int registros = comando.ExecuteNonQuery();
MessageBox.Show("Registros afectados:" + registros);

// Cierro la conexión
conexion.Close();
```

Por lo tanto, podemos realizar cualquier operación en la base de datos para insertar o borrar con la instrucción "Delete" de SQL. Para dejarlo perfecto deberíamos interceptar los errores que se puedan dar en la operación con:

```
try
{
    // Ejecuto la consulta y se obtiene el número de registros afectados
    int registros = comando.ExecuteNonQuery();
    MessageBox.Show("Registros afectados:" + registros);
}
catch
{
    // Procesamos el error
}

finally
{
    // Cerramos la conexión
    conexion.Close();
}
```

En esta parte del programa podemos distinguir entre dos tipos de instrucciones SQL las que recuperan datos o las que modifican datos (consultas de acción). Entre las segundas tenemos:

- Inserción de datos

  "Insert into tabla (campo1,campo2) values ('valor1','valor2')"
- Eliminación de datos

  "Delete from tabla where valor=1"
- Modificación de datos

  "Update tabla set dato1='valor1'

En estos casos simplemente obtendré el número de registros afectados, como hemos visto antes. El otro caso que nos queda es el más extenso y es cuando se realizan las consultas ya que lo que se devuelve es un conjunto de resultados más o menos complejo.

Otro ejemplo sin recoger el número de registros afectados:

```
// Abrimos la conexión
SqlConnection conexion = new SqlConnection(cadenaConexion);
conexion.Open();

// Creamos un comando de ejecución
SqlCommand comando = new SqlCommand();
comando.Connection = conexion;
comando.CommandText = "UPDATE Suppliers SET CompanyName = 'IES Comercio' WHERE SupplierID = 24";
comando.ExecuteNonQuery();

// Cierro la conexión
conexion.Close();
```

El mismo ejemplo recogiendo el número de registros afectados para comprobar si ha ido correcta la operación:

```
// Creamos la cadena de conexión
string cadenaConexion = "Data Source = MSI\\SQLEXPRESS;Initial Catalog = Northwind; " +
    "Integrated Security = True";

// Abrimos la conexión
SqlConnection conexion = new SqlConnection(cadenaConexion);
conexion.Open();

// Creamos un comando de ejecución
SqlCommand comando = new SqlCommand();
comando.Connection = conexion;
comando.CommandText = "UPDATE Suppliers SET CompanyName = 'IES Comercio' WHERE SupplierID = 24";

try
{
    int registrosAfectados = comando.ExecuteNonQuery();
    if (registrosAfectados == 1)
        MessageBox.Show("Actualización realizada correctamente", "Información",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
}

catch
{
    MessageBox.Show("Se ha producido un error en la actualización.", "Información",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
// Cierro la conexión
conexion.Close();
```

11.2. Crear comandos más robustos

La escritura de las sentencias SQL puede ser muy laboriosa y compleja porque en ocasiones tendremos que concatenar varios criterios para hacer una consulta o una inserción:

```
sql = "select * from datos where nombre='" + txt_nombre.text + "' and
ciudad='" + txt_ciudad.text;
```

La concatenación de los campos para realizar esta operación puede provocar varias deficiencias. Por ejemplo:

- Haya caracteres erróneos. Por ejemplo un apellido como "O'brian" daría un problema, porque el carácter ' ya has visto que es el separador de los literales. En este caso el usuario recibiría un error de ejecución
- Los usuarios pueden realizar una "inyección de SQL" que es uno de los modos de ataques más comunes para intentar acceder a sitios web. Esta técnica es muy utilizada y es la de meter código SQL dentro de un campo de datos.

Vemos entonces la necesidad de mejorar esa instrucción que hemos escrito antes para construir la sentencia y prevenir los errores por introducción de caracteres erróneos. Una forma sencilla aunque poco elegante sería:

```
string idAutor = txt_id.text.Replace("'", """);
```

Es decir, creamos otra variable donde hemos sustituido ese carácter ' por una comilla doble ". Entonces sí nos funcionará. Pero, claro, deberíamos crear tantas variables como campos. La mejor solución va a ser crear un comando parametrizado que simplemente nos va a sustituir unos valores por otros según le decimos la posición.

Select * from Clientes where clienteId= 'ABCD'

que sustituiremos por:

Select * from Clientes where clienteId= @clienteId

En esa variable @clienteId meteremos nuestro contenido del cuadro de texto de antes pero ahora la construcción de la SQL será siempre correcta, evitando tanto los caracteres erróneos como el fallo de seguridad de la "inyección de SQL". En inglés se llaman "placeholders", que vienen a ser unos "huecos" donde luego pondremos los valores definitivos. Veamos cómo quedaría y verás qué es sencillo. En la parte de la SQL es solo una cadena "string", toda seguida con esos indicadores:

```
string sql = "insert into Customers (CustomerID,CompanyName,ContactName,ContactTitle," +
    "Address,City,Region,PostalCode,Country,Phone,Fax)";
sql += " values (";
sql += "@CustomerID,@CompanyName,@ContactName,@ContactTitle," +
    "@Address,@City,@Region,@PostalCode,@Country,@Phone,@Fax)";
```

Y luego le asignamos los valores reales (omito parte del código para mostrar solo lo interesante):

```
// Definimos los objetos ADO.NET
SqlConnection conexion = new SqlConnection();
SqlCommand comando = new SqlCommand(sql, conexion);

comando.Parameters.AddWithValue("@CustomerID", txtCustomerID.Text);
comando.Parameters.AddWithValue("@CompanyName", txtCompanyName.Text);
comando.Parameters.AddWithValue("@ContactName", txtContactName.Text);
comando.Parameters.AddWithValue("@ContactTitle", txtContactTitle.Text);
comando.Parameters.AddWithValue("@Addres", txtAddress.Text);
comando.Parameters.AddWithValue("@City", txtCity.Text);
comando.Parameters.AddWithValue("@Region", txtRegion.Text);
comando.Parameters.AddWithValue("@PostalCode", txtPostalCode.Text);
comando.Parameters.AddWithValue("@Country", txtCountry.Text);
comando.Parameters.AddWithValue("@Phone", txtPhone.Text);
comando.Parameters.AddWithValue("@Fax", txtFax.Text);
```

Las diferencias son muy pocas, simplemente creamos una sentencia SQL mucho más sencilla al no tener que concatenar los valores de los cuadros de texto y luego hacemos la asignación. Es imprescindible que trabajes así. Con esto evitarás muchos problemas en la creación de la SQL, sobre todo cuando son muy extensas.

Para el caso de una actualización solo pondríamos la sentencia SQL adecuada, todo lo demás es igual:

```
// Definimos los objetos
string sql = "update Customers set";
sql += " CompanyName=@CompanyName,";
sql += " ContactName= @ContactName, ContactTitle=@ContactTitle,";
sql += " Address=@Address, City=@City, Region=@Region,";
sql += " PostalCode=@PostalCode, Country=@Country,";
sql += " Phone=@Phone, Fax=@Fax";
sql += " where CustomerID=@CustomerIDoriginal";

// Definimos los objetos ADO.NET
SqlConnection conexion = new SqlConnection(cadenaConexion);
SqlCommand comando = new SqlCommand(sql, conexion);

comando.Parameters.AddWithValue("@CompanyName", txtCompanyName.Text);
comando.Parameters.AddWithValue("@ContactName", txtContactName.Text);
comando.Parameters.AddWithValue("@ContactTitle", txtContactTitle.Text);
comando.Parameters.AddWithValue("@Addres", txtAddress.Text);
comando.Parameters.AddWithValue("@City", txtCity.Text);
comando.Parameters.AddWithValue("@Region", txtRegion.Text);
comando.Parameters.AddWithValue("@PostalCode", txtPostalCode.Text);
comando.Parameters.AddWithValue("@Country", txtCountry.Text);
comando.Parameters.AddWithValue("@Phone", txtPhone.Text);
comando.Parameters.AddWithValue("@Fax", txtFax.Text);
comando.Parameters.AddWithValue("@CustomerIDoriginal", lstEmpresas.SelectedItem);
```

Como ves es muy parecido a lo visto antes pero tenemos una diferencia muy grande e importante, fíjate al final de la sentencia SQL:

```
sql += " where CustomerID=@CustomerIDoriginal";
```

Significa que actualizaremos todos los registros que coincidan con ese identificador de registro. Como es clave única será sólo uno. ¿Y cuál debo actualizar? Pues precisamente el del valor de la fila seleccionado en el cuadro de lista. Además tenemos estos detalles:

- No necesitamos un DataReader porque no se devuelven datos.
- Se utiliza un comando dinámico "Update" mediante un objeto "Command". Este comando busca los campos del registro y los sustituye por los valores proporcionados por los cuadros de texto.
- No actualizamos el identificador único "ID" porque es el índice del elemento que debemos actualizar.
- El método "ExecuteNonQuery" devuelve el número de filas afectadas, que solo será una al decirle la clave única del registro que debe modificar. Una vez realizada llamamos a rellenar el cuadro de lista otra vez.

Para borrar registros es el caso más sencillo de todos porque la instrucción SQL es más sencilla:

```
// Definimos los objetos
string sql = "delete from Customers ";
sql += " where CustomerID=@CustomerIDoriginal";

// Definimos los objetos ADO.NET
SqlConnection conexion = new SqlConnection(cadenaConexion);
SqlCommand comando = new SqlCommand(sql, conexion);

comando.Parameters.AddWithValue("@CustomerIDoriginal", lstEmpresas.SelectedItem);
```

Nos funciona pero nos da un problema ya que como este cliente tiene pedidos suyos en la tabla de los pedidos (orders), nos avisa de que se rompe la relación. Es normal, imagina el caso del banco con las operaciones del cajero de un usuario. Si el banco da de baja al usuario de la tabla de clientes deberá borrar también los movimientos de la tabla de movimientos. Sino, puesto que la tabla de movimientos tenía como índice el número de cliente no se podrán acceder a sus datos ya que ese número de cliente desaparece al borrar su ficha. Luego por coherencia de datos al borrar un "dato maestro" habría que borrar todos los datos asociados de las tablas.

La idea de todo esto es cómo construir la sentencia SQL con esos parámetros que nos va a ahorrar muchos problemas por la inclusión de caracteres extraños en los comandos SQL que construyamos.

12. Datos conectados

Hemos visto antes la recuperación de datos en un modelo conectado: DataReader. Realizamos una conexión, hacemos la consulta, escribimos los datos (DataReader) y nos conectamos. El otro modelo es el "desconectado". Donde se realiza una conexión, se recuperan los datos en un DataSet y luego se desconecta. Después, si hay que hacer alguna actualización de datos un "DataAdapter" actualizará los datos en el servidor. Ahora nos centraremos en ese modelo conectado con el DataReader.

Una vez que tenemos la conexión ahora tenemos dos pasos que realizar:

- Especificar qué datos queremos recuperar.
- Recuperar los datos.

Para realizar el primer paso utilizaremos un objeto de tipo comando "SqlCommand" que es muy fácil de definir:

```
SqlCommand comando = new SqlCommand();
```

Veamos un ejemplo de cómo crearlo de tres formas distintas:

```
// Método 1. Creándolo en dos pasos
SqlCommand comando = new SqlCommand();
comando.Connection = conexion;

// Método 2. Creándolo en un solo paso
SqlCommand comando = new SqlCommand("…comando…", conexion);

// Método 3. Utilizando el método CreateCommand
comando = conexion.CreateCommand();
```

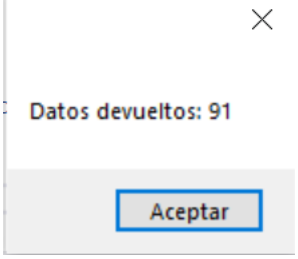
Por supuesto partimos de un objeto conexión ya creado y que abriremos para la ejecución del comando. El segundo método es el más sencillo de escribir y el que veremos enseguida. Pero antes quiero explicarte esta ejecución:

```
int numRegistros = Convert.ToInt32(comando.ExecuteScalar());
```

El método **ExecuteScalar** ejecuta la consulta y devuelve la primera columna de la primera fila del conjunto de resultados que devuelve la consulta. Se omiten todas las demás columnas y filas. En este caso lo que nos devuelve es el número de filas que va a recuperar en la consulta porque vamos a hacer un "select count(*)" de la tabla, valor muy útil para mostrar al usuario y para realizar algunas operaciones con índices ya que sabemos el número de filas de la tabla devuelta. Vamos a verlo en un ejemplo:

```
string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog = Northwind; Integrated Security = True";

// Creamos el objeto conexion
SqlConnection conexion = new SqlConnection(cadenaConexion);
// Creamos el objeto comando en un solo paso
SqlCommand comando = new SqlCommand("select count(*) from Customers", conexion);
// Abrimos la conexion
conexion.Open();
int total = Convert.ToInt32(comando.ExecuteScalar());
MessageBox.Show("Datos devueltos: " + total);
```

Como vemos ejecutamos un "select count(*)" que nos devuelve el número de filas. Como esa sentencia SQL solo devuelve ese valor coincide con la filosofía del comando de devolver el valor de la primera fila y primera columna:



La sentencia SQL ejecuta una operación para contar el número de filas.

Nota: La diferencia entre ExecuteNonquery y ExecuteScalar está clara. El primer método nos dice cuántos registros se han visto afectados por el comando ejecutado y el segundo método nos devuelve el valor de la primera fila/columna de la consulta.

Un objeto DataReader permite la navegación hacia delante y de sólo lectura de los registros devueltos en la consulta. A diferencia del resto de objetos que siguen un esquema desconectado los DataReader permanecen conectados durante todo el tiempo que realiza el recorrido por los registros que contienen. Las clases que utilizan este tipo de operaciones son SqlDataReader y OleDbDataReader según sea una conexión para SQL Server o para Bases de datos Ole DB. Recuerda que la técnica importante de ADO.NET es la de los objetos desconectados pero aun así tiene uno para operar conectado como este.

Los métodos más importantes del DataReader son:

Método	Descripción
Close	Cierra el DataReader liberando los recursos
GetXXX	Conjunto de métodos para obtener los valores de las columnas: GetBoolean(), GetInt32(), GetString(). Como parámetro se especifica el número de orden de la columna a recuperar
GetName	Devuelve el nombre de la columna del índice especificado
GetValue	Devuelve el valor de la columna en formato nativo con el índice especificado
GetValues	Toma un matriz Object y la rellena con los valores de todas las columnas del RecordSet
NextResult	Desplaza el puntero al siguiente conjunto de resultados si lo hubiera
Read	Desplaza el cursor actual al siguiente registro permitiendo obtener los valores del mismo a través del objeto DataReader. Devuelve True si existen más registros o False si no hay más. La posición inicial es antes del primer registro, por lo tanto hay que comenzar con un Read (mira el ejemplo siguiente)

12.1. Recuperar los datos

ExecuteReader es el método que utilizaremos para recuperar los datos. Este método nos va a devolver un resultado en forma de filas y columnas que podremos tratar como una tabla. Cada fila de esta tabla de resultados, **DataRecord**, tiene los siguientes métodos:

- **Get[Tipo de datos]**. Tendremos tantos como tipos de datos, GetByte para recuperar un dato de tipo Byte, GetBoolean, ... Si no sabemos el tipo de datos de la columna podemos utilizar la genérica "GetValue" que devuelve un valor de tipo "Object".
- **GetName**. Recupera el nombre de la columna.
- **GetOrdinal**. Para recuperar la columna por el número ordinal indicado.

- **IsdbNull**. En las bases de datos se pueden almacenar valores nulos. Con este método comprobaremos que no hay un valor nulo en esa columna. Es importante prever este valor porque es fácil que se pueda dar una excepción por un valor nulo no esperado.

Veamos una forma de declarar un DataReader y lo que implica su estado de conectado

```
SqlDataReader sqlDR = comando.ExecuteReader(CommandBehavior.CloseConnection);
```

Tenemos un extraño parámetro, esto es porque si recuperamos un objeto DataReader de un componente, el DataReader permanece conectado y no hay ninguna forma de desconectarlo. Para resolver este problema, configuraremos el método ExecuteReader de un objeto SqlCommand de forma que la conexión del DataReader se cierre automáticamente al cerrar el DataReader. Si pasamos el indicador System.Data.CommandBehavior.CloseConnection al método ExecuteReader, la conexión del DataReader se cerrará cuando se cierre el DataReader.

Es decir, al llamar a este método, puede que sea necesario un determinado modo de trabajo denominado comportamiento de comandos. ExecuteReader tiene una sobrecarga que toma un argumento del tipo CommandBehavior. Estos son los comandos que podemos incluir en la consulta:

Comportamiento	Descripción
CloseConnection	Cierra automáticamente la conexión al cerrar el lector de datos.
Default	Establecer esta opción equivale a llamar a ExecuteReader sin parámetros.
KeyInfo	Esta consulta devuelve sólo información de clave principal y metadatos de columna.
SchemaOnly	Esta consulta devuelve sólo metadatos de columna.
SequentialAccess	Activa el lector para cargar datos como una secuencia de cadena.
SingleResult	Sólo devuelve el primer conjunto de resultados.
SingleRow	Esta consulta debe devolver una única fila.

Cuando se recupera una gran cantidad de datos, mantener abierta la conexión se vuelve un problema. Para resolver esto, el DataReader es un flujo de sólo hacia delante y sólo lectura devuelto desde la base de datos. En la memoria se mantiene sólo un registro a la vez. Debido a que su funcionalidad es muy específica (o limitada), es “ligero”. Por eso es interesante cerrarlo a la vez que lo ejecutamos como hemos visto antes. Para acceder a cada uno de las columnas de la tabla resultante podemos hacer así:

```
// Accediendo a un campo mediante su nombre
    Datos["Apellidos"]
// Accediendo a un campo mediante su posición en la tabla. Devuelve un Object.
    Datos[1].ToString()
```

Accediendo a un campo mediante su posición y la función GetString. Si se sabe qué tipo de dato contiene la columna se pueden evitar conversiones utilizando directamente la función apropiada según el tipo de dato de la columna.

```
Lector.GetString(1)
'o Getxxx según el tipo de datos o utilizar el genérico GetValue
```

Veamos cómo hacer una lectura de datos:

```
if (sqlDR.HasRows)
  {
        while (sql_rd.Read)
              Listbox1.AddItem(sqlDR.GetValue(0) +" "+ sqlDR.GetValue(1));
  }
```

La propiedad HasRows comprueba primero si hay filas en el resultado. Si es así comenzamos un bucle de "solo hacia adelante" utilizando el método "Read". Recuerda la particularidad de estos "datareaders" que solo permiten realizar una lectura desde el primero hasta el final, sin posibilidad de movernos registros hacia atrás.

La recuperación de las columnas la podemos hacer por cualquiera de los métodos que hemos visto antes pero siempre es más rápida acceder a su número de columna GetValue(1). Estos objetos son útiles cuando necesitemos una forma rápida de hacer una consulta concreta y para rellenar controles auxiliares. Acuérdate de que si no lo remediamos se queda una conexión permanente con el servidor por eso debemos ponerle una forma de comportamiento haciendo que se cierre la conexión al hacer la consulta.

EJEMPLO

Vamos a realizar una consulta completa metiendo el resultado en una matriz dinámica de tipo ArrayList. Creamos una aplicación Windows y ponemos un control de tipo cuadrícula DataGridView. Realizamos ahora la consulta del DataReader en el evento clic de un botón:

```
using System.Data.Common;
using System.Data.SqlClient;
using System.Collections;

namespace Ejemplo15___T12
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        // Creamos la cadena de conexión
        string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESS;Initial Catalog =
Northwind; Integrated Security = True";

        private void Form1_Load(object sender, EventArgs e)
        {
            ArrayList matrizResultado = new ArrayList();
            using (SqlConnection conexion = new SqlConnection(cadenaConexion))
            {
                SqlCommand comando = new SqlCommand("Select * from Suppliers",
conexion);

                conexion.Open();
                SqlDataReader resultado =
comando.ExecuteReader(CommandBehavior.CloseConnection);
                if (resultado.HasRows)
                {
                    foreach (DbDataRecord dbregistro in resultado)
                        matrizResultado.Add(dbregistro);

                }
            } //En este momento se hace un conexion.dispose automático

            // Enlazamos con el DataGridView
            dGridViewResultado.DataSource = matrizResultado;

        }
    }
}
```

La novedad en este ejemplo es que en lugar de recoger los valores individualmente con "GetValue" hemos recuperado la fila completa con "DbDataRecord" que hemos ido añadiendo a un "arraylist". Al trabajar de esta forma y poner el resultado en una cuadrícula nos olvidamos del objeto "OleDbConnection" ya que podemos estar ya desconectados definitivamente del servidor.

Como detalles podemos ver que la cuadrícula resultante detecta que son elementos de solo lectura y así lo activa en las celdas. De todas formas habrás observado que estamos trabajando con pocas filas en nuestros resultados.

Es decir, en estos sencillos ejemplos se nos muestran unos pocos registros. Pero ¿qué pasaría si tuviéramos miles de filas como respuesta?. Pues que el usuario tendría que esperar la ejecución y devolución de todos los datos hasta poder tener el control del programa. Esto es porque estamos ejecutando de forma síncrona, es decir, conectamos, hacemos la consulta y recuperamos todos los datos. A medida que van llegando esos datos lo metemos en una matriz y cuando termina el bucle hacemos el enlace. Esto es válido para no muchos registros pero si nuestras tablas tienen muchos deberíamos utilizar una forma "asíncrona" de recuperar los datos.

Es un momento interesante para aprender una nueva instrucción: "using" que declara que vamos a utilizar un objeto durante un corto espacio de tiempo. Cuando terminamos de utilizar el objeto inmediatamente se liberan los recursos. El código sería:

```
using (objeto)
{
...
}
```

Al finalizar el uso del objeto fuerza a un "dispose()" del objeto que es lo mismo que destruirlo y así liberamos los recursos. En nuestro objeto de conexión equivaldría a un "close()". En un sencillo ejemplo de prueba de conexión quedaría:

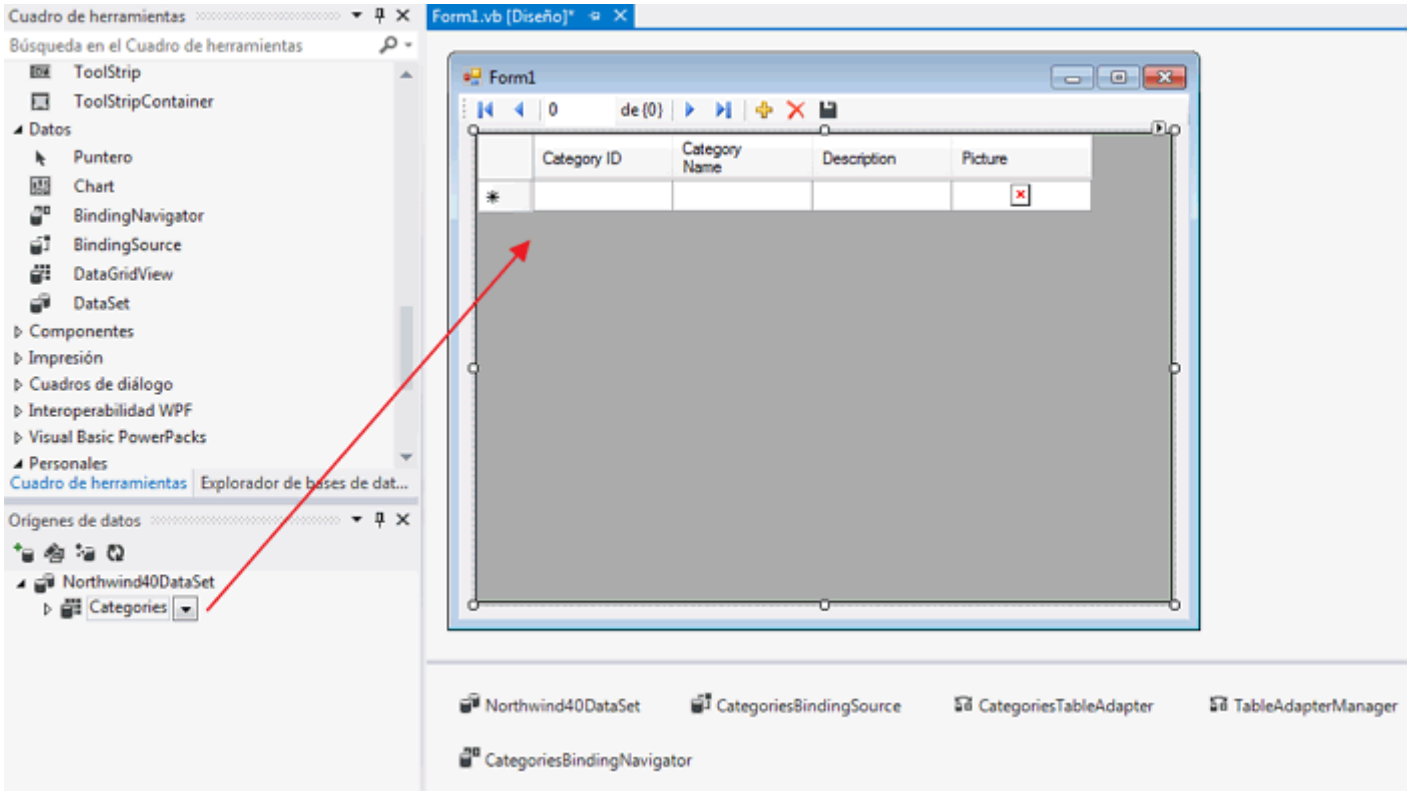
```
using System.Data.SqlClient;

namespace Ejemplo16___T12
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // Creamos la cadena de conexión
        string cadenaConexion = "Data Source = SERVIDOR\\SQLEXPRESSS;Initial Catalog =
Northwind; Integrated Security = True";

        private void Form1_Load(object sender, EventArgs e)
        {
         try
          {
           using (SqlConnection conexion = new SqlConnection(cadenaConexion))
            {
             //Intentamos abrir la conexion
             conexion.Open();
             lblEstado.Text = "Versión del servidor: " + conexion.ServerVersion + "\n";
                    lblEstado.Text += "La conexión es: " + conexion.State.ToString();

            }
          }
            catch (Exception ex)
            {
                //Vemos el error producido:
                lblEstado.Text = "Error en la conexión de la base de datos. ";
                lblEstado.Text += ex.Message;
            }
            finally
            {
                // Ha ido todo bien cerramos la conexión
                //conexion.close();
                // lblEstado.Text = "La conexión está ahora: " +
conexion.State.ToString();
            }

        }
    }
}
```

Es decir, cuando hemos dejado de utilizar el objeto, después del using() se destruye, por lo tanto, no hace falta cerrarlo.

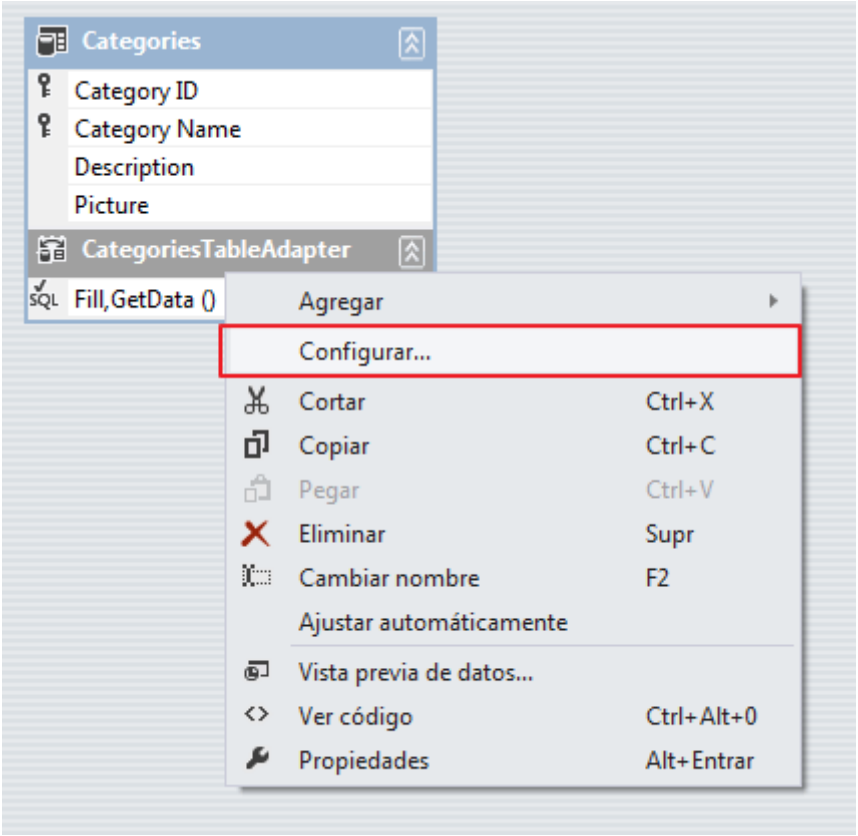
16. Actualizar datos

Una vez hechas las operaciones con las tablas para filtrar y ordenar nos queda la modificación de los datos y su volcado a la base de datos. Tendremos que realizar, por tanto, una sincronización de los datos.

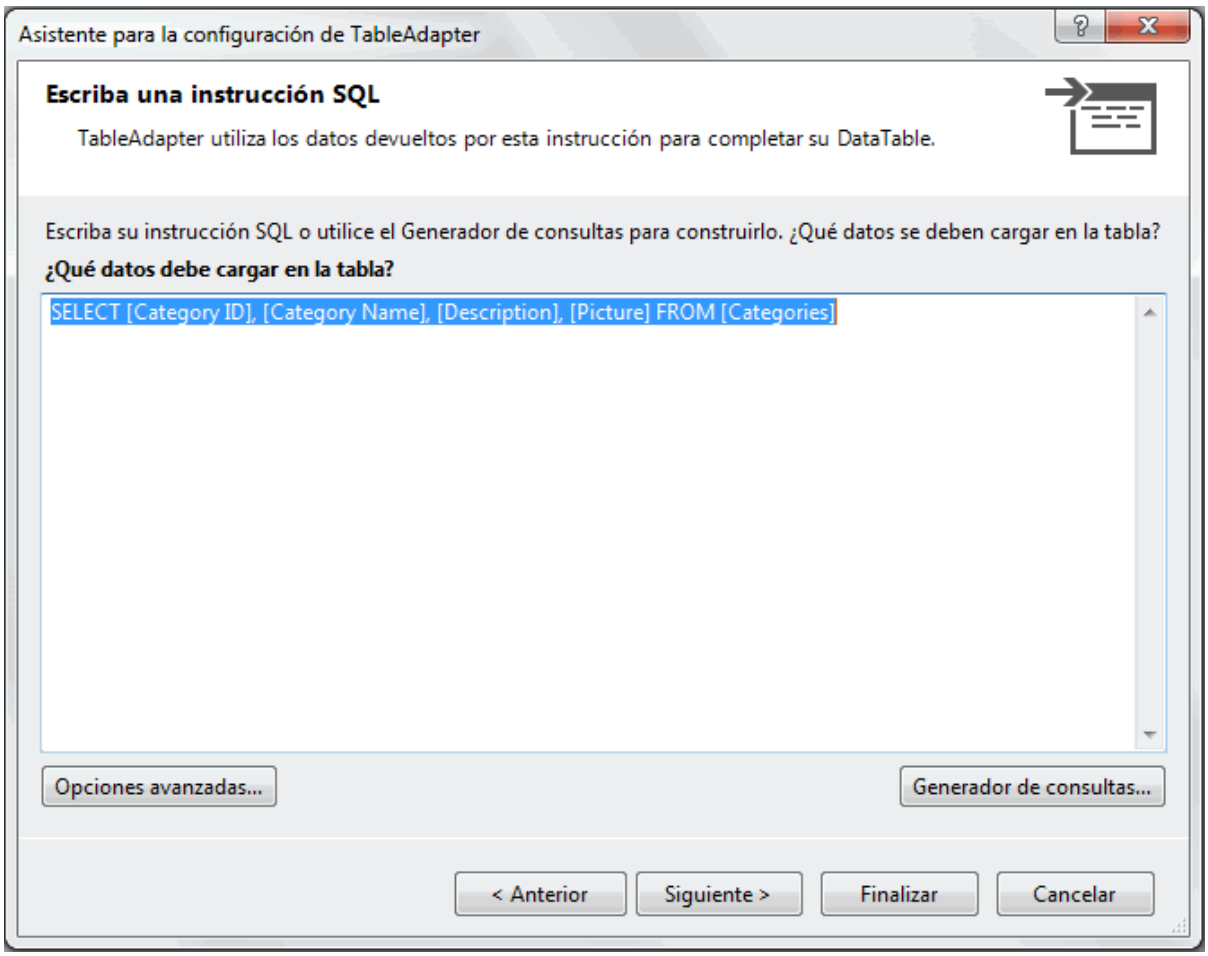
Vamos a repetir como se generan los comandos de actualización con el IDE. Partimos de una tabla sencilla y la arrastramos a nuestro IDE, por ejemplo, la de Categorías.



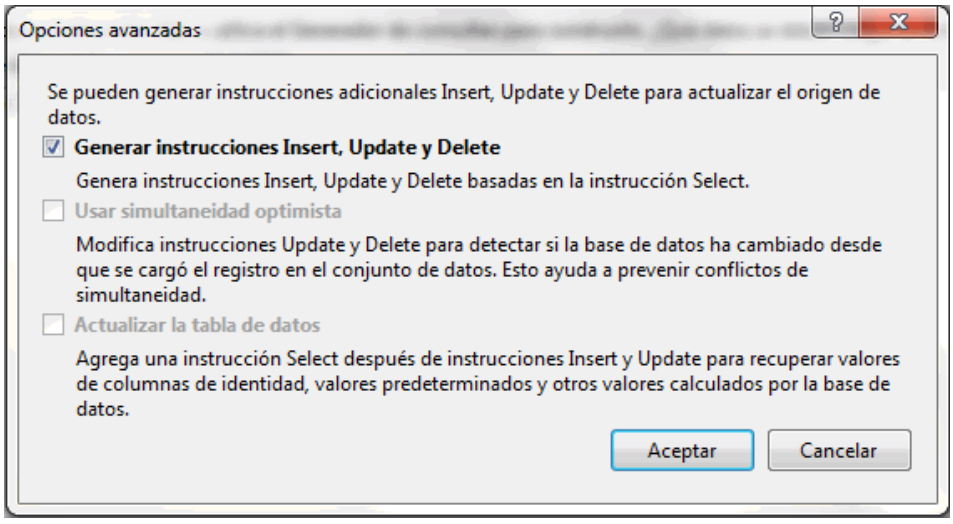
Si vemos los detalles sabemos que nos ha añadido un Dataset:



Vamos a seleccionar "Configurar":



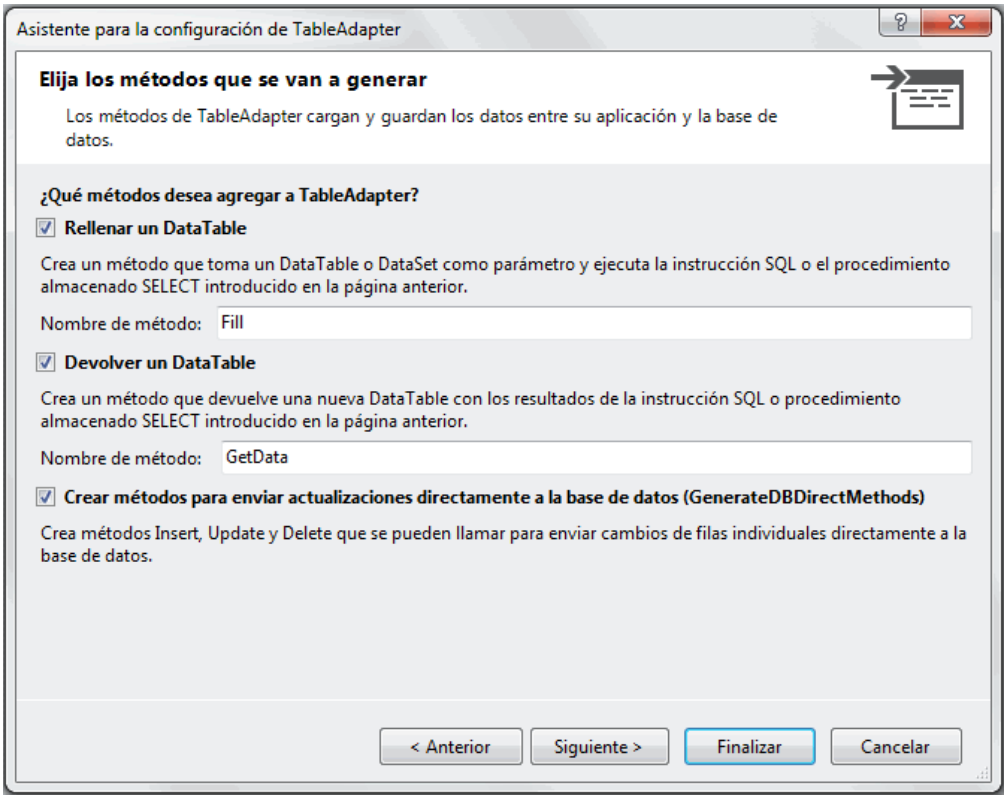
Como vemos, esa es la instrucción que ha utilizado para rellenar el Dataset. Si pulsamos en las opciones avanzadas:



Que nos van a permitir:

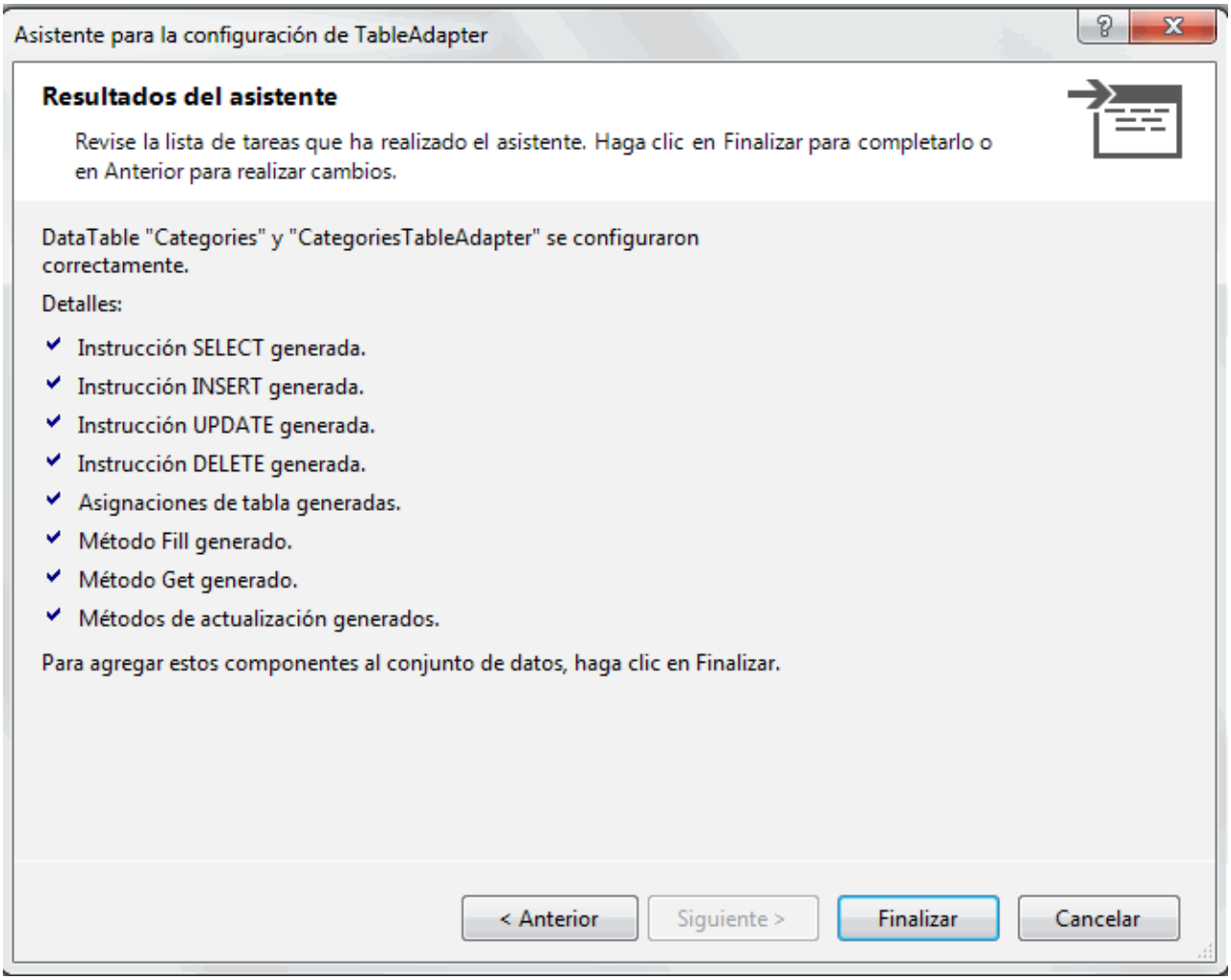
- Especificar que en el adaptador incluya los comandos de actualización de datos
- Utilizar la concurrencia optimista
- Actualizar la tablas

Aceptamos y pulsamos en siguiente:

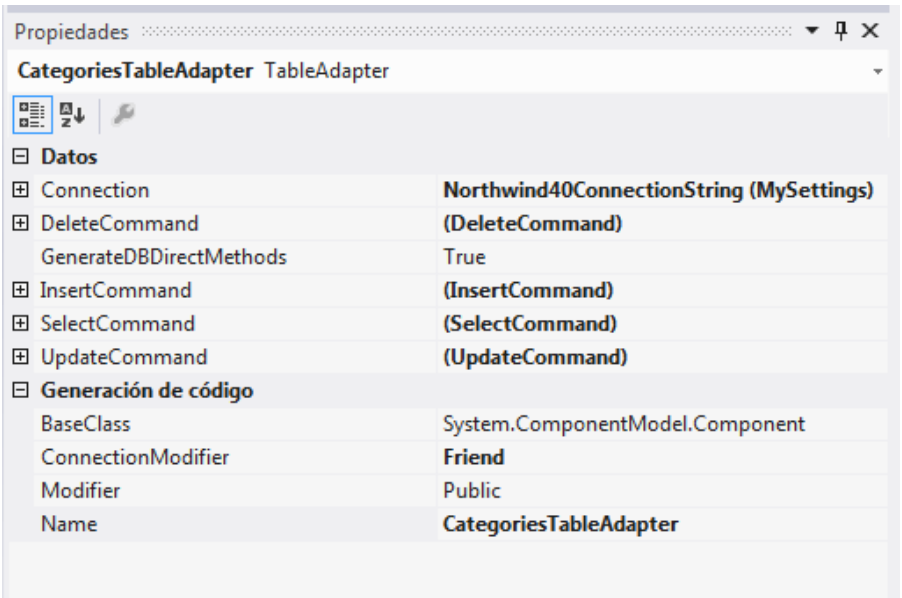


Nos avisa de que rellenará los datos con el método Fill, y que con Getdata obtendrá los valores.

Finalmente:



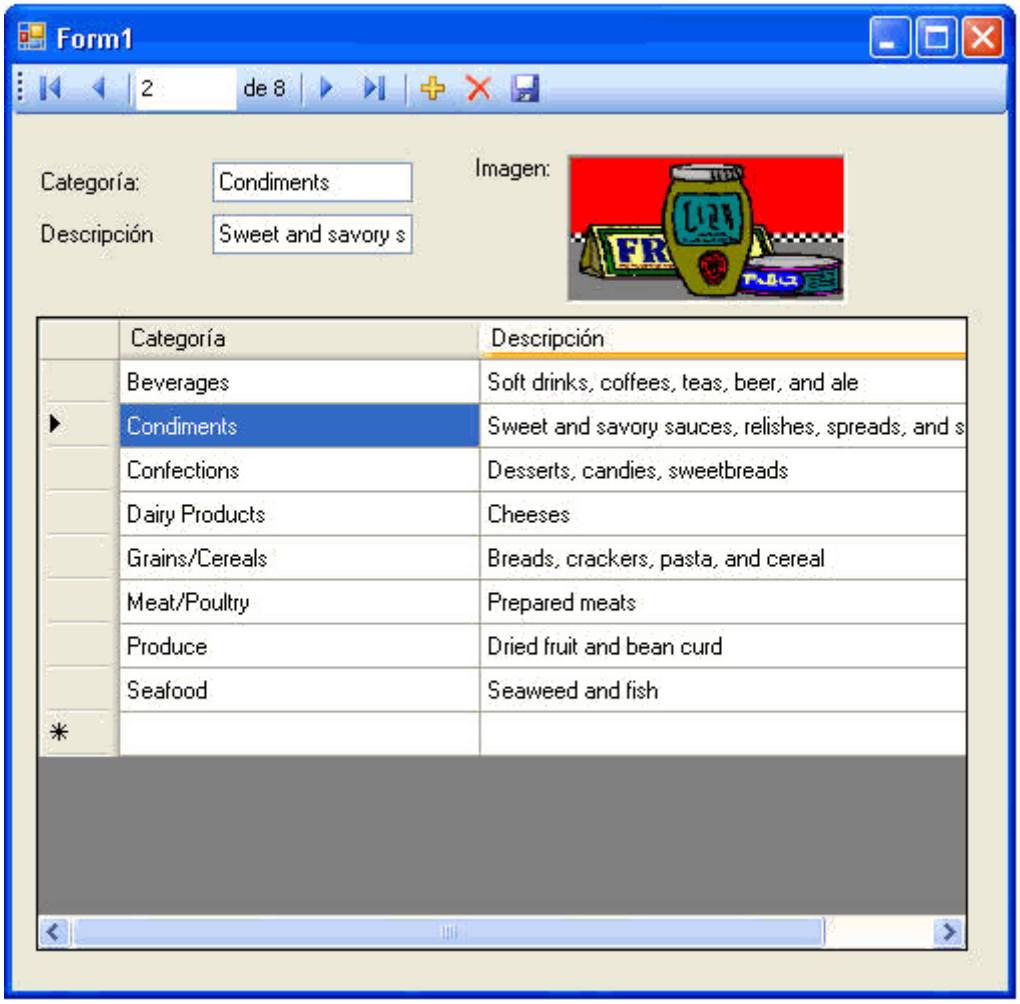
Nos genera todo lo necesario para actualizar los datos de las filas. Si nos fijamos en las propiedades del adaptador tendremos todas las instrucciones:



Ejercicios

Ejercicio 1

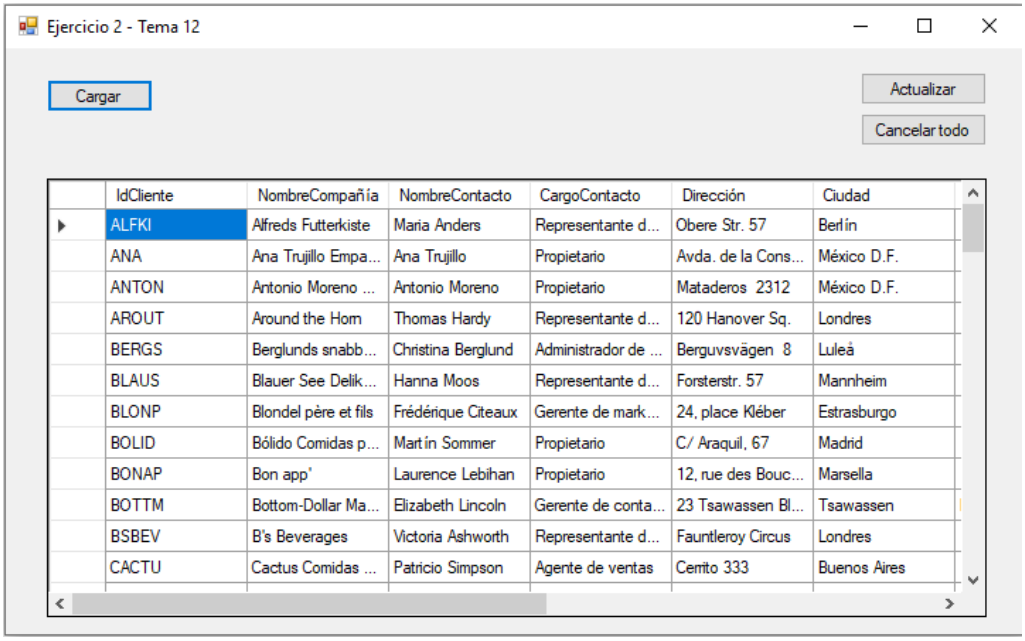
Crea la explotación completa de la tabla de categorías, con una cuadrícula que solo permita añadir y borrar y con los campos para la edición:



Nota: La columna del identificador no se muestra y en la cuadrícula tampoco la imagen. En la imagen el contenido se debe ajustar automáticamente para que quepa en el control. Pon los títulos de las columnas correctamente.

Ejercicio 2

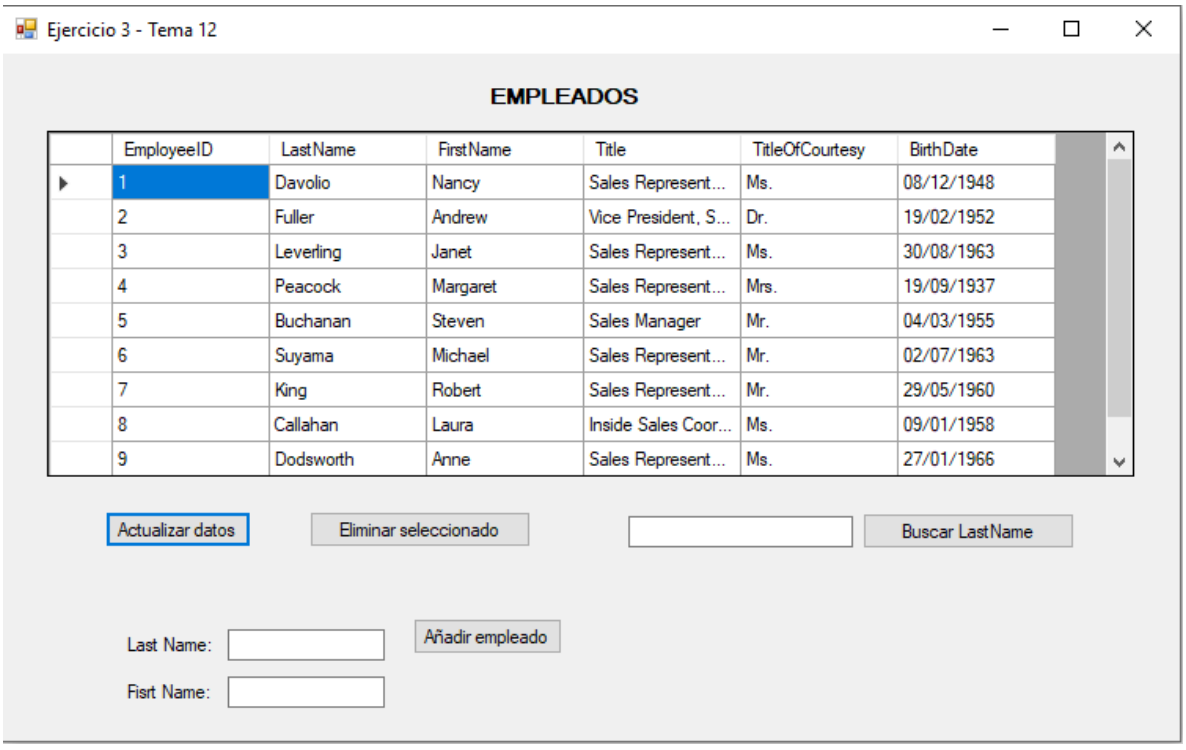
Crea un formulario para la tabla Clientes de la base de datos de ejemplo northwind.



Realiza el código manualmente.

Ejercicio 3

Crea el siguiente formulario para la tabla Employees de la base de datos northwind.



Nota: Utiliza los controles que se generan al crear el origen de datos e insertar su correspondiente DataGridView en el formulario.

Ejercicio 4

Crea el siguiente formulario para las tablas Customers y Orders de la base de datos northwind.

Según el cliente seleccionado en el Combobox deberá mostrar los datos del mismo en los cuadros de texto y sus pedidos en el DataGridView.