

Bloque 2. Visual Basic .NET

UT 3. Confección de interfaces de usuario y creación de componentes visuales

TEMA 12**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), que comentaremos ahora para poder situar estas tecnologías.

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.

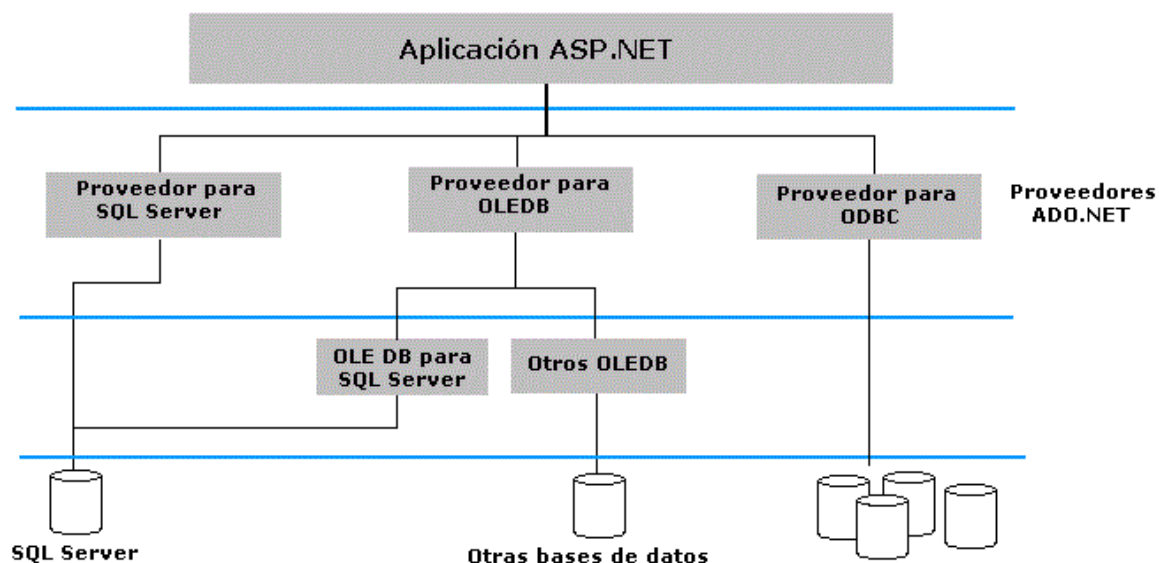
Con ADO.NET se consigue estar conectado al servidor sólo lo estrictamente necesario para realizar la operación de carga de datos en el DataSet, así se minimiza el tiempo de conexión y de uso de la base de datos. Se pueden conectar más usuarios, disminuyen los tiempos de respuesta y se aceleran las operaciones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

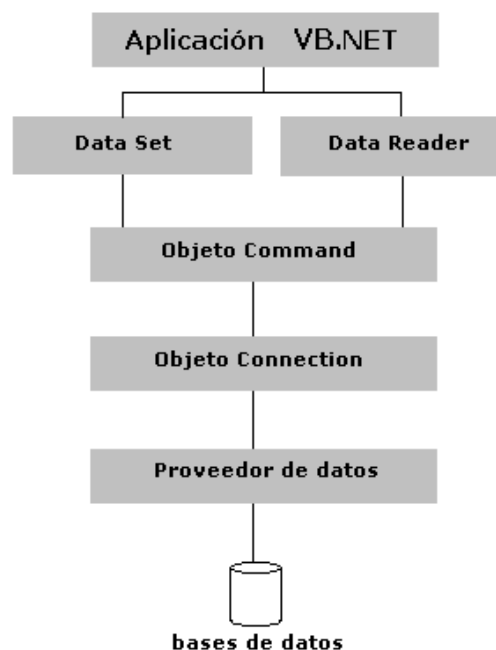
Pero... si estoy trabajando con muchos datos esto será un inconveniente porque ¡¡tendré que volver a cargarlos cuando me conecte otra vez!! Pues sí, lo que parece que no es la mejor solución: cargar una copia completa de los datos de trabajo tiene algunos detalles que lo convierten en ventajas. Las ventajas 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.

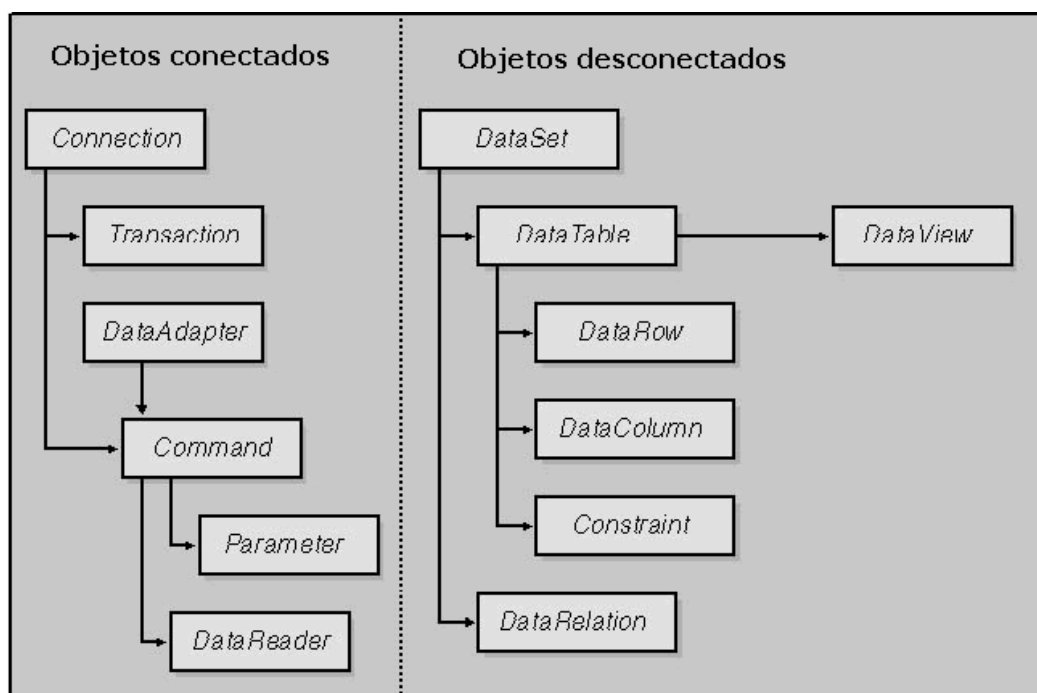


Para poder trabajar con datos en nuestras aplicaciones necesitaremos 6 elementos. Puede haber alguna variación en algún caso especial, pero hazte idea de que estas son las seis partes que necesitaremos para acceder a datos:

- Origen de datos, es decir, los datos en el formato admitido: SQL Server, Access, Oracle, ...
- Proveedor para acceder a esa base de 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 VB.NET.

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

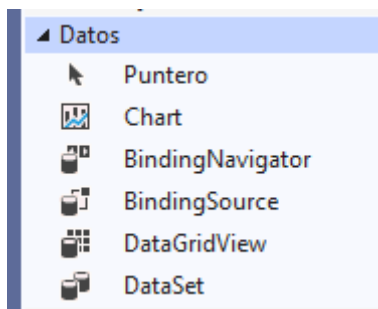


Como en otras ocasiones es muy útil tener esta estructura a mano porque nos servirá para comprender la jerarquía de los objetos y nos ayudará al escribir el código porque sabremos los objetos que debemos utilizar.

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 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 DB. 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 DB. 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 de 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:

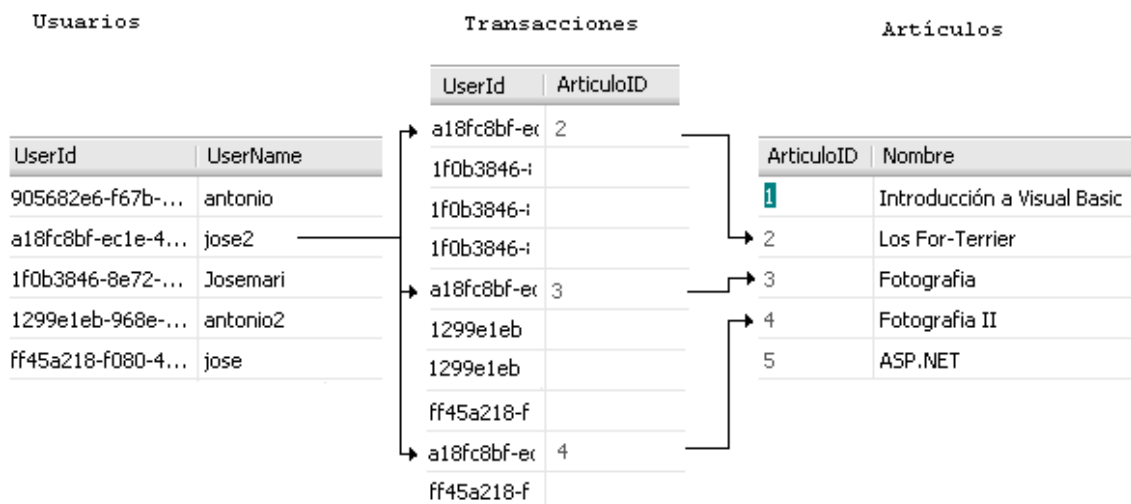
- Usuario
- Libro
- Fecha de compra
- ¿Está pagado?

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 "ArticuloID". 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.



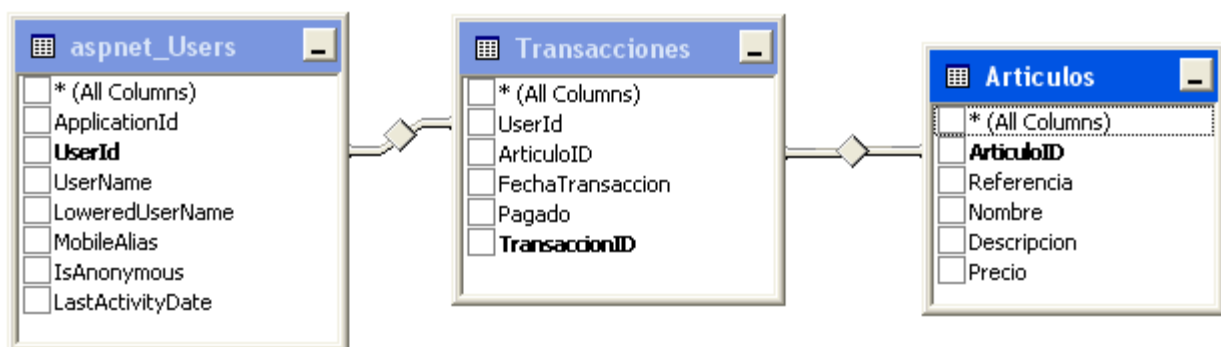
Así que en cada compra que hagamos meteremos una línea en la tabla de transacciones indicando quién y qué ha comprado. Aquí la relación es "muchos a muchos" ya que muchos usuarios pueden comprar el mismo libro e igualmente muchos libros pueden ser comprados por un usuario.



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

Usuarios		Transacciones		Artículos	
UserId	UserName	UserId	ArticuloID	ArticuloID	Nombre
905682e6-f67b-4... antonio		a18fc8bf-ec1e-4... jose2	3	1	Introducción a Visual Basic
1f0b3846-8e72-4... Josemari		1f0b3846-i 1f0b3846-i		2	Los For-Terrier
1299e1eb-968e-4... antonio2		1f0b3846-i a18fc8bf-ec1e-4... 1299e1eb	3	3	Fotografia
ff45a218-f080-4... jose		1299e1eb 1299e1eb		4	Fotografia II
		ff45a218-f a18fc8bf-ec1e-4... ff45a218-f	3 4	5	ASP.NET

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.

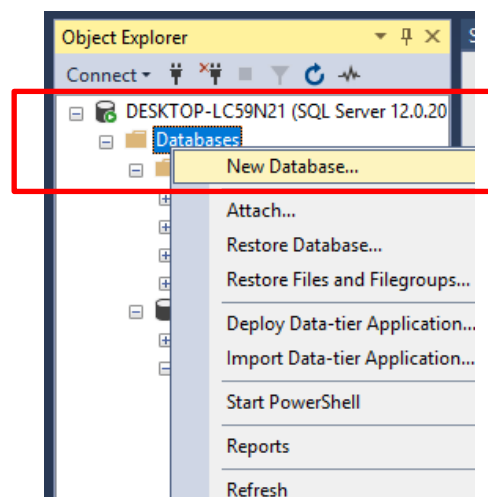
3. Los primeros pasos con SQL Server

Existen varias versiones de SQL Server, una de ellas es gratuita y se llama SQL Server Express. Este servidor tiene gran potencia y nos metemos de lleno en el mundo profesional. Utilizaremos SQL Server 2017 Express.

Necesitaremos tener instalados el servidor de bases de datos SQL Server 2017 Express y el entorno SQL Server Management Studio 18.4 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 18:

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.

```
SQLQuery1.sql - D:\59N21\Amparo (52)) *  X
USE [Northwind]
GO
/***** Object: Table [dbo].[CustomerDemographics]    Script Date: 08/12/2011 11:46:03 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[CustomerDemographics](
    [CustomerTypeID] [nchar](10) NOT NULL,
    [CustomerDesc] [ntext] NULL,
    CONSTRAINT [PK_CustomerDemographics] PRIMARY KEY NONCLUSTERED
(
    [CustomerTypeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/***** Object: Table [dbo].[Region]    Script Date: 08/12/2011 11:46:03 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Region](
    [RegionID] [int] NOT NULL,
    [RegionDescription] [nchar](50) NOT NULL,
    CONSTRAINT [PK_Region] PRIMARY KEY NONCLUSTERED
(
    [RegionID] ASC
)
```

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

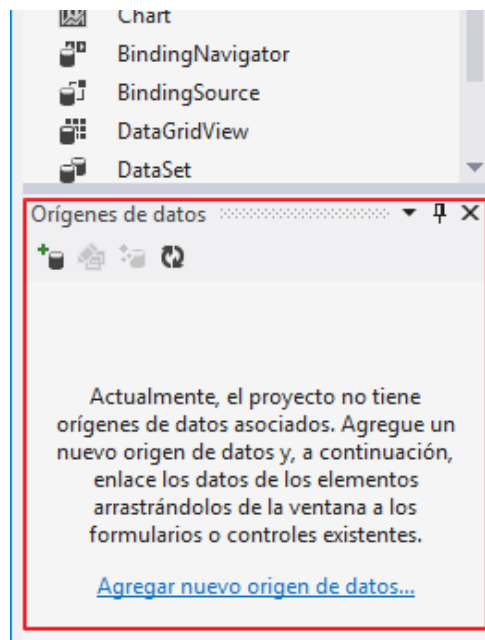
Ejemplo:

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

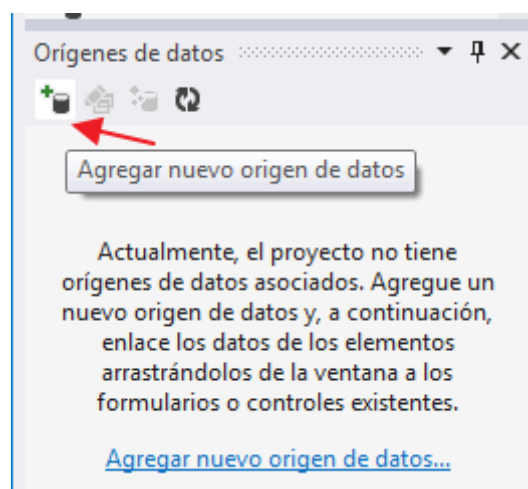
3. Nos fijamos en las propiedades de la base de datos Northwind en Server Name o Server "SERVERSQLEXPRESS" porque lo tendremos que utilizar para la conexión desde Visual Studio. Para verlas, sobre la bbdd botón derecho → Propiedades.

3.1. Conexión de la base de datos

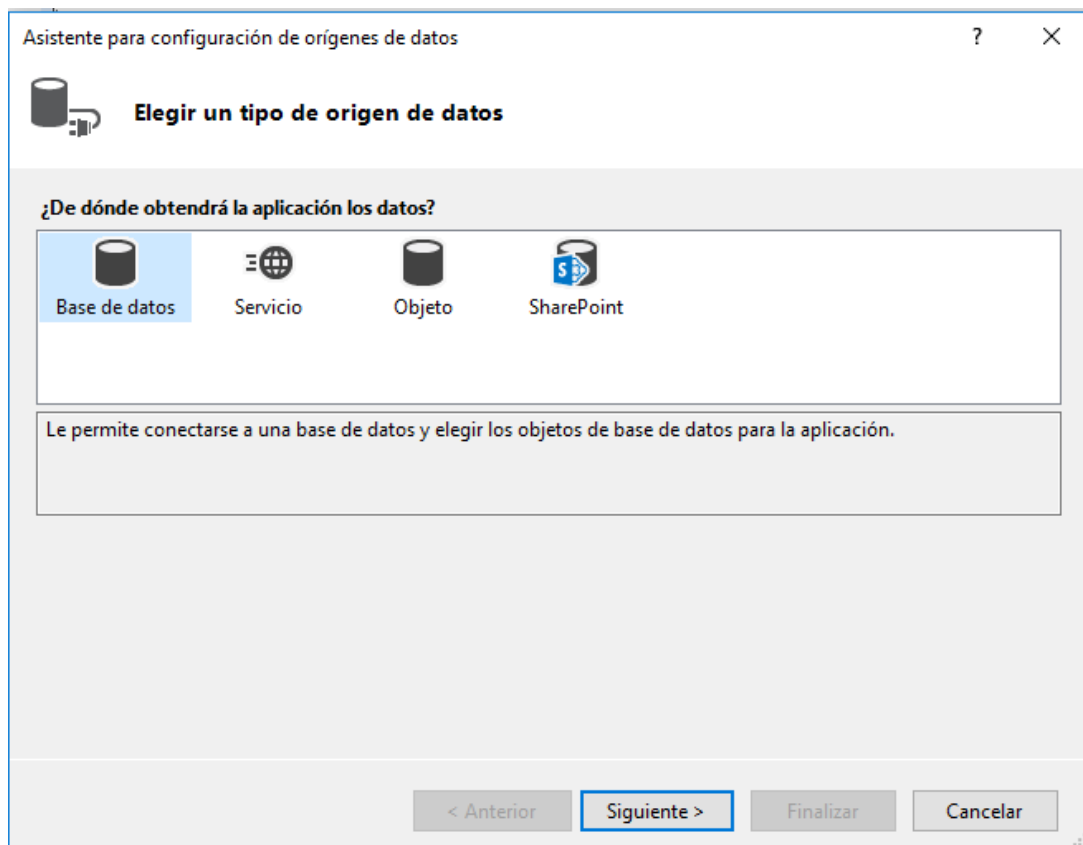
Vamos a establecer una primera conexión para explorar una base de datos de prueba. Creamos una nueva aplicación, y haremos clic "orígenes de datos" que tenemos como segunda solapa del cuadro de herramientas. Si no te aparece la activas en el menú "Ver":



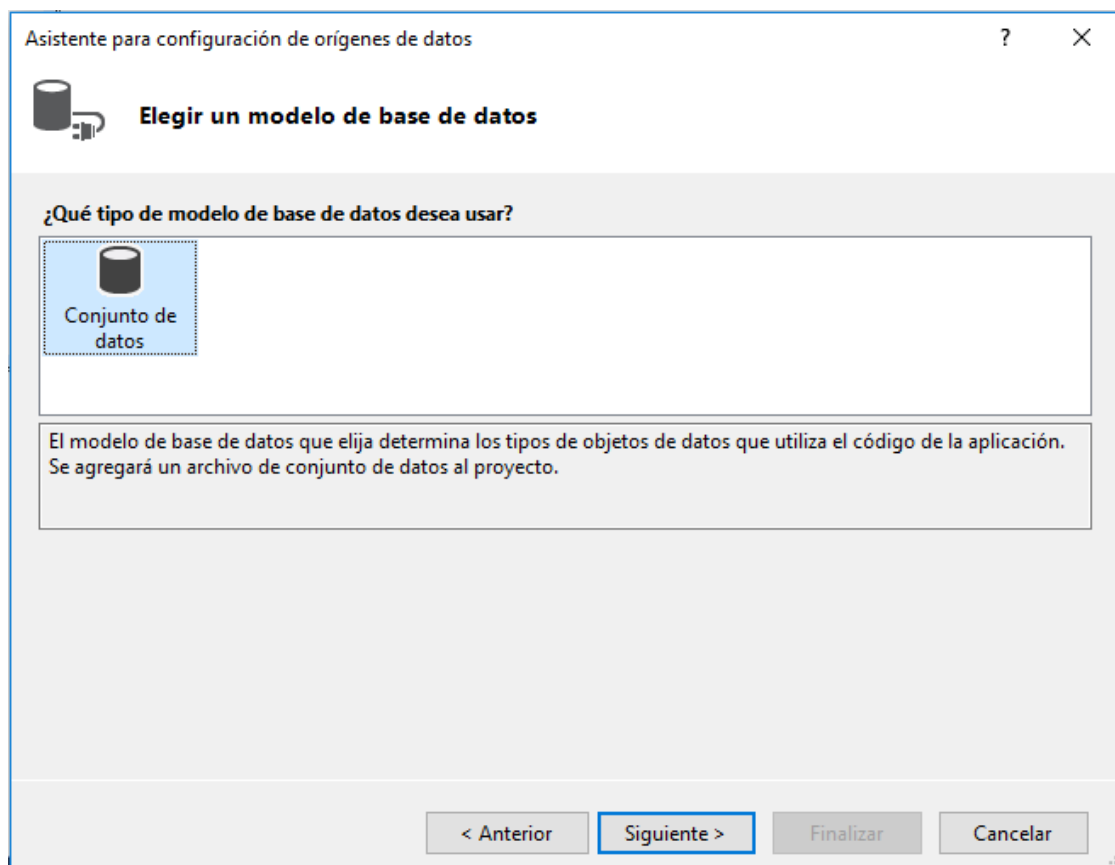
Hacemos clic en el enlace o en el primer icono de su barra de herramientas:



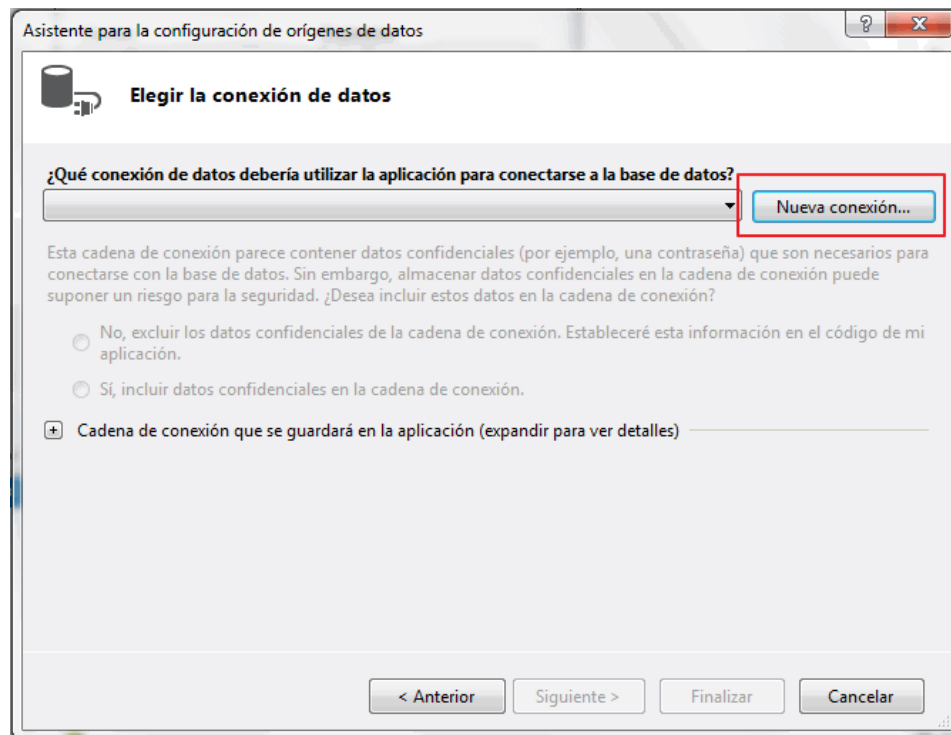
Le indicamos que va a ser una base de datos:



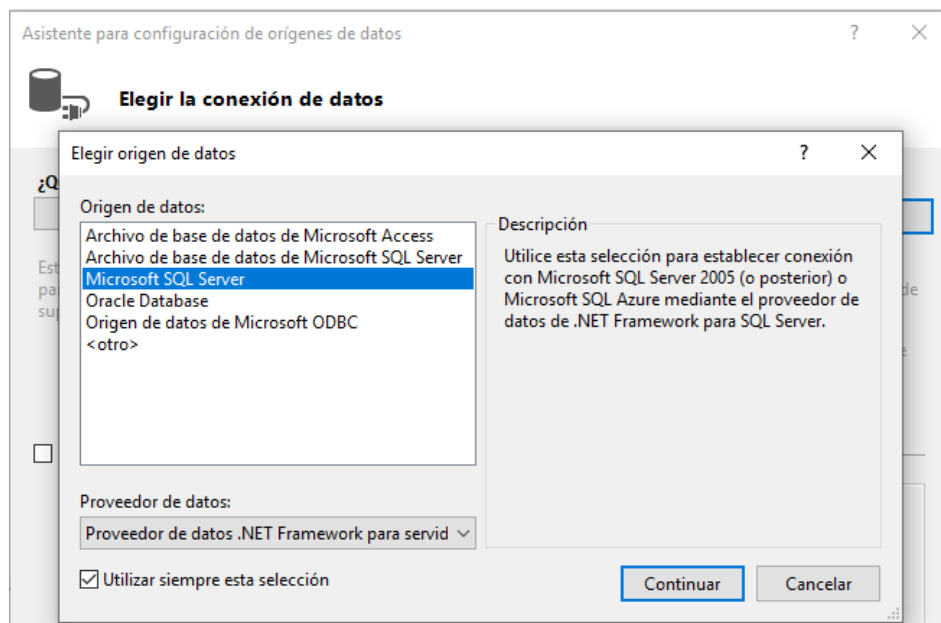
Nos preguntará ahora si queremos conectarnos con un origen de datos:



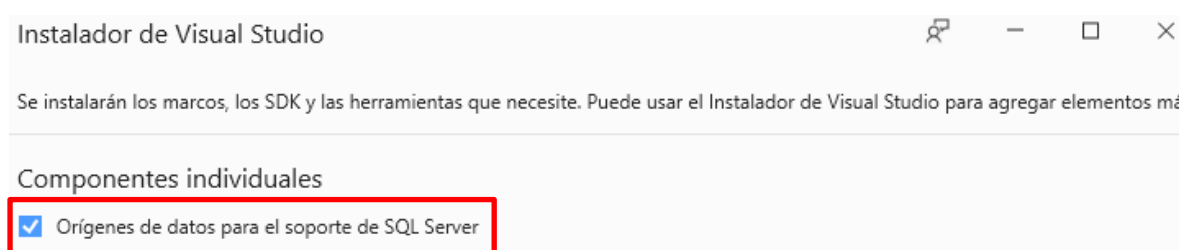
Lo seleccionamos y pulsamos en "Siguiente"



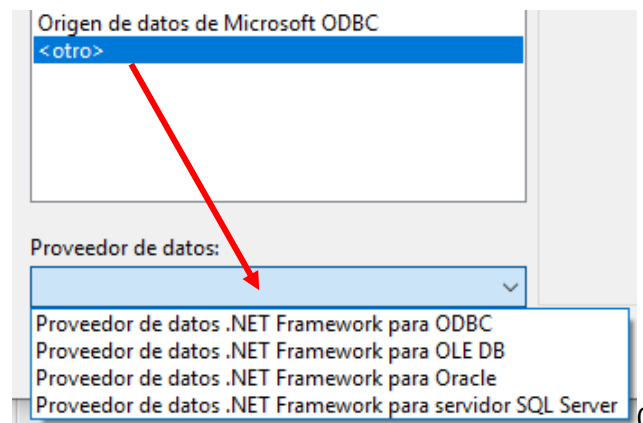
Vamos a crear una nueva conexión de datos, así que pulsamos en el botón de "Nueva conexión":



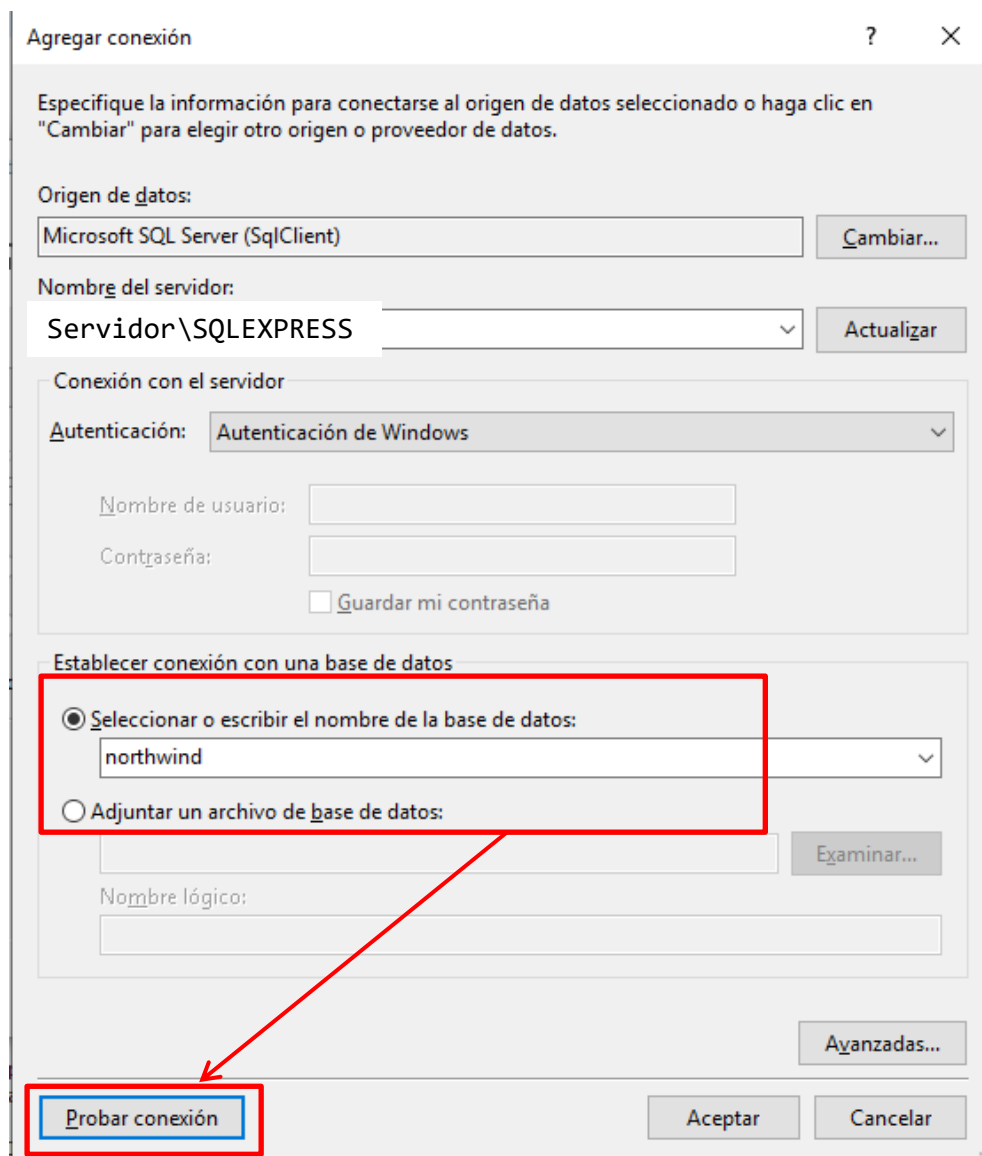
Al realizar esta acción nos indicará que necesitamos instalar el siguiente paquete para trabajar con SQL Server: "Orígenes de datos para el soporte de SQL Server"



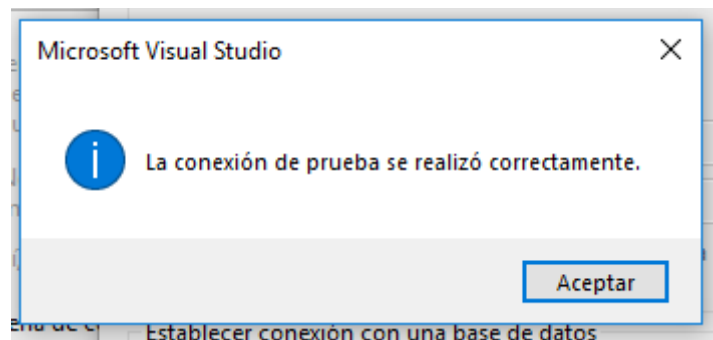
Seleccionamos el origen de datos, que va a ser "Microsoft SQL Server". Si pulsamos "Otro" podemos ver los otros proveedores:



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

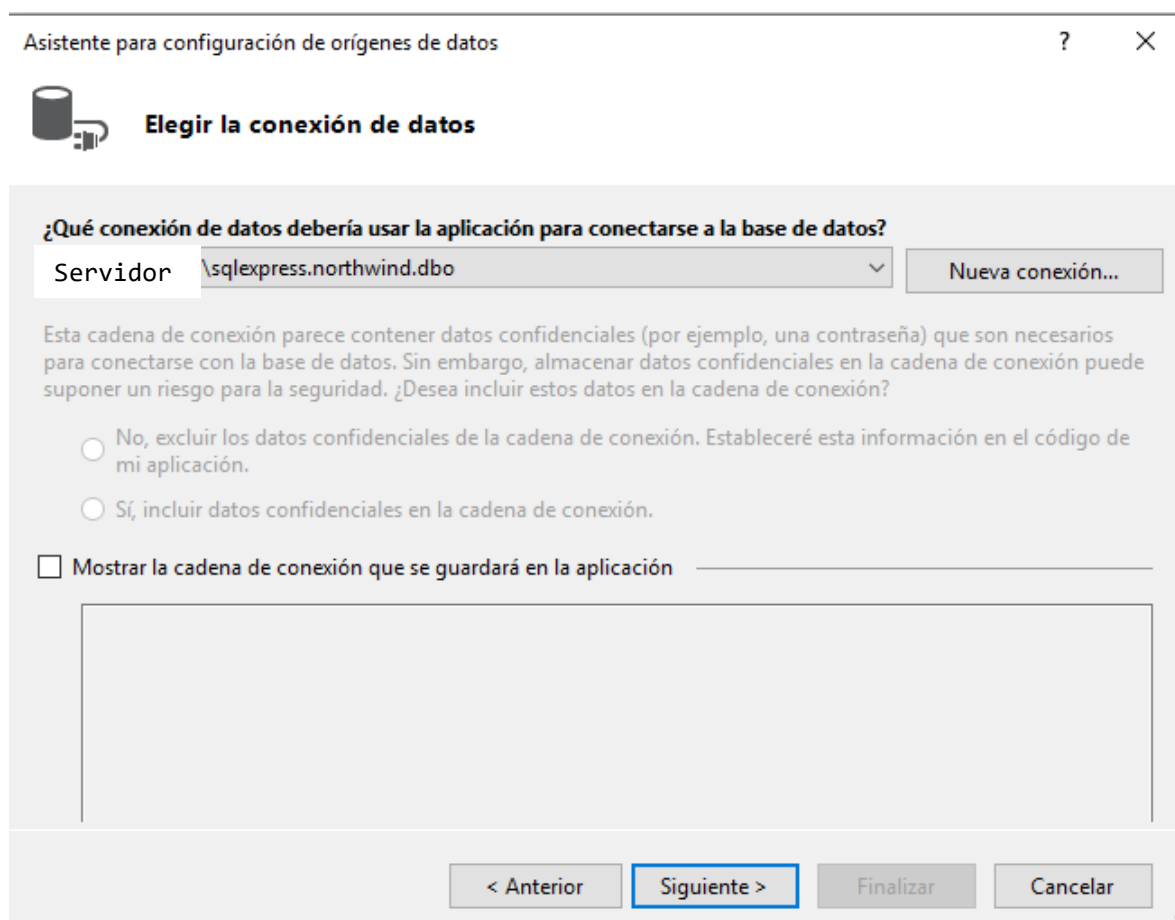


Pulsamos en "Probar Conexión" para que nos indique si el proceso ha ido bien:




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.

Asistente para configuración de orígenes de datos

 **Guardar cadena de conexión en el archivo de config. de la aplicación**

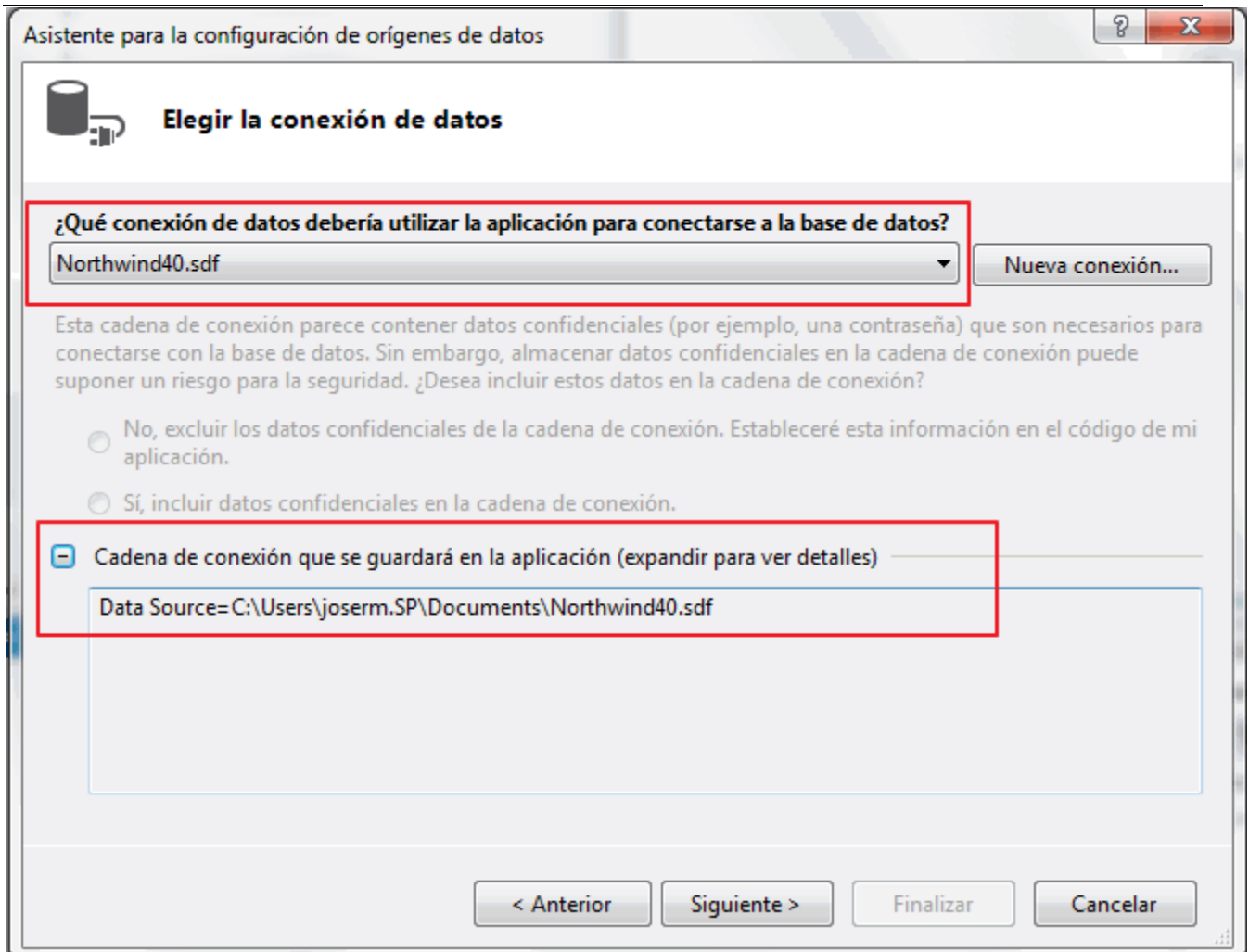
El almacenamiento de las cadenas de conexión del archivo de configuración de aplicación facilita el mantenimiento y la implementación. Para guardar la cadena de conexión en el archivo de configuración de la aplicación, escriba un nombre en el cuadro y, a continuación, haga clic en Siguiente.

¿Desea guardar la cadena de conexión en el archivo de configuración de la aplicación?

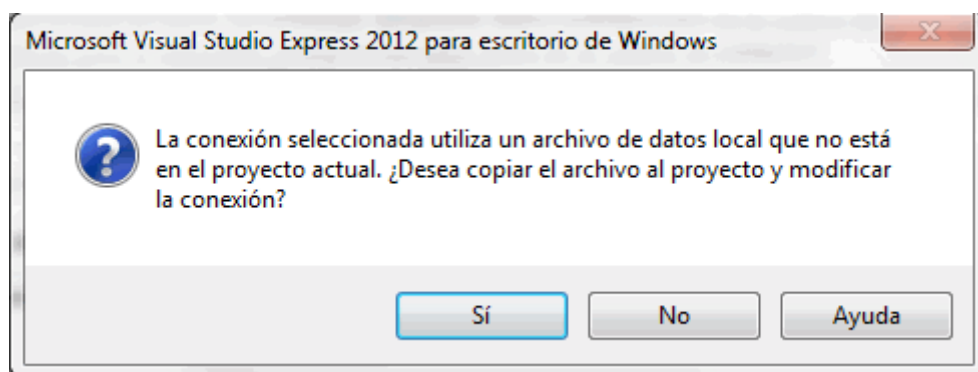
☒ Sí, guardar la conexión como:

< Anterior **Siguiente >** Finalizar Cancelar

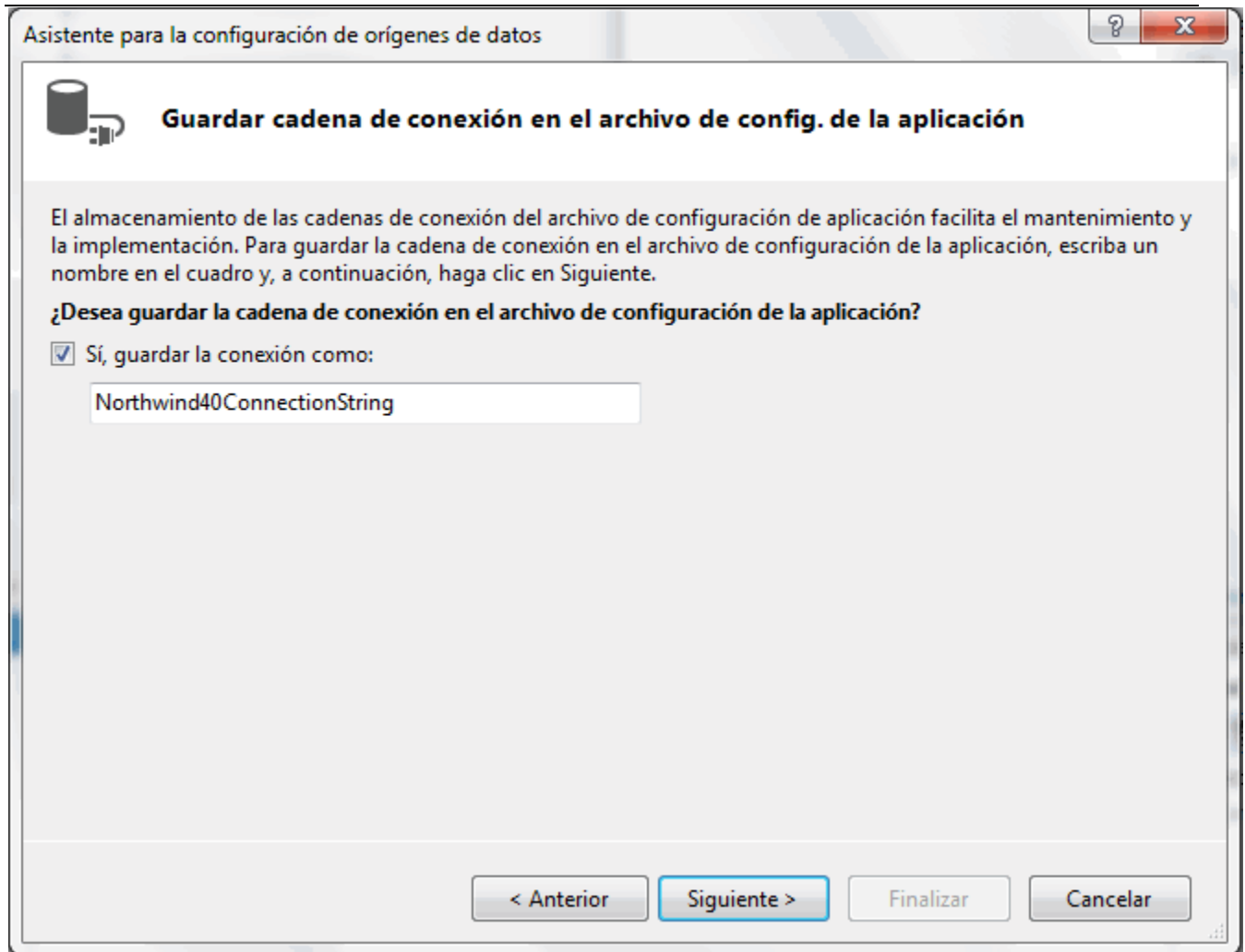
Si el tipo de conexión es otro llegaremos a una pantalla similar a ésta:



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":

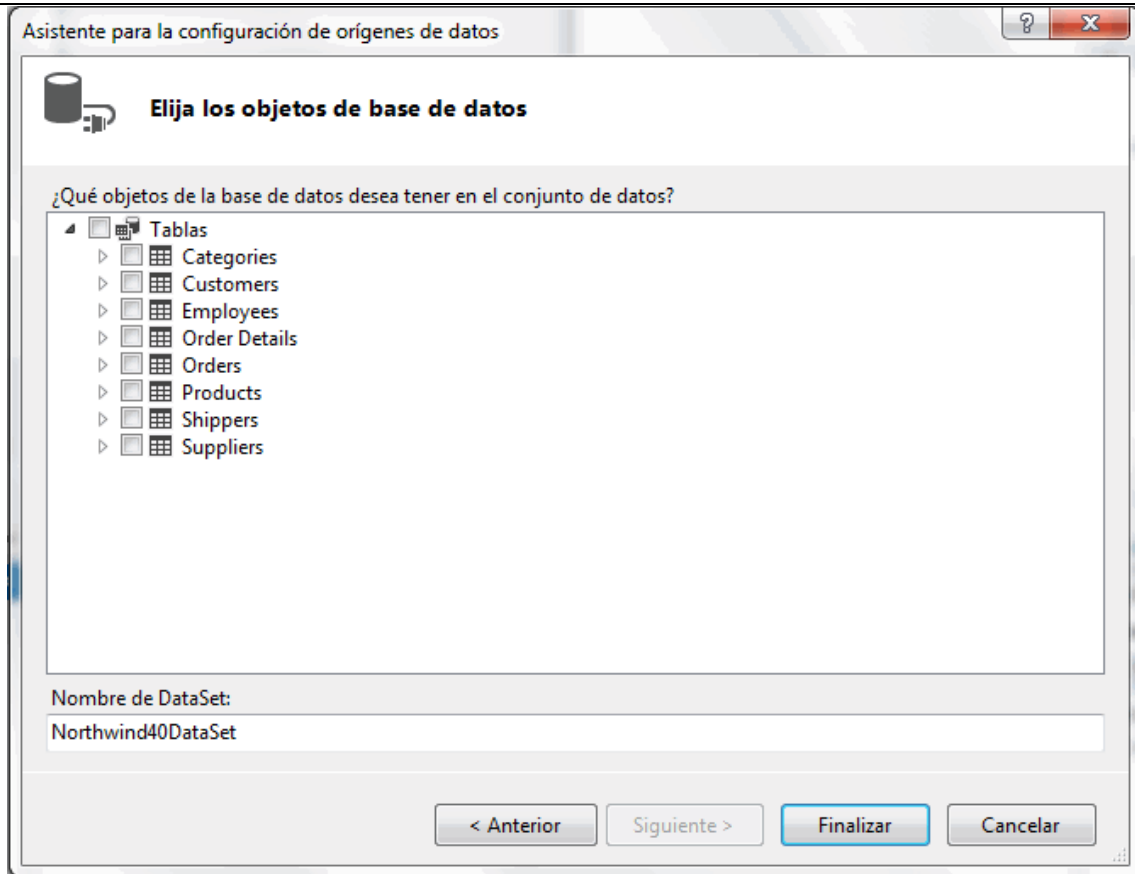


Y nos pregunta si queremos anexar esta base de datos a nuestro proyecto, le diremos que sí:

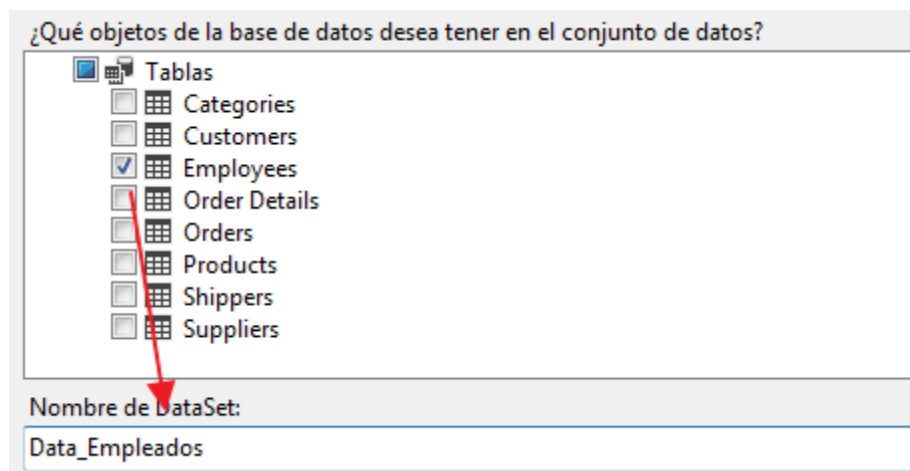


Le asociaremos un nombre más legible para hacer referencia en el código que hagamos a él en lugar de la cadena anterior. El IDE nos va a guardar en la configuración de nuestro proyecto un enlace a esta base de datos, llamando a este enlace con el nombre que le hayamos puesto en esa pantalla.

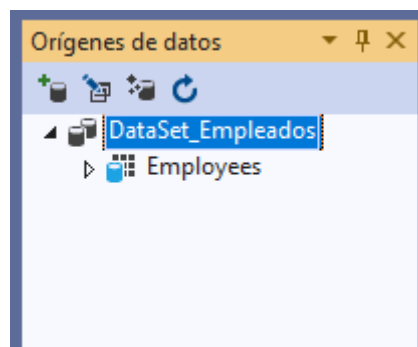
Pulsamos siguiente para ver ya el contenido de esa conexión de base de datos:



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



Pulsamos en Finalizar y ya tenemos todo listo:

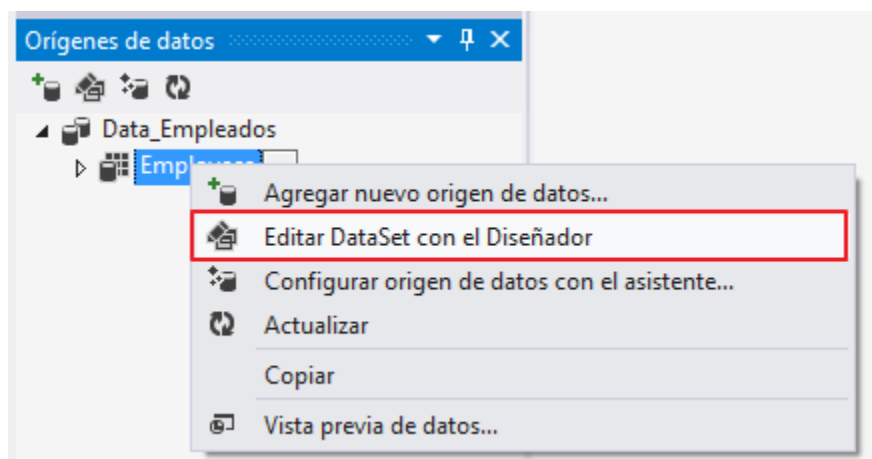


Recapitulemos:

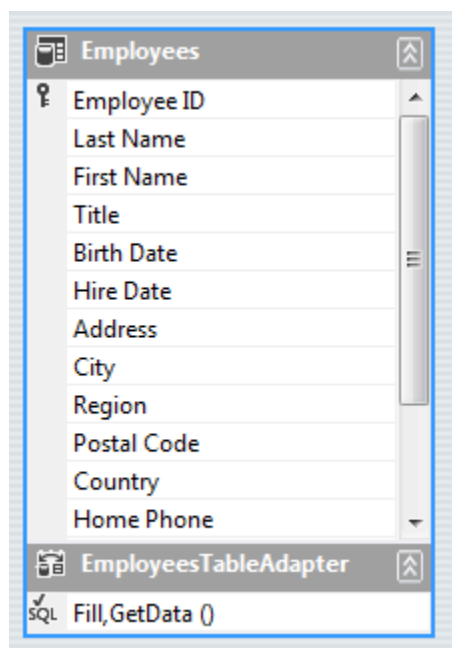
1. Hemos creado una conexión de una base de datos de tipo SQL Server.
2. Hemos vinculado con "northwind".
3. Ha creado una cadena de conexión para apuntar a esta definición.
4. Hemos abierto la base de datos y de todas las tablas hemos seleccionado "Employees"
5. Finalmente hemos asociado el nombre "DataSet_Empleados"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

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

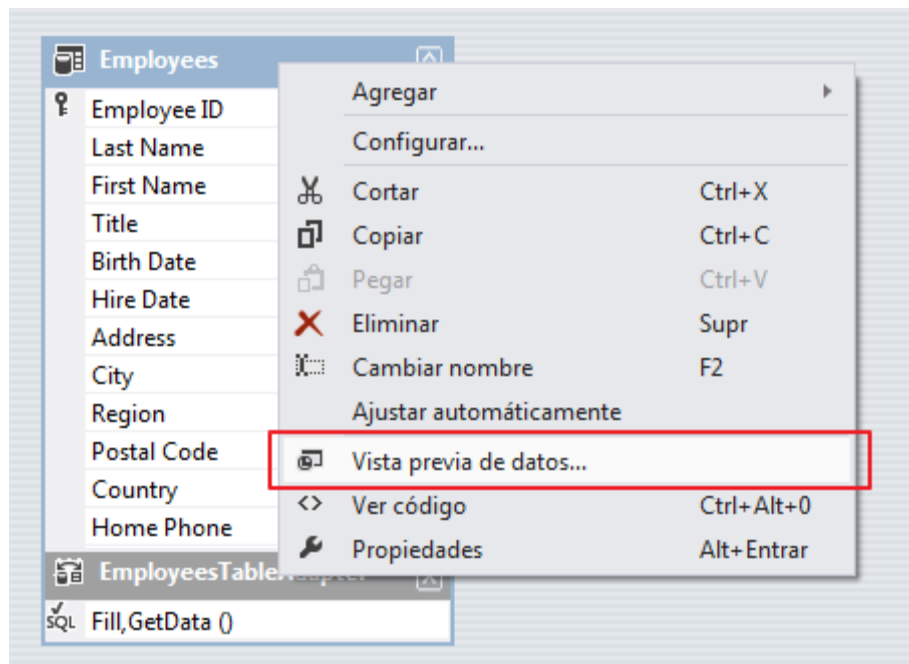
Sigamos viendo más detalles de esta pantalla. Pulsa en el nombre de la vista que hemos creado y seleccionamos:



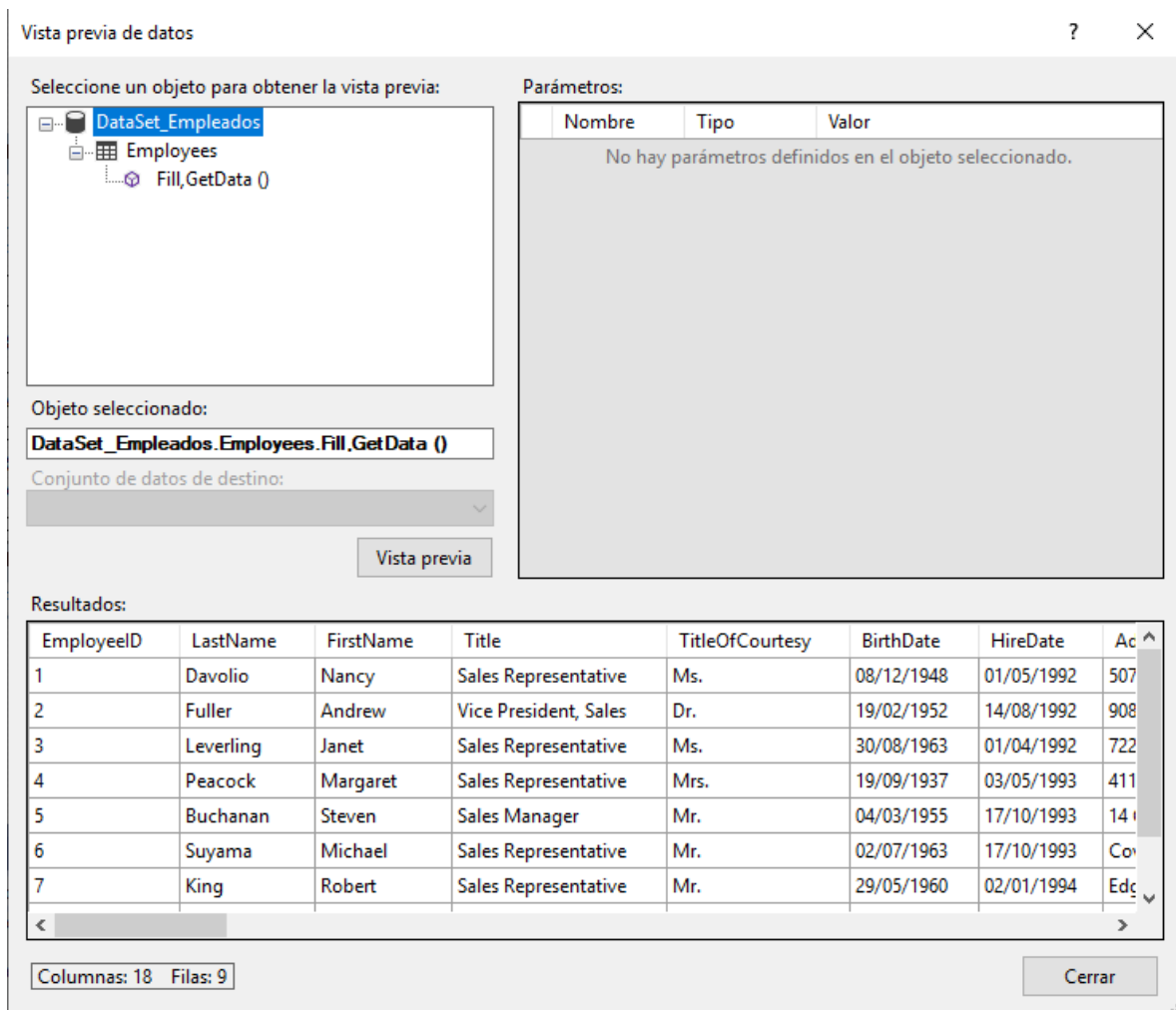
Podremos ver la tabla que hemos añadido a nuestro conjunto de datos, si hubiéramos seleccionado más estarían aquí:



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

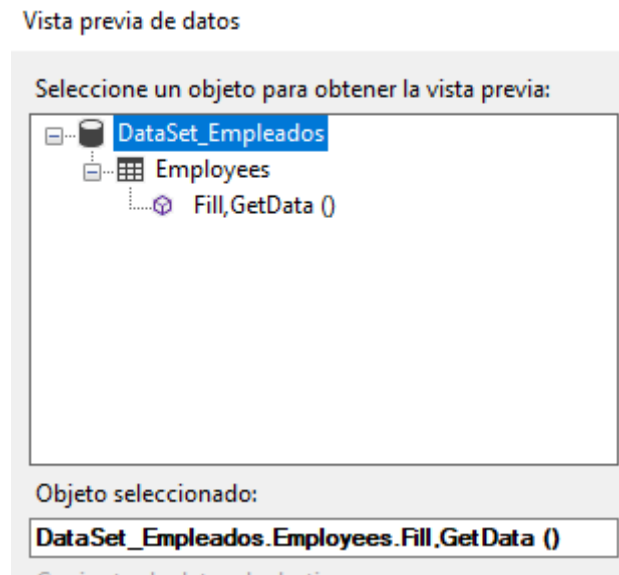


Nos mostrará los datos que contiene esta tabla:



Nos ha mostrado el contenido de la tabla.

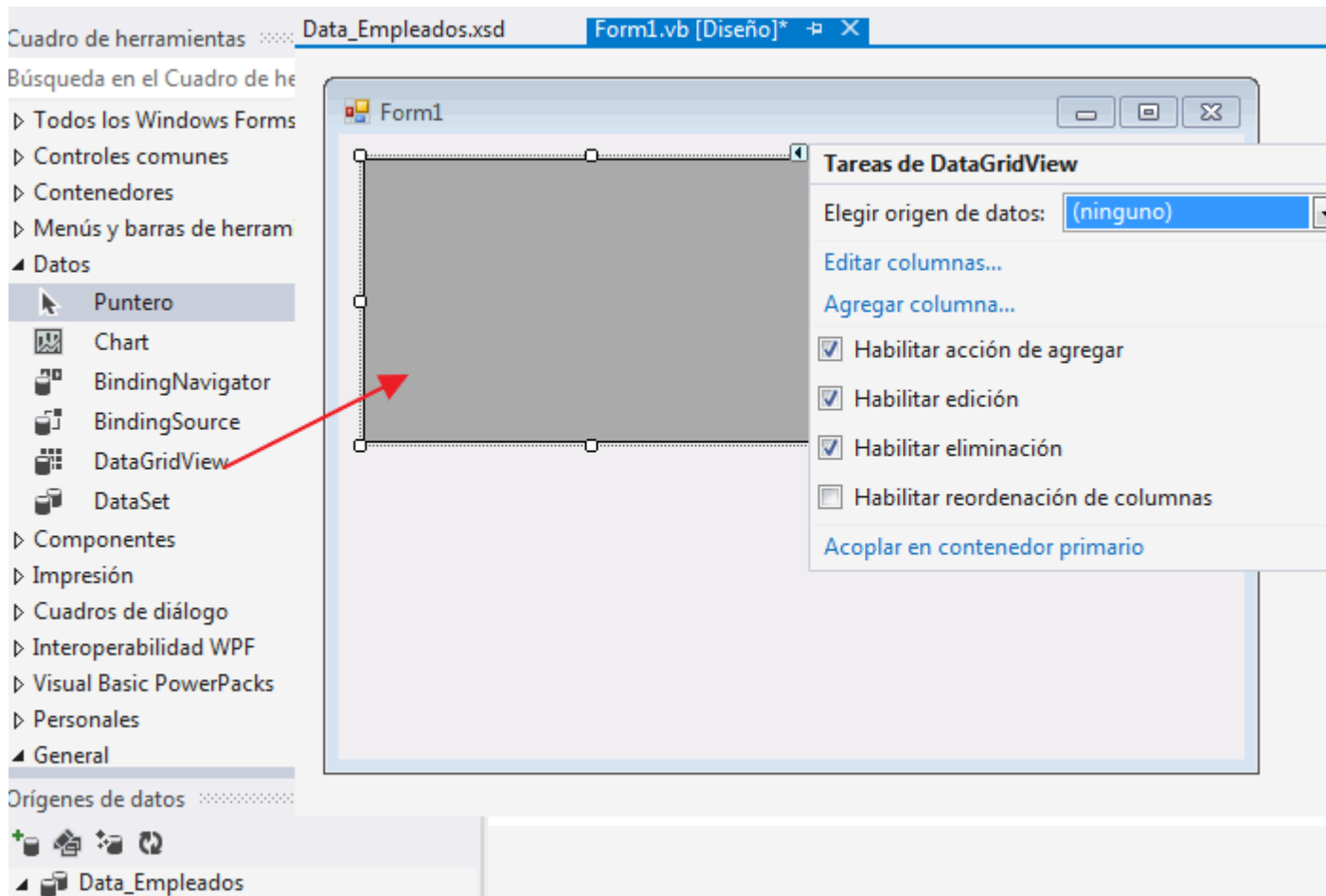
Ahora nos fijamos en el cuadro de arriba a la izquierda:



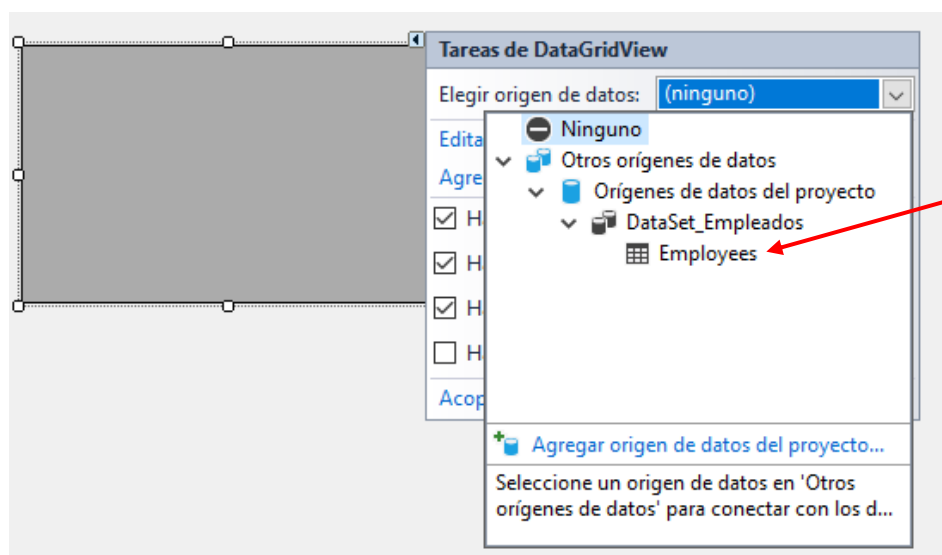
Nos muestra la forma con que ha recuperado los datos que aprenderemos luego. Parece que ese "DataSet" que hemos llamado "DataSet_Empleados" tiene dentro una tabla "Employees" que ejecutando el método "Fill, GetData" nos ha permitido recuperar los datos. Bien, poco a poco: cadena de conexión, conexión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

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

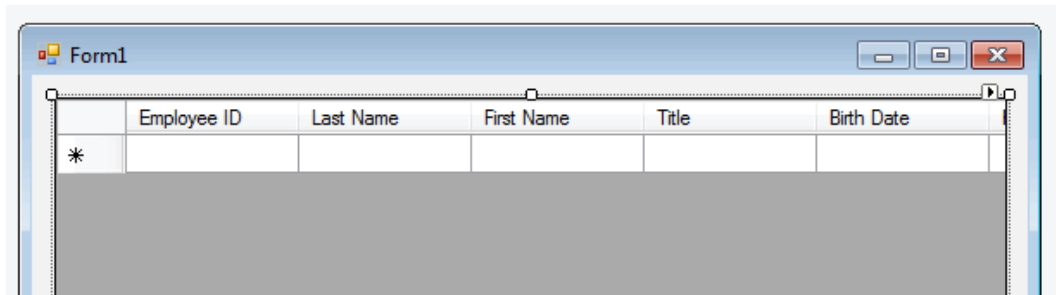
Volvamos a nuestro programa y vamos a añadir un control del tipo "DataGridView" situado en la sección de control de acceso a datos, que tiene estas opciones en su menú de propiedades más comunes:



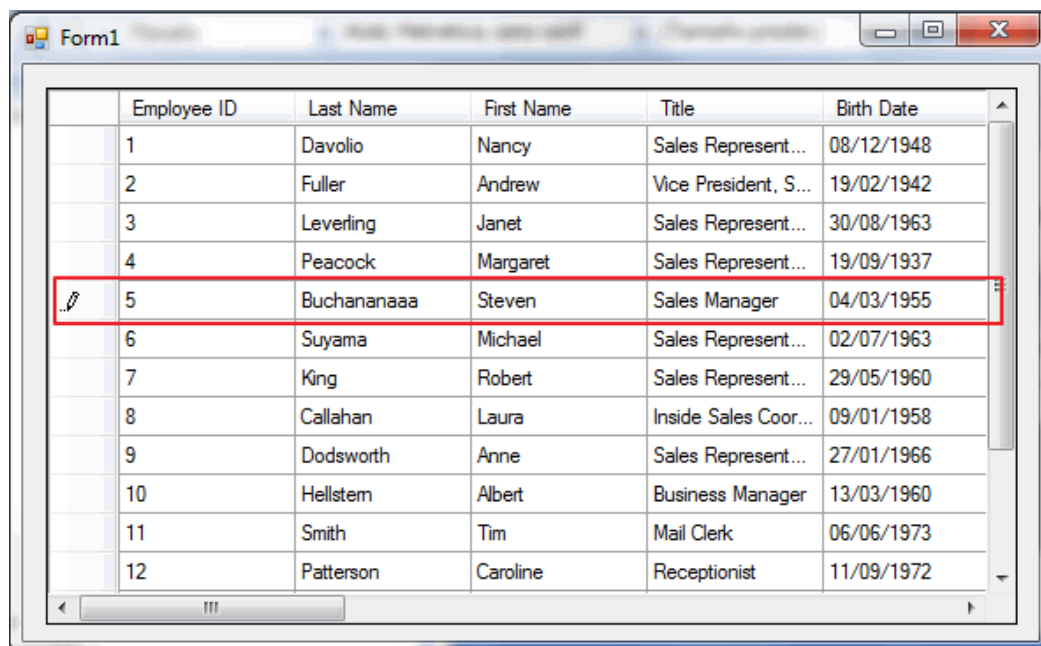
Pulsamos en el origen de datos y seleccionamos:



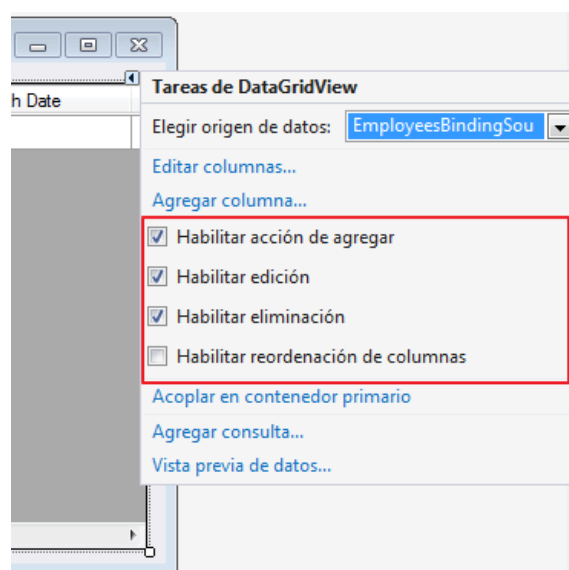
Veremos cómo cambia el aspecto de la cuadrícula colocándose todas las columnas de la tabla de la base de datos que hemos seleccionado:



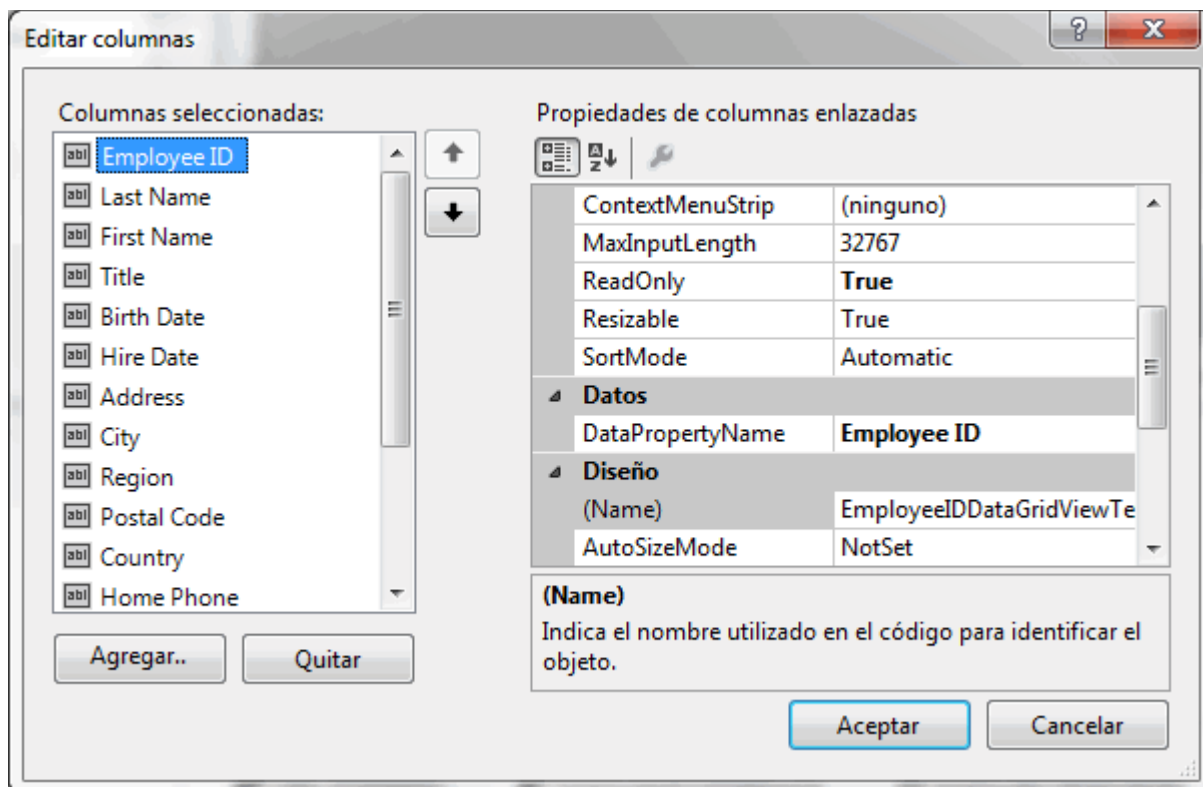
Vamos a ejecutar el programa:



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...

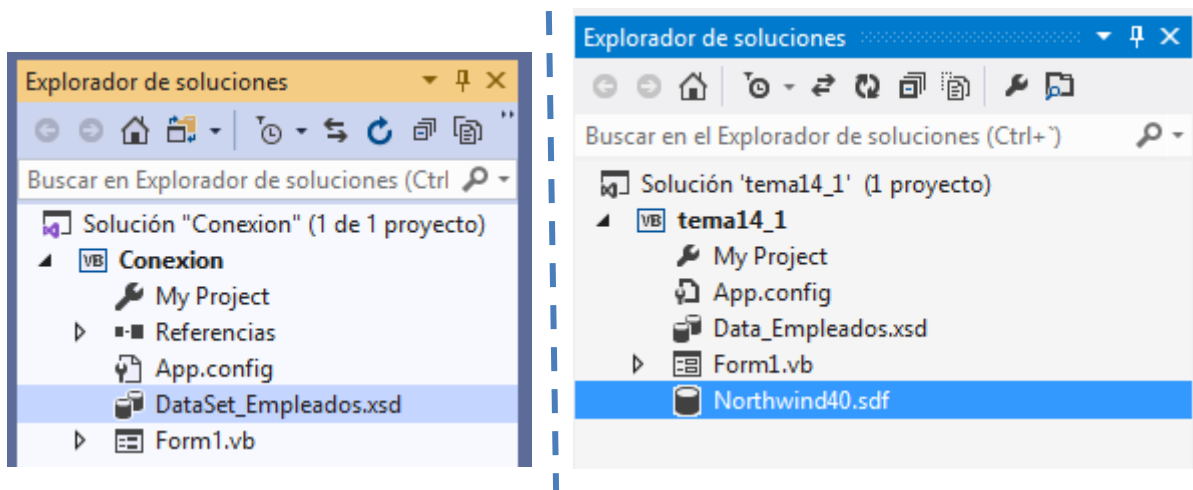
Para terminar, veamos el único código que nos ha puesto para hacer esta carga al iniciarse el programa:

```
Public Class Form1
    0 referencias
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'TODO: esta línea de código carga datos en la tabla 'DataSet_Empleados.Employees'
        Me.EmployeesTableAdapter.Fill(Me.DataSet_Empleados.Employees)
    End Sub
End Class
```

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

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 DataSet_Empleados.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":

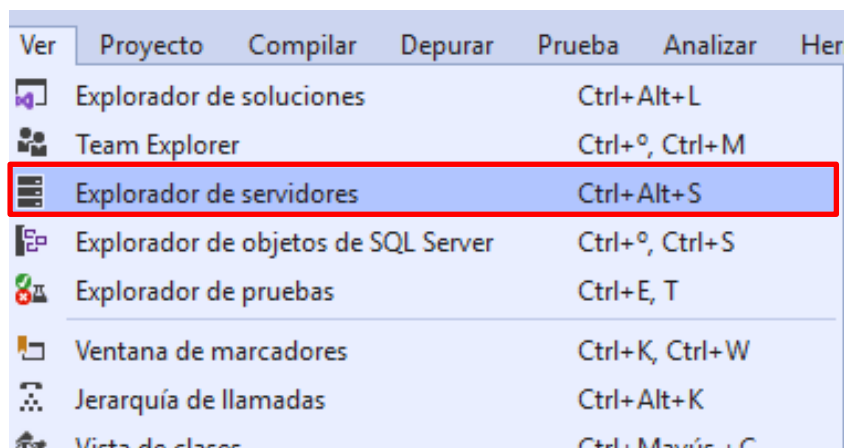
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="Conexion.My.MySettings.northwindConnectionString"
        connectionString="Data Source=SERVIDOR\SQLEXPRESS;Initial Catalog=northwind;Integrated Security=True"
        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.

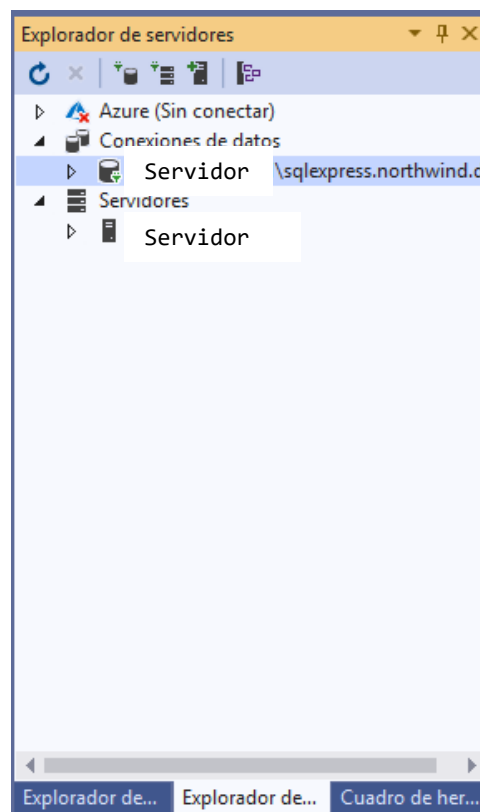
4. SQL y las tablas en las bases de datos

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.

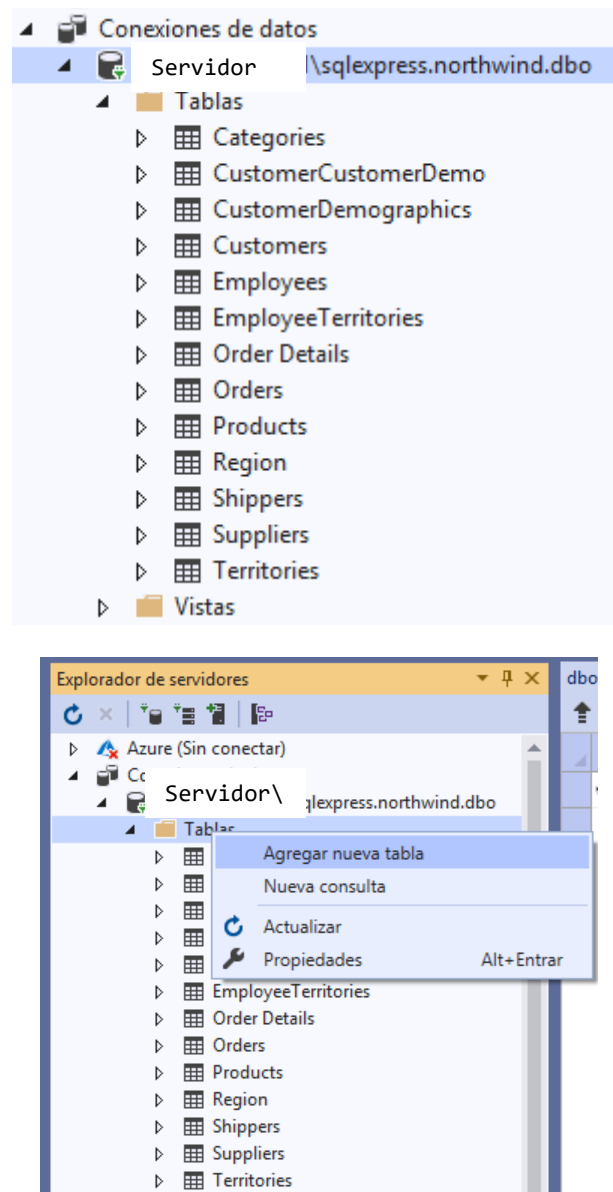
Vamos a activar ahora una vista de las conexiones a las bases de datos, seleccionaremos Ver en el menú y luego:



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:
 - tinyint: 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:

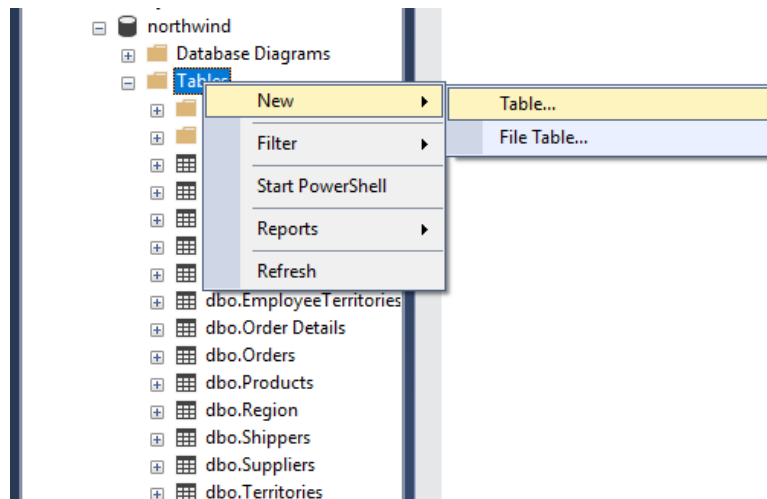
Tipo de datos	Detalles	Espacio
bigint	enteros largos -2 ⁶³ hasta 2 ⁶³	8 bytes
binary	binario fijo hasta 8000	n bytes
bit	0, 1 ó null	1 byte para 8 campos de bit
char (n)	texto fijo hasta 8.000	n bytes

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 Identifiers	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

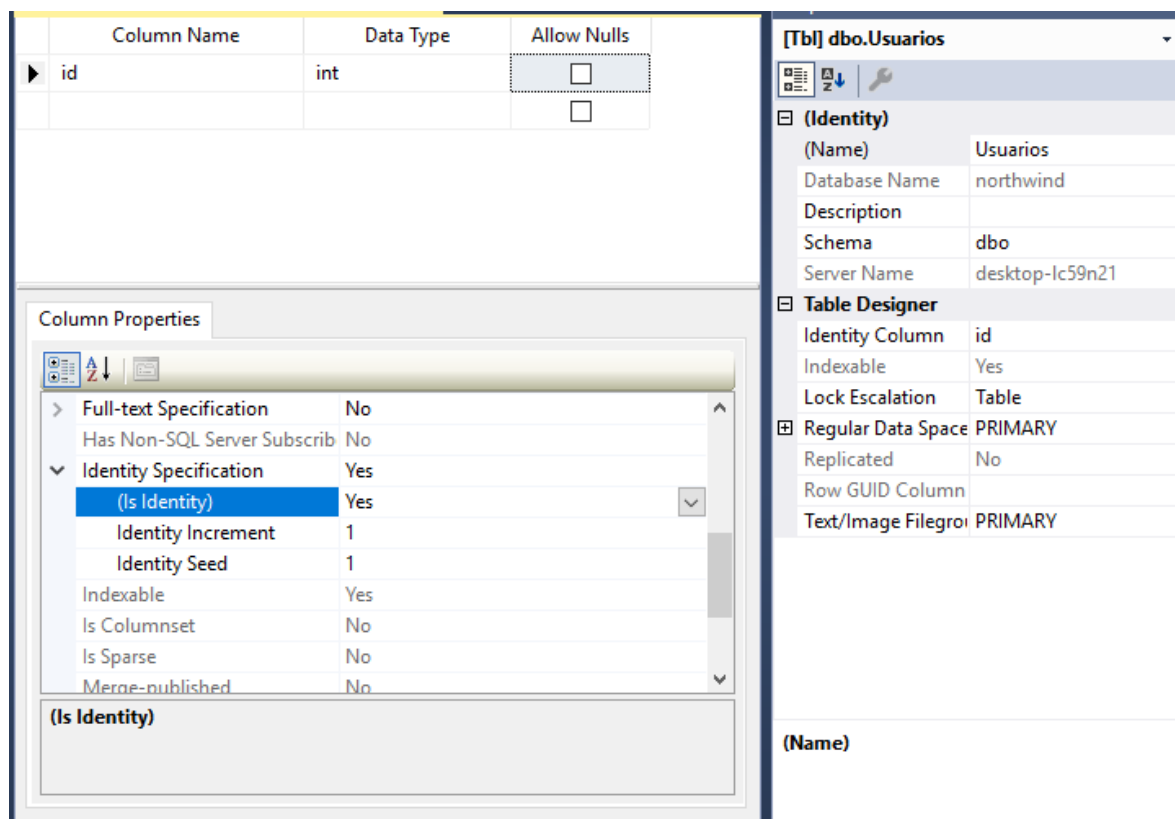
Si continuamos en este entorno para crear las tablas necesitamos conocer más sobre T-SQL.

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**.

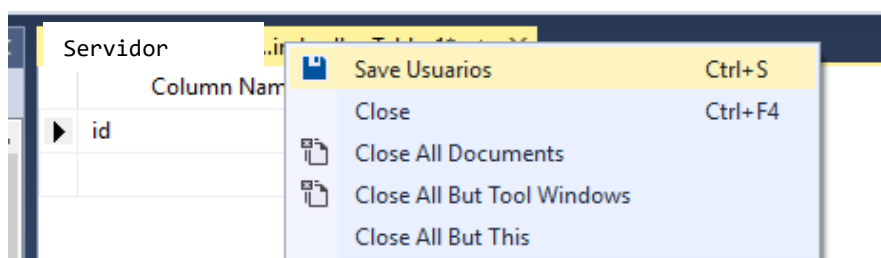
Creamos una nueva eligiendo la opción:

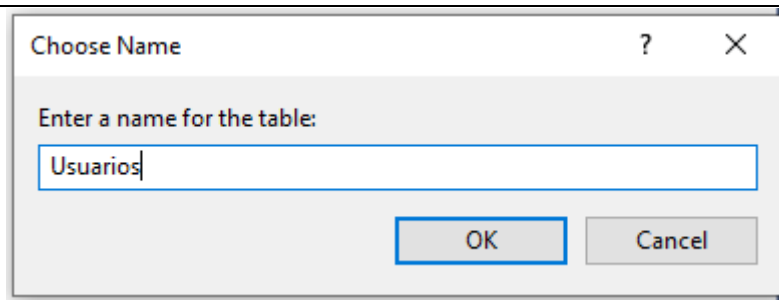


Vamos a introducir un campo como el siguiente:



Le ponemos nombre a la tabla así:





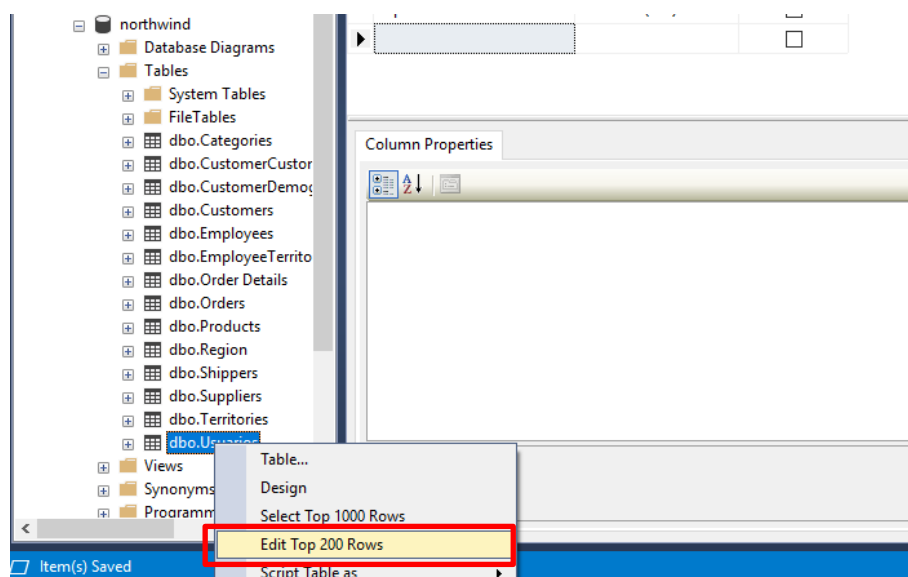
Primero hemos puesto arriba un nombre a esta tabla, "Usuarios", y le hemos añadido un campo que se llama "id" de tipo entero "Int" y luego he marcado debajo que es de tipo "Identity". Esto significa que va a ser un valor autonumérico. Este tipo de característica la "identity" hace que ese campo se rellene 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:

Usuarios: consulta..._1\Northwind40.sdf) Form1.vb			
	id	Nombre	Apellidos
	1	Jose	Rodriguez
	2	Ana	Lopez
	3	Jesus	Martinez
►*	NULL	NULL	NULL

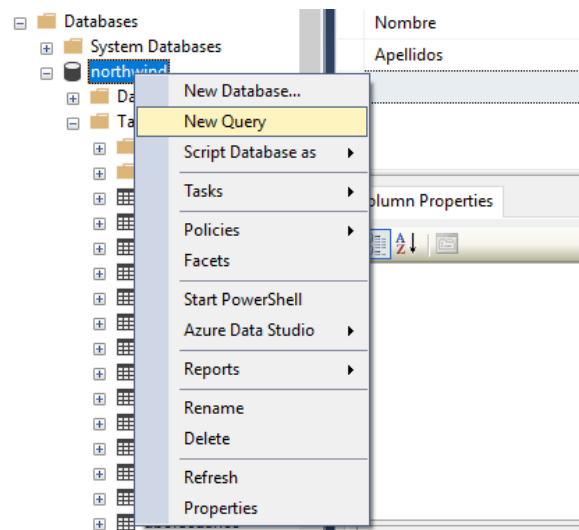
Hemos podido comprobar que los datos inicialmente son "Null" y que al ir introduciendo datos en "Nombre" y "Apellidos" y pulsar el cursor hacia abajo, se rellenaba automáticamente la columna de "Id". Perfecto, eso es lo que queríamos, un campo que se rellene solo con un valor único para así poder definir 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":

Escala	
GUID de fila	False
Identity	True
Incremento de valor de identidad	1
Precisión	
Valor de inicialización de identidad	1
Valor predeterminado	
Incremento de valor de identidad El incremento de cada valor de identidad.	

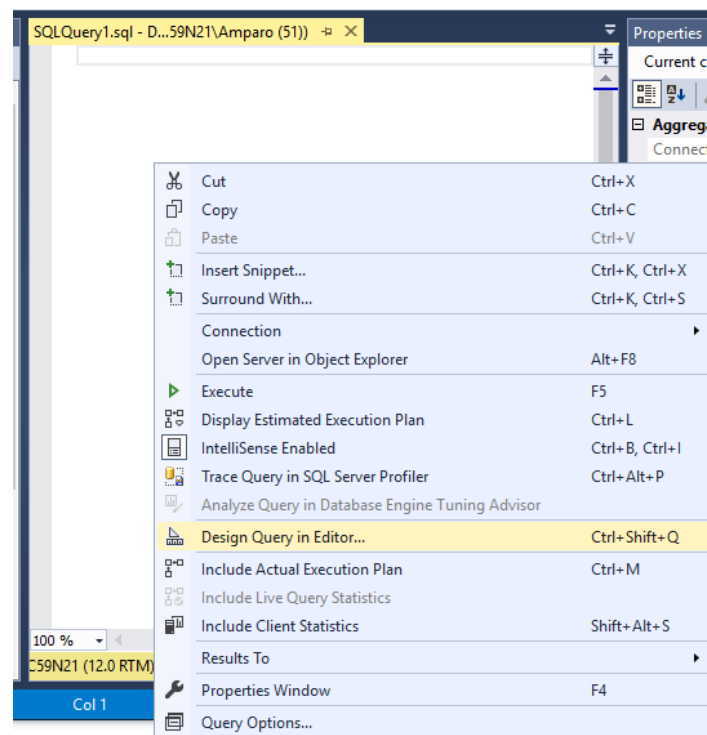
Veamos un rápida explicación de porqué necesitamos una clave única en nuestras tablas... Imaginemos que tenemos dos "fernandez lopez" en nuestra tabla de empleados y ejecutamos una instrucción para cambiar un dato en un "fernandez lopez" que tengo en pantalla. SQL Server o la base de datos no sabrá distinguir 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

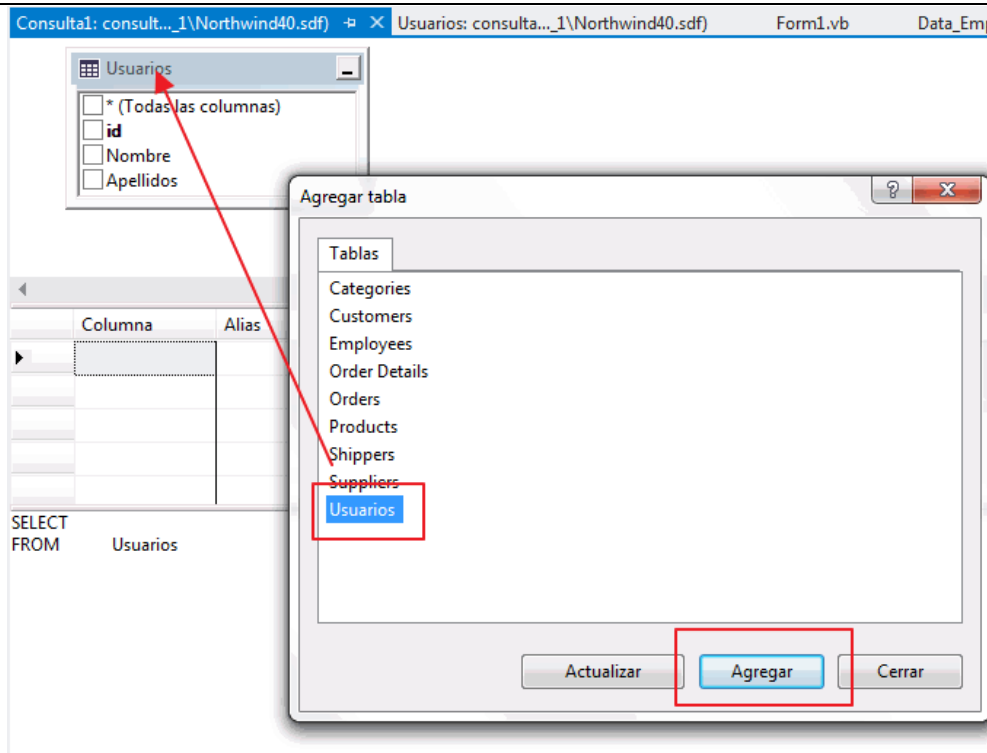
Tenemos más posibilidades para generar consultas con nuestro administrador de conexiones. Seleccionemos la opción:



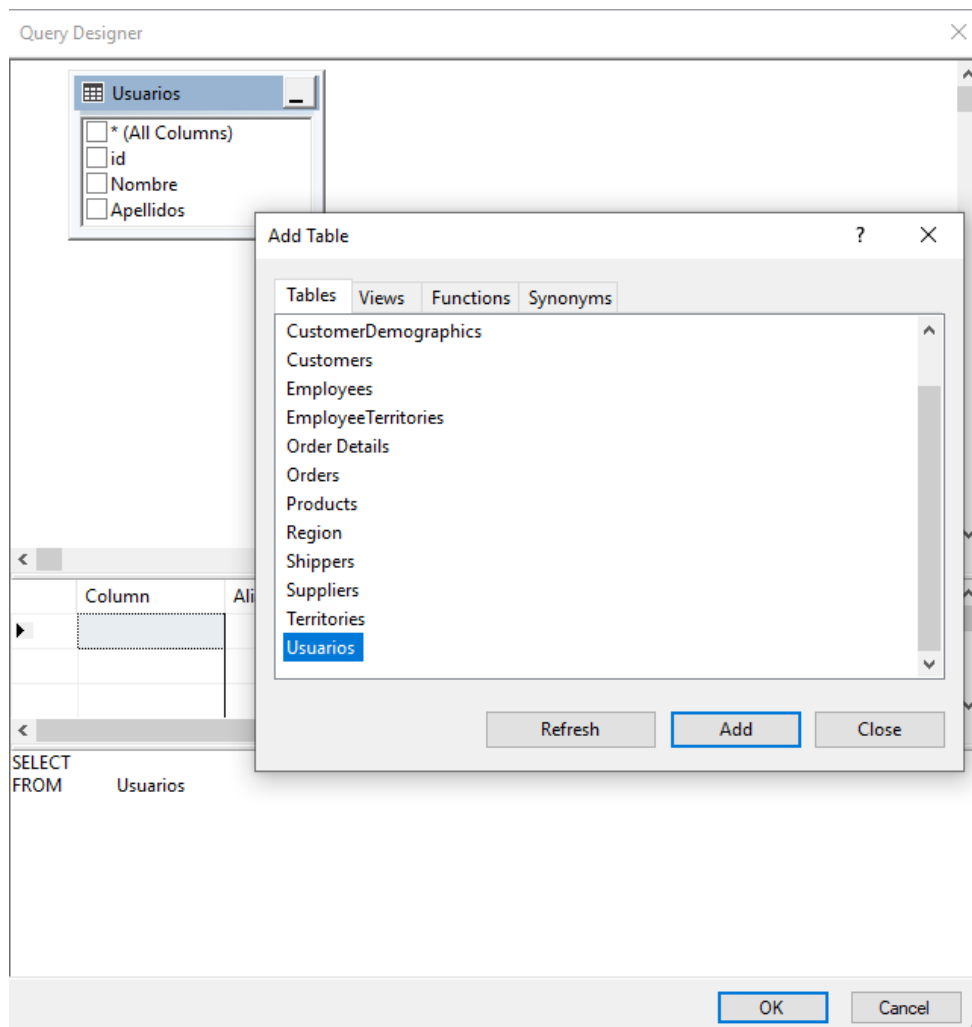
Hacemos botón derecho sobre el espacio en blanco que nos aparece y elegimos “Design Query in Editor”



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:



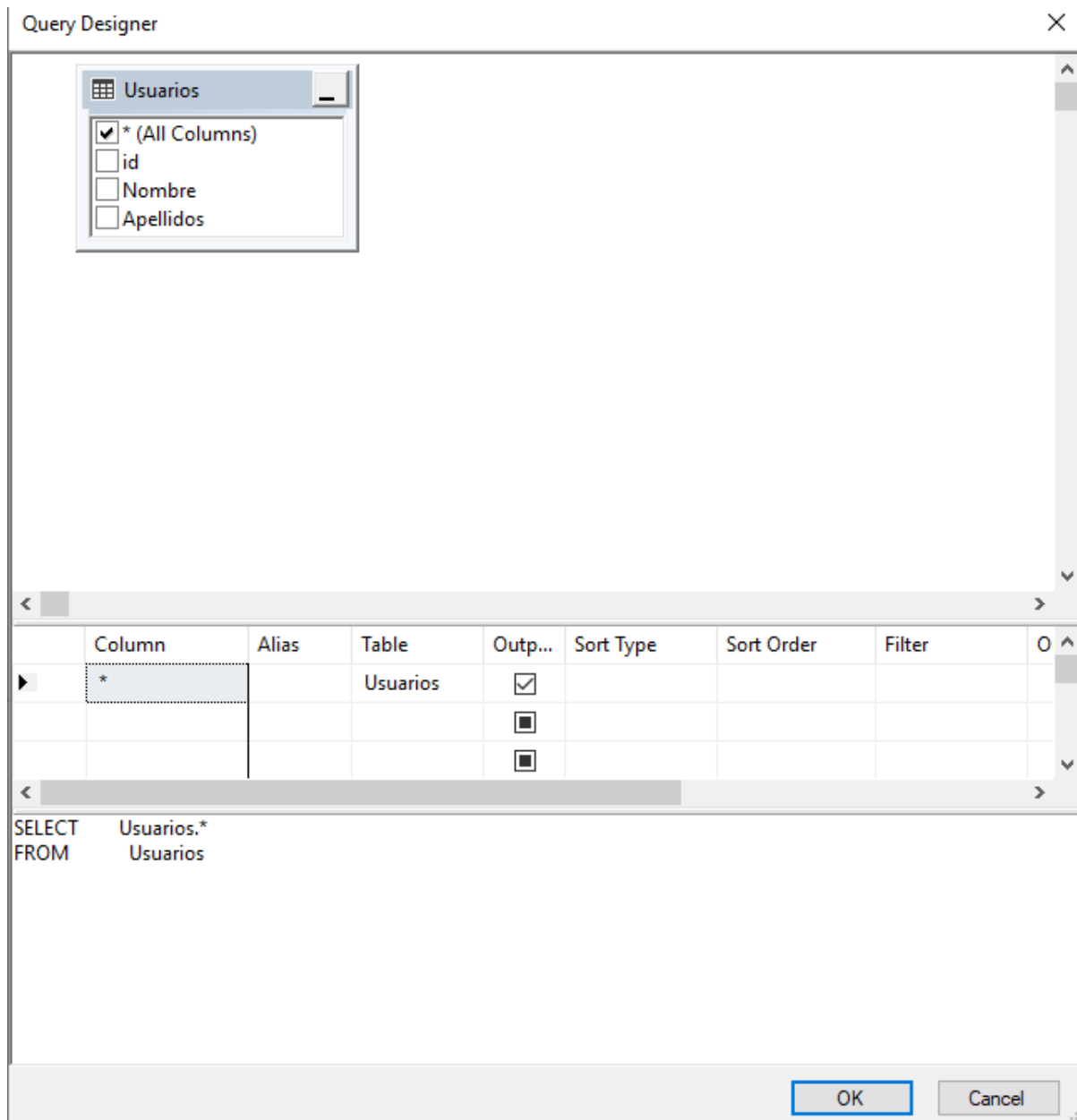
Al pulsar en Agregar aparece ahora en el diseñador.



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

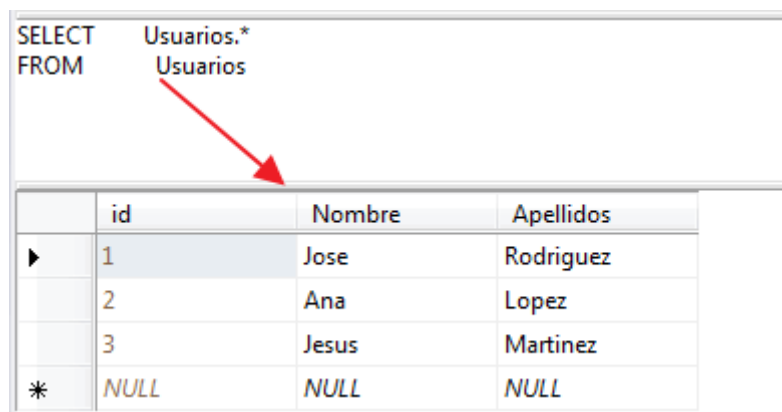
Construimos la consulta seleccionando los campos y las opciones necesarias y pulsamos “Ok”.

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:



	id	Nombre	Apellidos
▶	1	Jose	Rodriguez
	2	Ana	Lopez
	3	Jesus	Martinez
*	NULL	NULL	NULL

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.

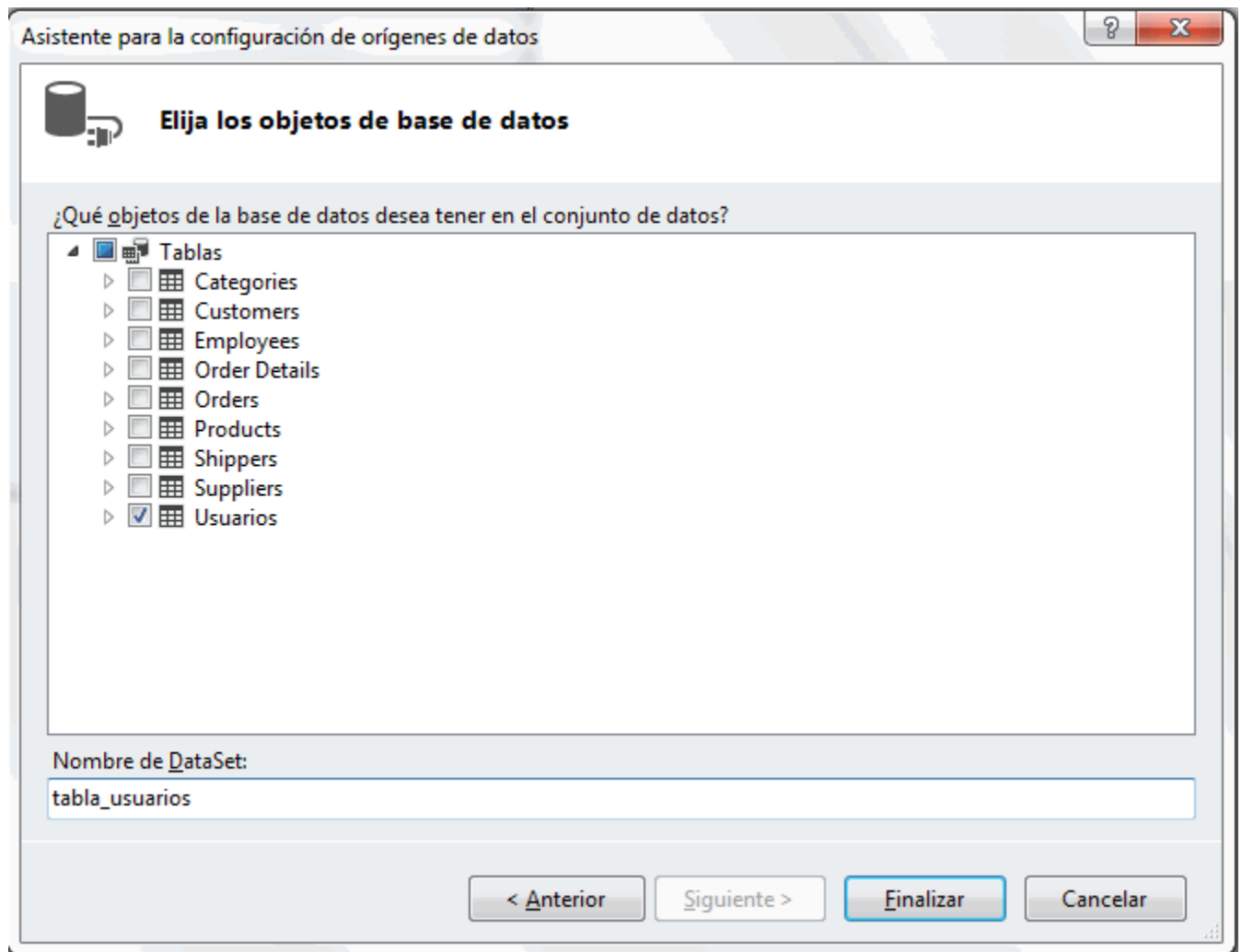
5. Código ADO.NET

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.

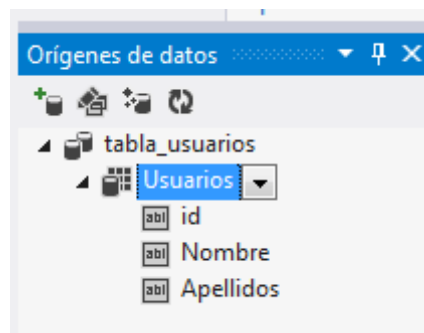
Comenzaremos creando un nuevo proyecto. En este proyecto añadiremos un nuevo origen de datos, tal y como hicimos antes.

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.

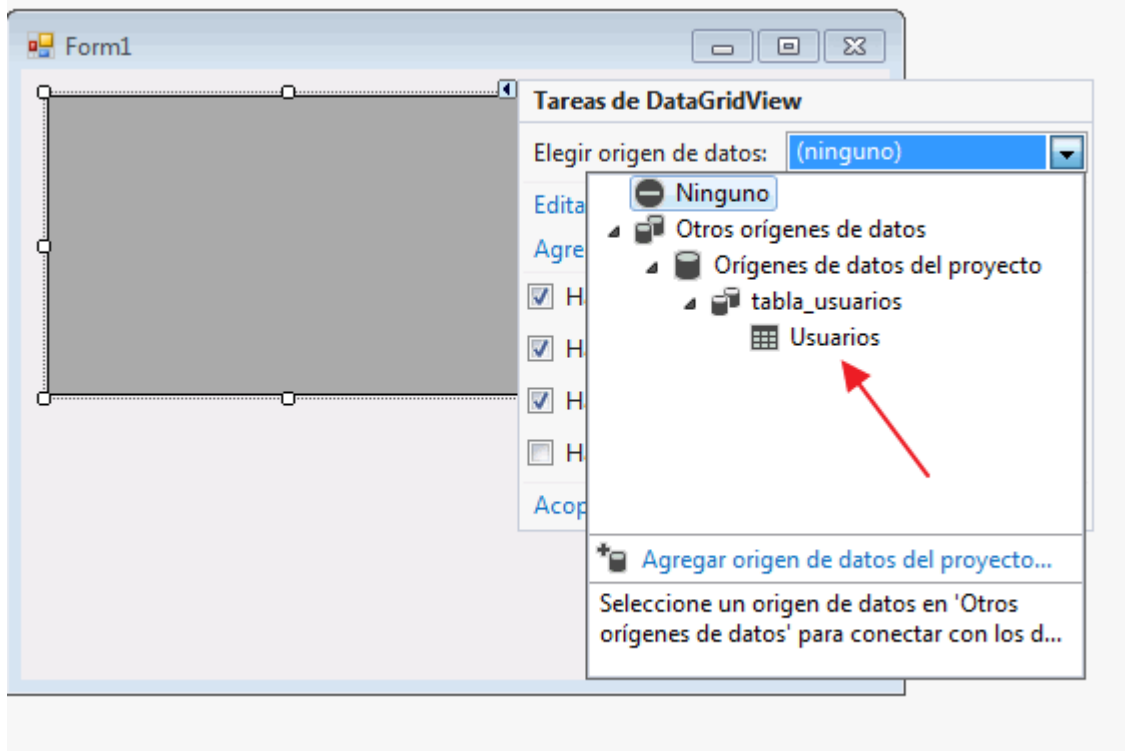
Sigamos con el asistente que nos crea la conexión, seleccionando ahora esta tabla:



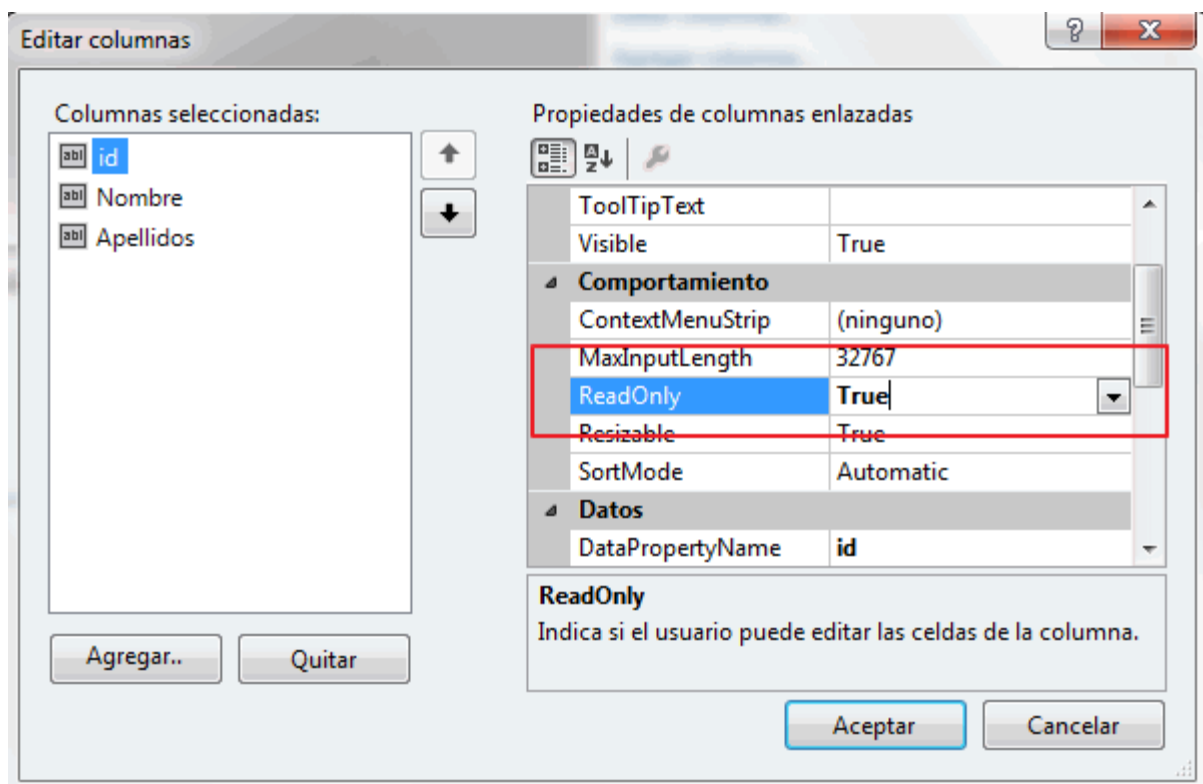
Finalizamos y ya tenemos incorporado a nuestro proyecto una conexión esta tabla:



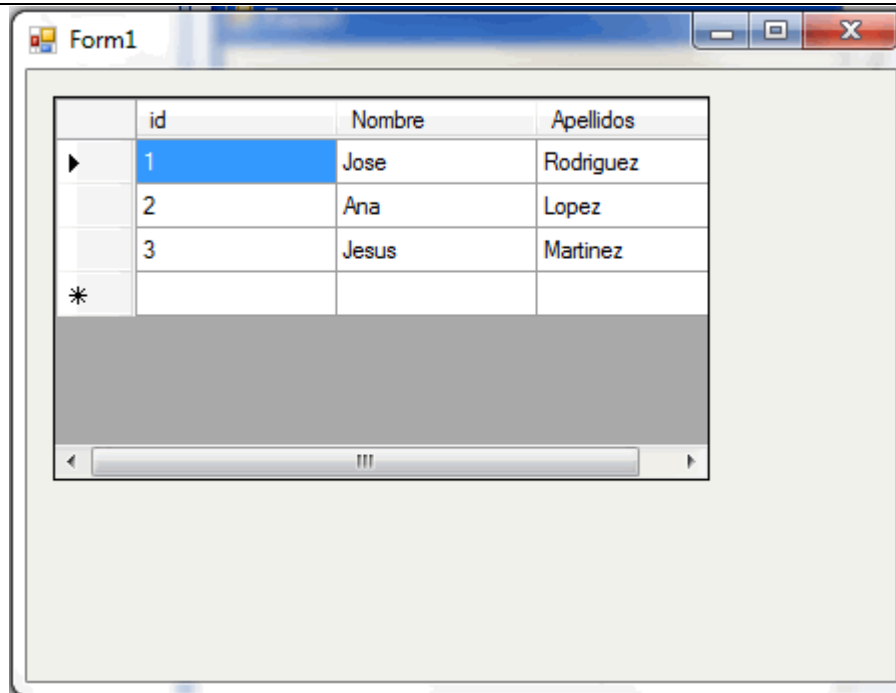
Añadimos un control "DataGridView" y seleccionamos este origen de datos:



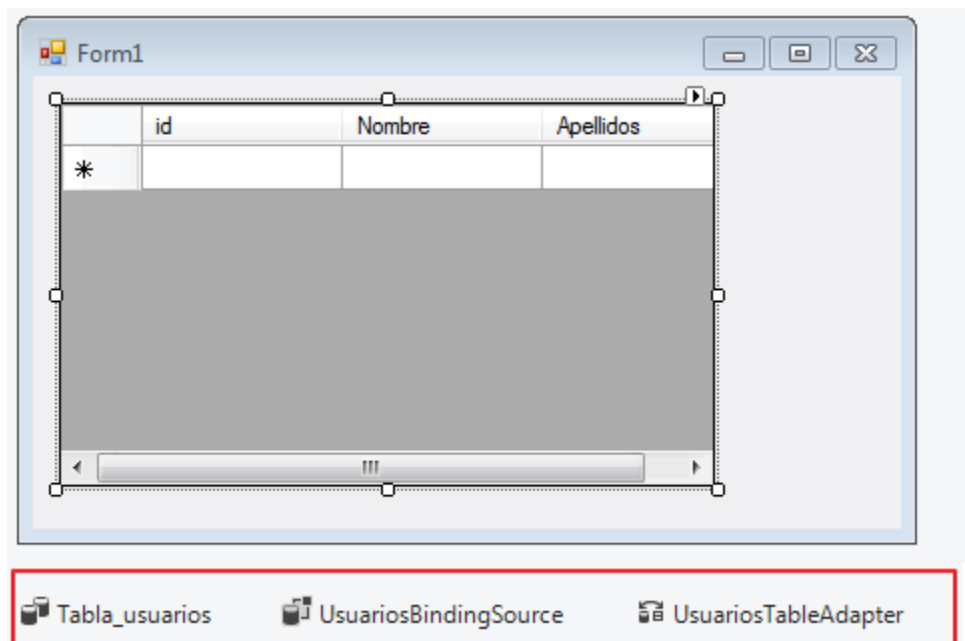
Vinculamos la tabla, le ponemos unos títulos de las columnas y marcamos la del identificador de la fila "id" como de solo lectura ya que no podemos modificarla:



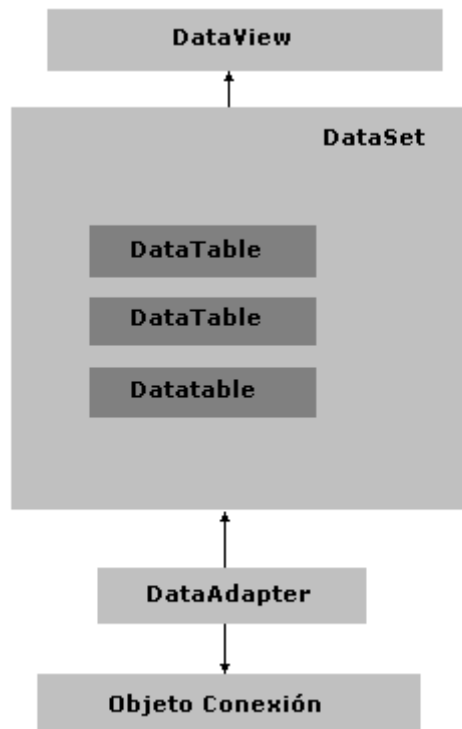
Bueno, no ha hecho falta. Al tratarse de un campo especial que se incrementa automáticamente no debe ser editable y nos lo marca ya como de solo lectura. Ejecutemos el programa para comprobar que funciona:



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

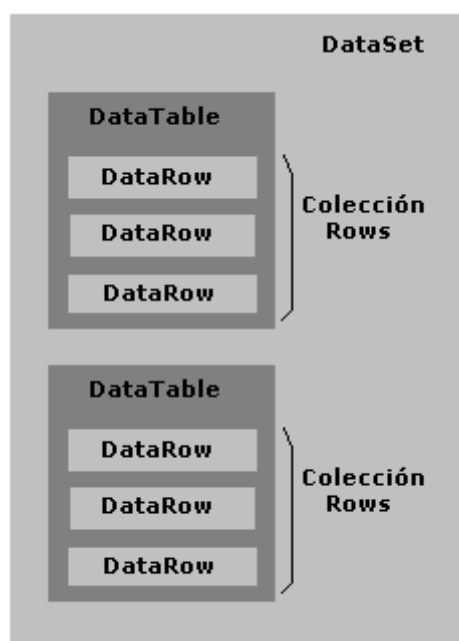


Tenemos varios objetos que iremos viendo a largo de este tema dedicado a las bases de datos. Veamos este esquema de objetos:



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".

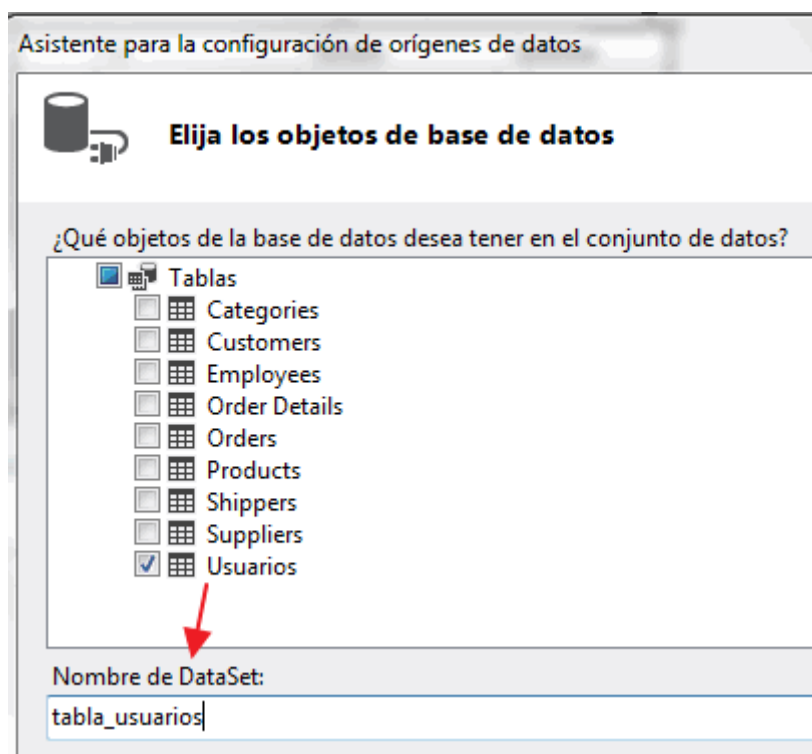
Por tanto, ese adaptador o "DataAdapter" es el que nos comunica con los datos. El otro objeto, el "Dataset" es el contenedor de los datos. Es decir, donde están nuestras consultas, como en el ejemplo que hemos hecho. Estos DataSets son muy versátiles, veamos este esquema:



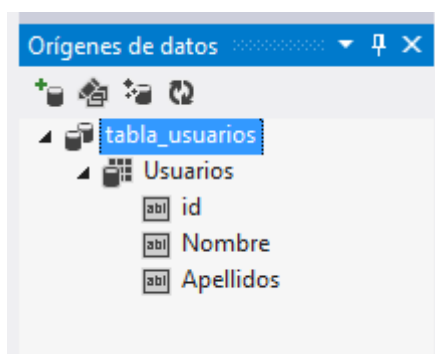
Se compone de una colección de tablas "DataTable" que a su vez tienen una colección de filas. En nuestro ejemplo el Dataset tiene una sola tabla "DataTable" y con una colección de filas "DataRow" que son los registros que hemos metido antes. De momento nos quedaremos con los nombres de los objetos, sigamos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



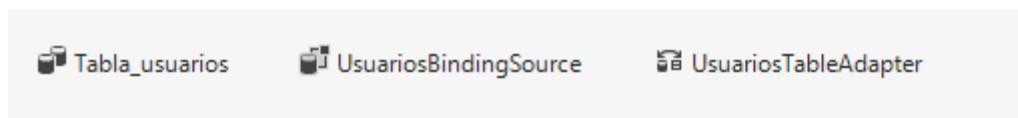
Así vamos aprendiendo los iconos. Jerárquicamente aparece primero el DataSet y posteriormente la tabla con los datos. Por tanto: "tabla_usuarios" es el "Dataset" , "Usuarios" es un "DataTable" contenido dentro del Dataset y las filas de datos son una colección de "DataRows" de esta tabla. Esto se va 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:

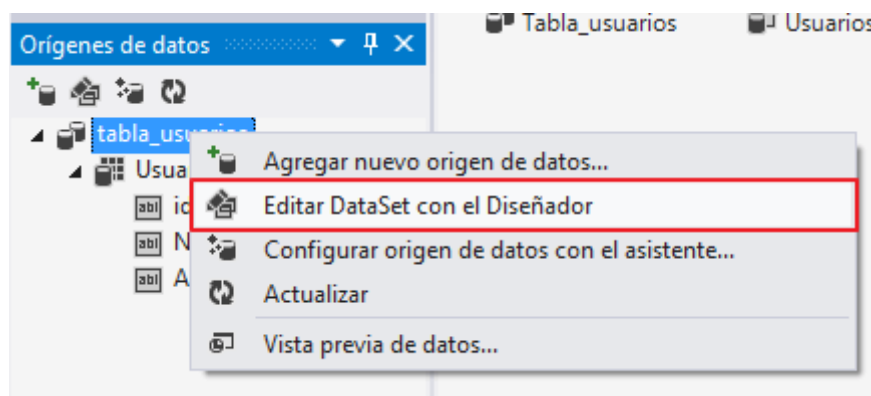
Propiedad	Descripción
InsertCommand	Objeto de la clase Command que realiza una inserción de datos
SelectCommand	Objeto de la clase Command que ejecuta una sentencia Select de SQL
UpdateCommand	Objeto de la clase Command que realiza una modificación de datos
DeleteCommand	Objeto de la clase Command que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

Volvamos a los iconos que nos ha creado:

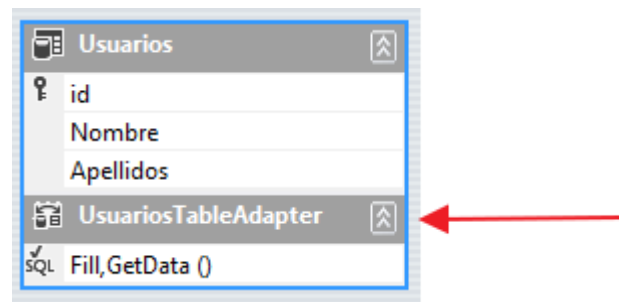


Tenemos un DataSet llamado "tabla_usuarios", una conexión "UsuariosBindingSource" y un DataAdapter que le ha llamado "UsuariosTableAdapter". Tenemos, por tanto, todo lo necesario: conexión, adaptador y dataset. Lógicamente es lo que nos ha añadido VB porque es lo que necesita para poder acceder a los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

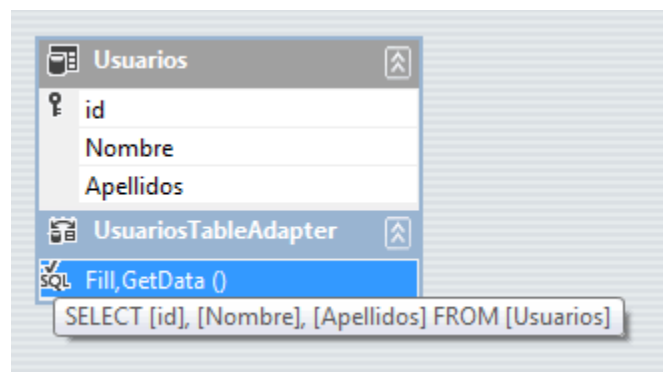
Ahora revisemos esto:



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":



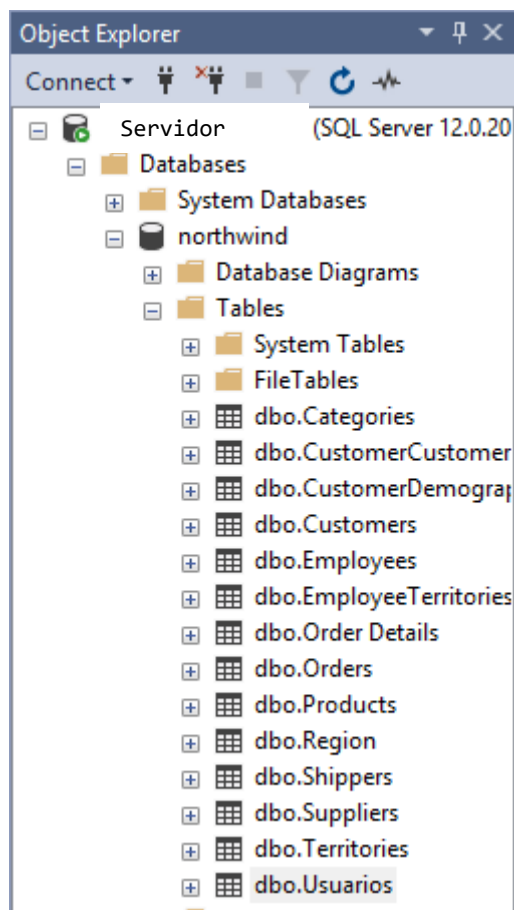
Este adaptador, como vemos, tiene almacenado cómo debe hacer la consulta para poder generar el dataset de "usuarios". Como ves es la conexión entre los datos físicos y nuestro Dataset con el resultado "Usuarios".

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:

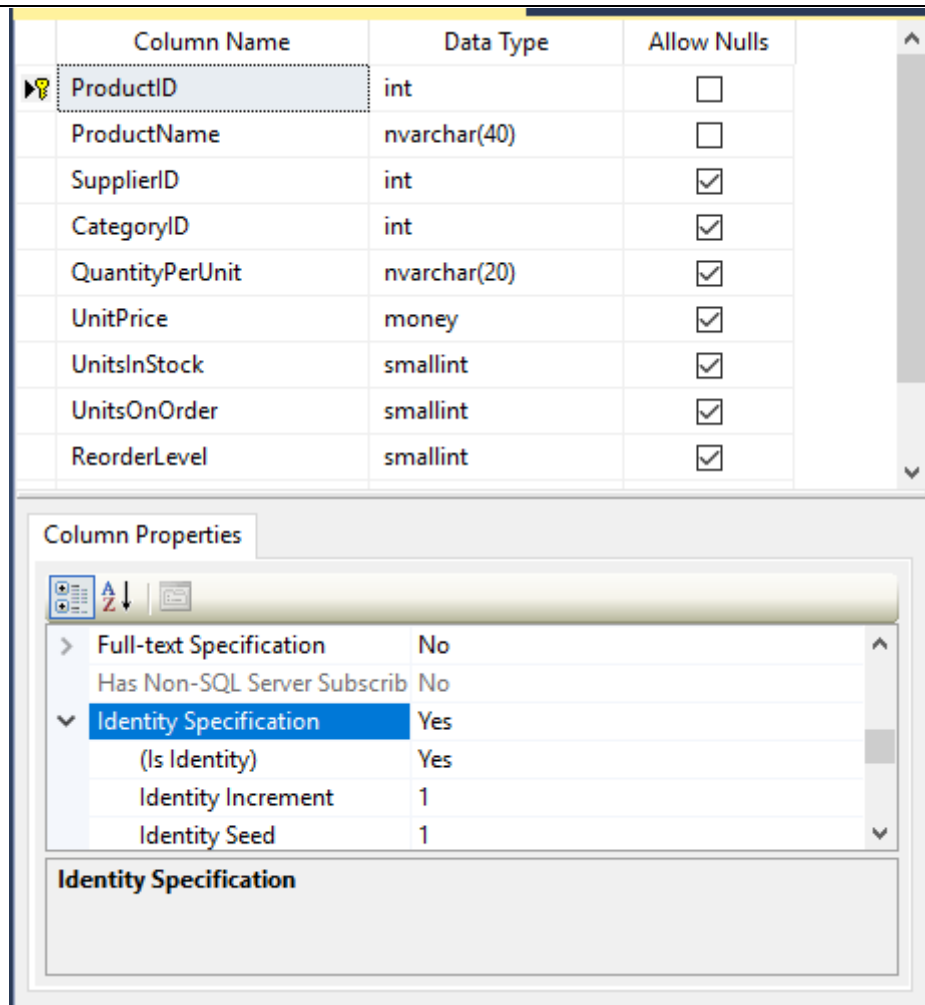


La primera ventana es un editor general de bases de datos. Simplemente es una herramienta para crear tablas, modificar su estructura,

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.

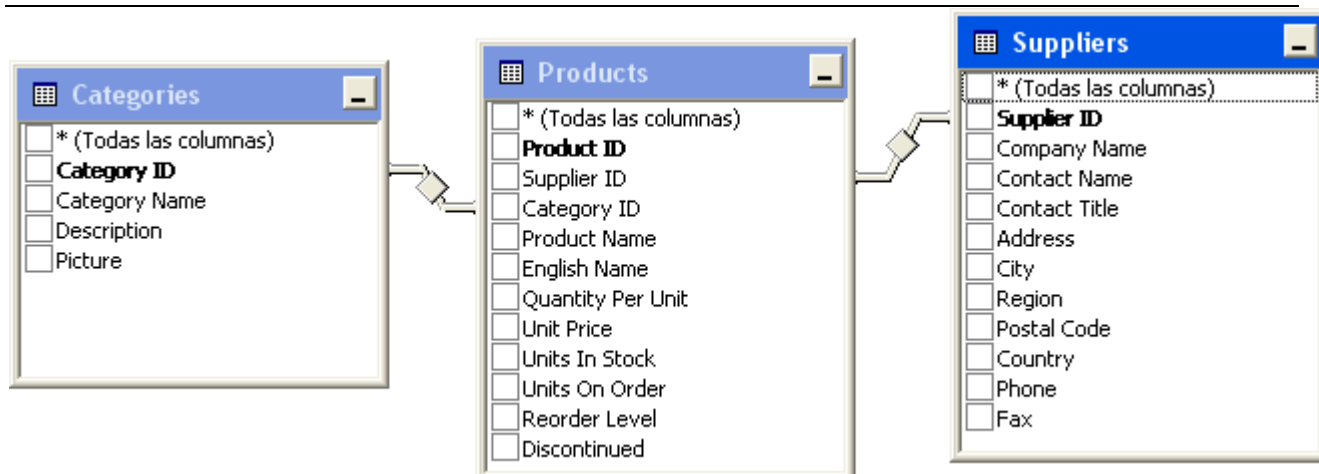
En el explorador de bases de objetos anterior vemos que hay una tabla llamada "Products", vamos a pulsar con el botón derecho para su "Diseño":



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.

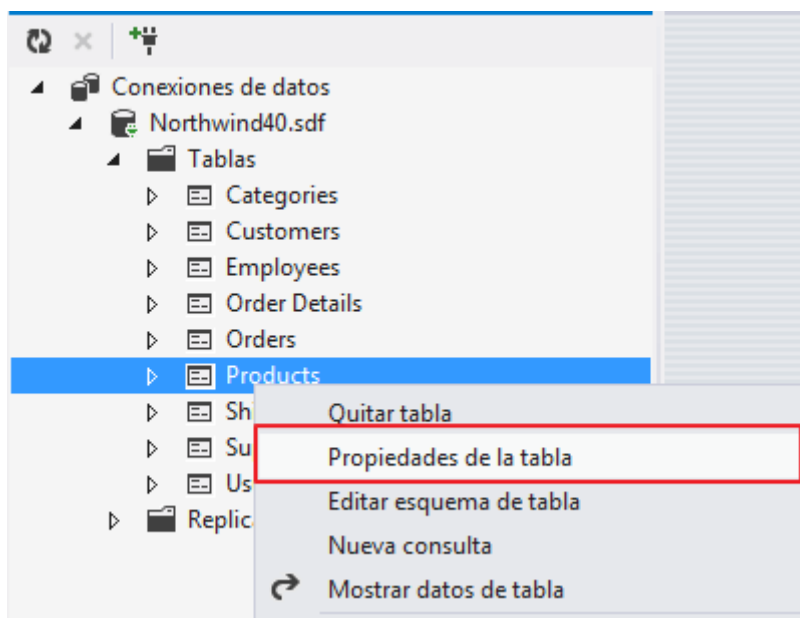
Vemos además como está relacionada con otras tablas, una de proveedores "Suppliers" (con el campo "SupplierID") y otra de categorías "Category" (con el campo CategoryID) donde almacena sólo el identificador de cada uno de estos, ya que con este dato quedan perfectamente conectadas las tablas.

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:

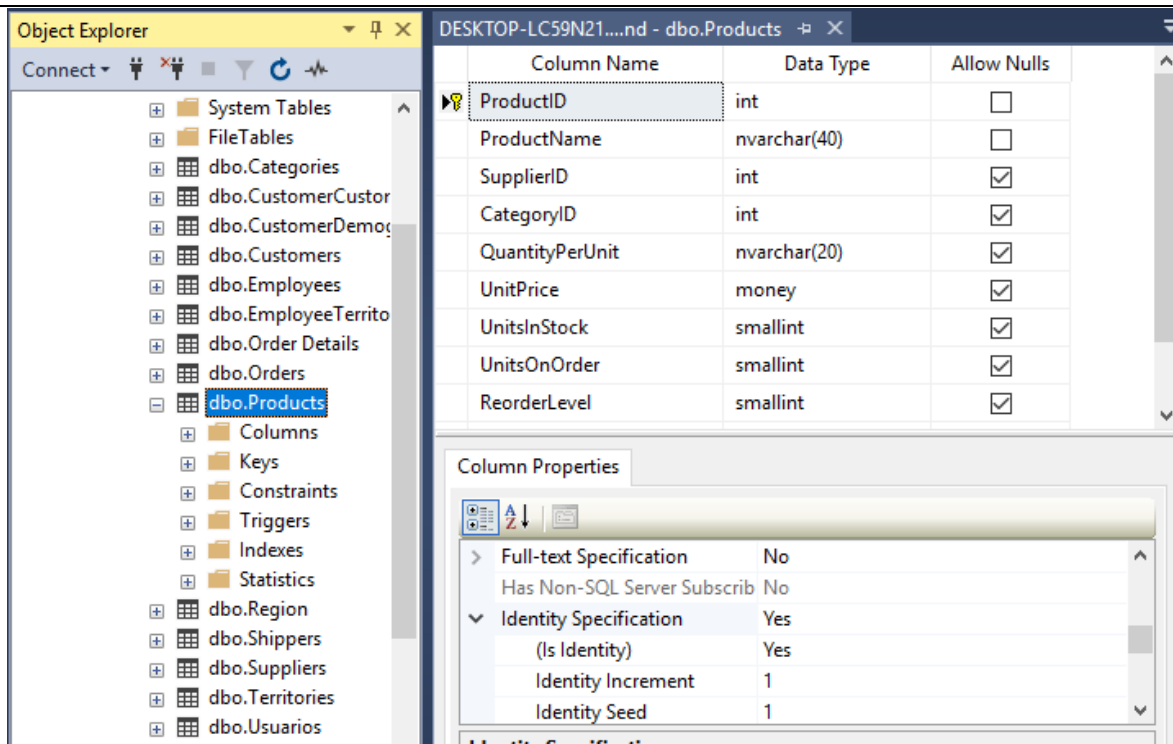


La tabla de productos está relacionada con la tabla de categorías a través del campo "Categoryid" que por supuesto son del mismo tipo de datos. La tabla Productos está relacionada también con la tabla de proveedores "Suppliers" a través del campo "SuplierID" que si te fijas, son los campos claves de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

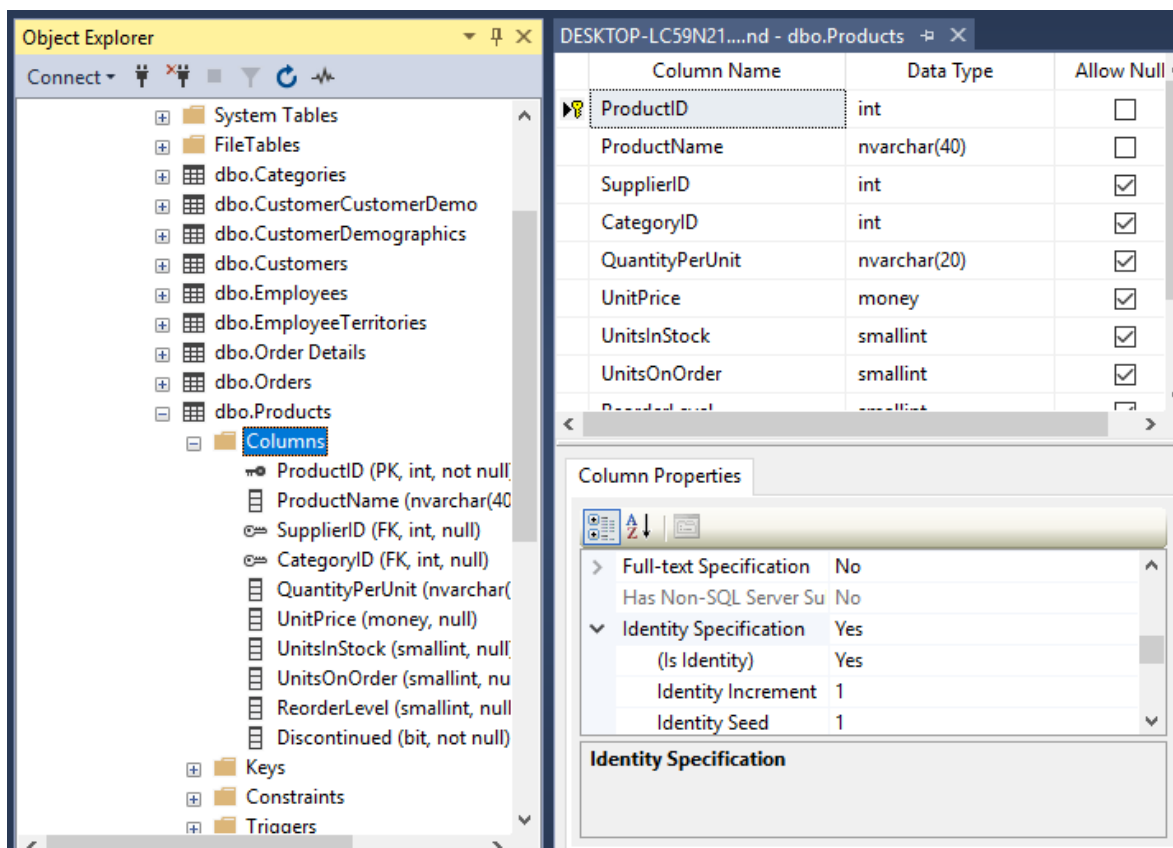
En nuestro ejemplo se producen dos relaciones de muchos a uno: Un proveedor puede tener muchos productos y lo mismo con una categoría que la pueden tener muchos productos. Vamos a pulsar con el botón derecho en la tabla de los pedidos categorías y le vamos a decir que:



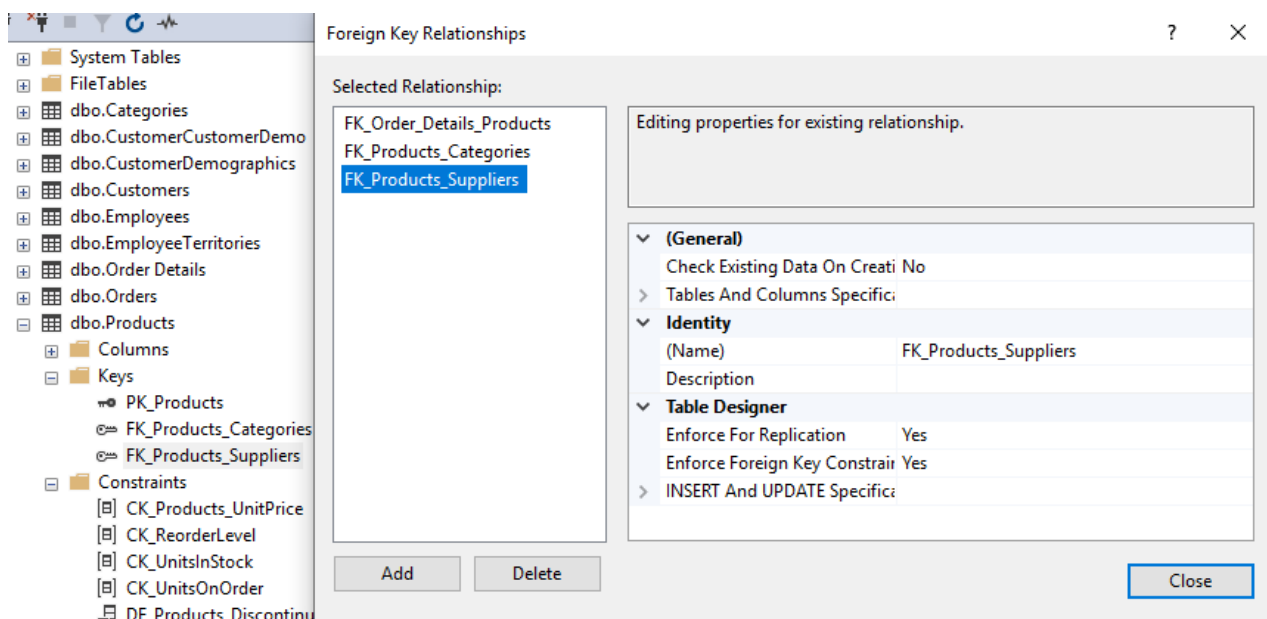
Queremos ver sus propiedades. No editarla sino sólo ver sus propiedades:



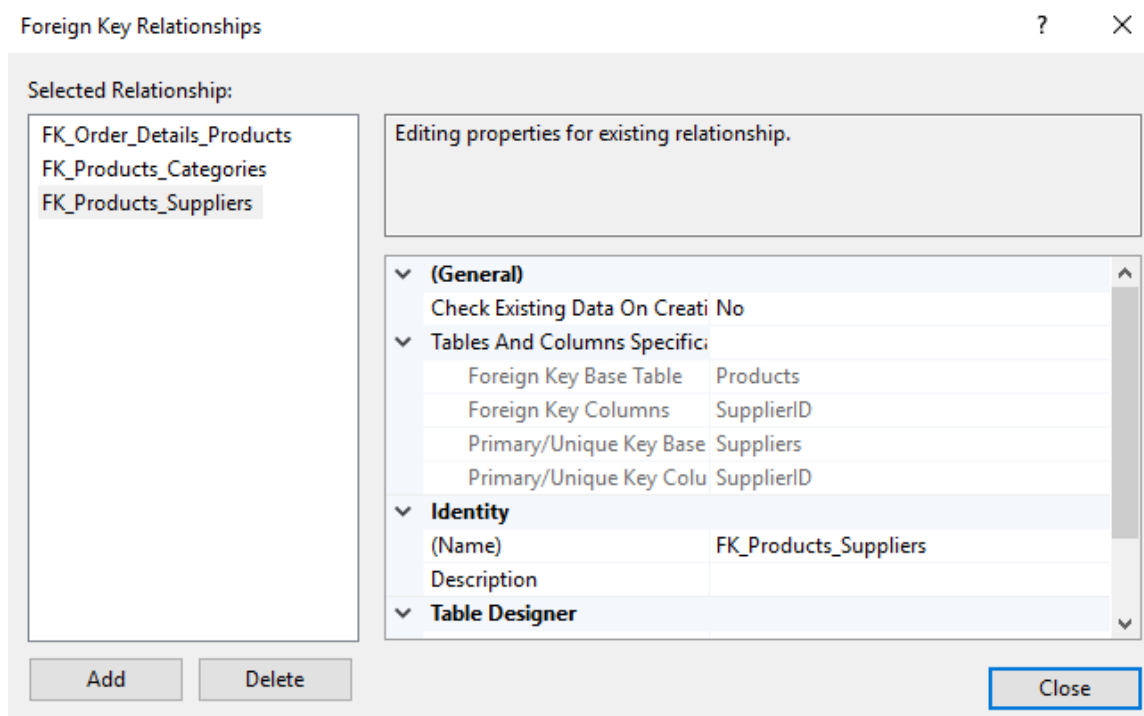
En la segunda opción de la izquierda, "Columna", podemos ver todos los detalles de la tabla. Fíjate que nos indica que campos son claves, el tipo de datos, si alguno de ellos es de tipo "identity" y algunos datos interesantes más.



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 "SupplierID" que se llaman de igual forma en las dos tablas, así es como debe ser.

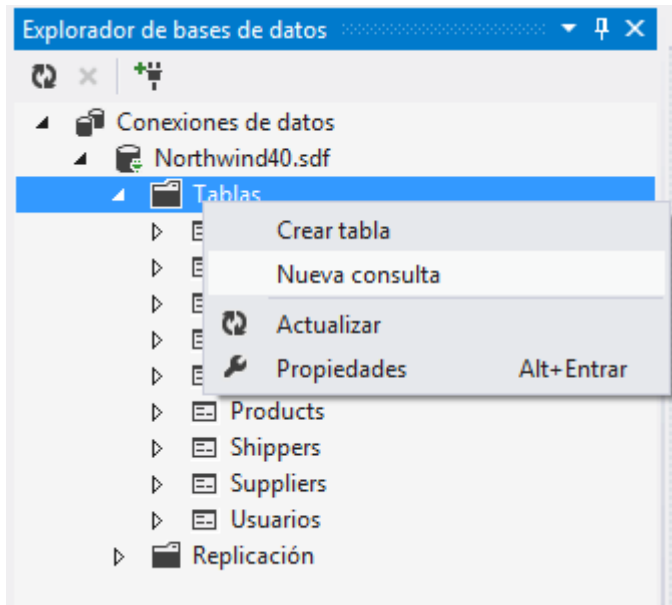


Si pulsamos en el desplegable, veremos la otra relación existente con la otra tabla.

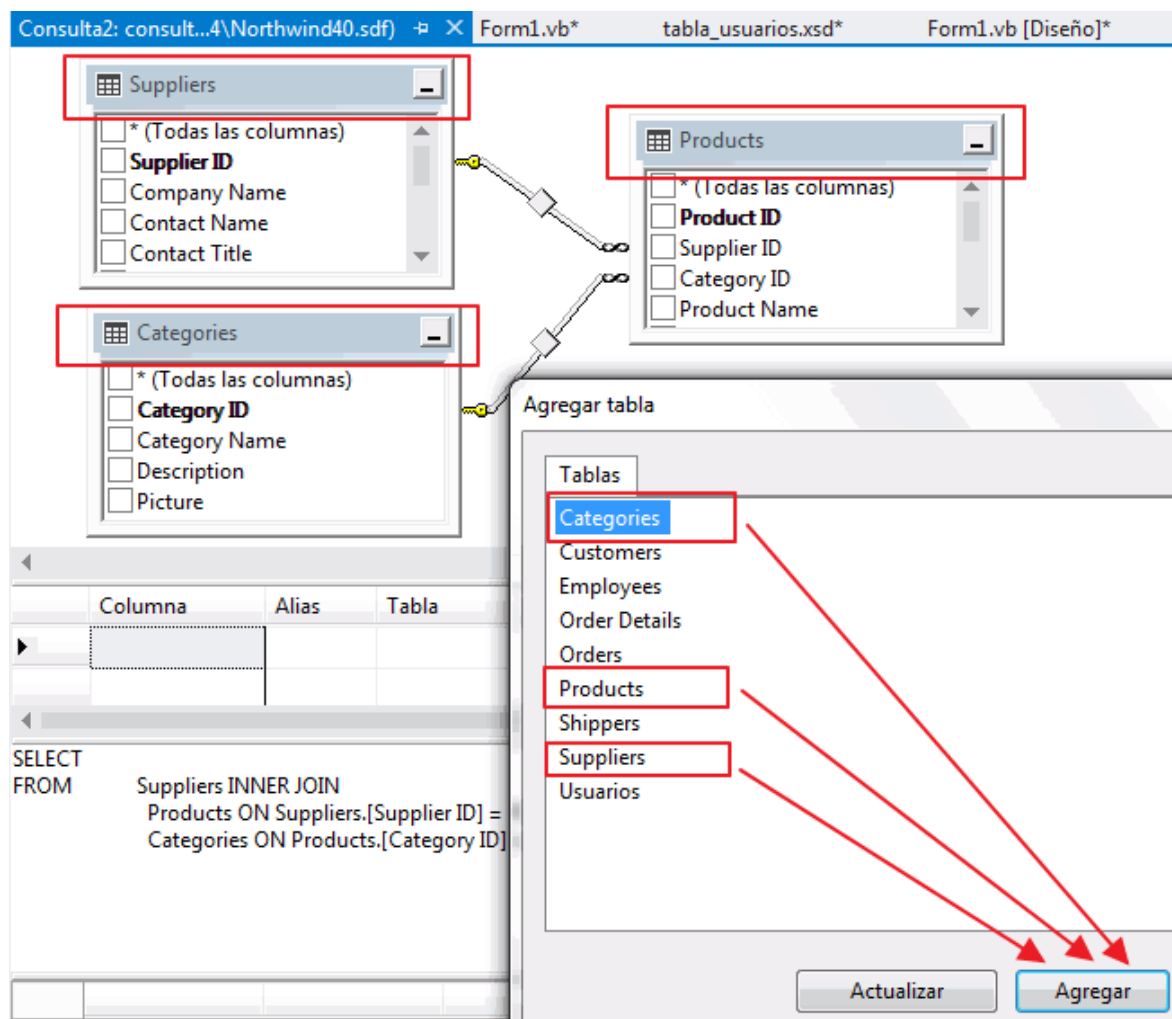
La opción de "Agregar" permitiría crea una nueva.

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

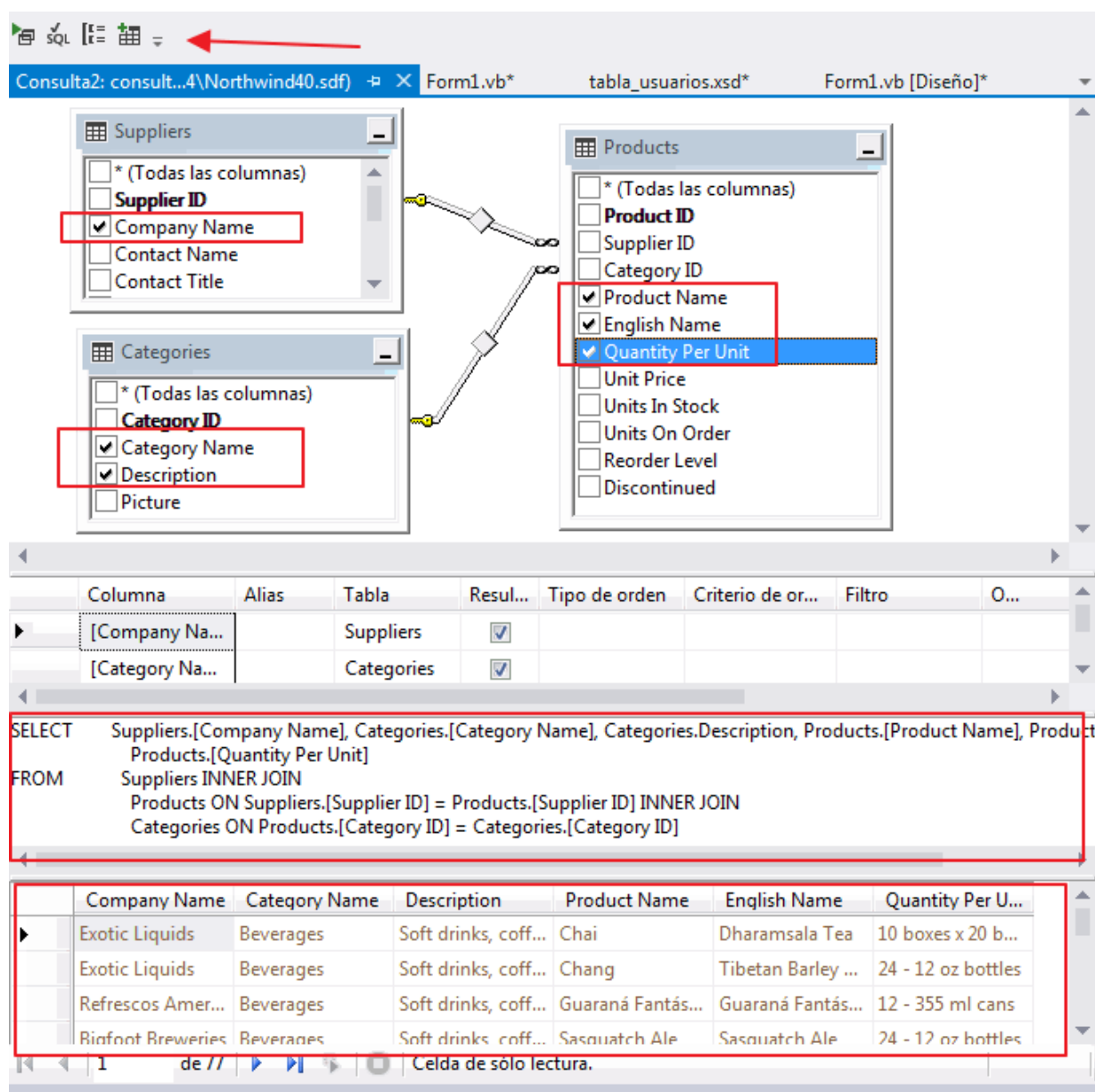
Vamos a realizar ahora una consulta cruzando estas tres tablas. Pulsamos con el botón derecho en nuestra base de datos o tablas para elegir una nueva consulta:



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:



Columna	Alias	Tabla	Resul...	Tipo de orden	Criterio de or...	Filtro	O...
[Company Na...		Suppliers	<input checked="" type="checkbox"/>				
[Category Na...		Categories	<input checked="" type="checkbox"/>				

```

SELECT Suppliers.[Company Name], Categories.[Category Name], Categories.Description, Products.[Product Name], Products.[Quantity Per Unit]
FROM Suppliers INNER JOIN
Products ON Suppliers.[Supplier ID] = Products.[Supplier ID] INNER JOIN
Categories ON Products.[Category ID] = Categories.[Category ID]
    
```

	Company Name	Category Name	Description	Product Name	English Name	Quantity Per U...
▶	Exotic Liquids	Beverages	Soft drinks, coff...	Chai	Dharamsala Tea	10 boxes x 20 b...
	Exotic Liquids	Beverages	Soft drinks, coff...	Chang	Tibetan Barley ...	24 - 12 oz bottles
	Refrescos Amer...	Beverages	Soft drinks, coff...	Guaraná Fantás...	Guaraná Fantás...	12 - 355 ml cans
	Bigfoot Breweries	Beverages	Soft drinks, coff...	Sasquatch Ale	Sasquatch Ale	24 - 12 oz bottles

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: :



Elegir un tipo de origen de datos

¿De dónde obtendrá la aplicación los datos?

 Base de datos	 Servicio	 Objeto
--	---	---



Le permite conectarse a una base de datos y elegir los objetos de base de datos para la aplicación.

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



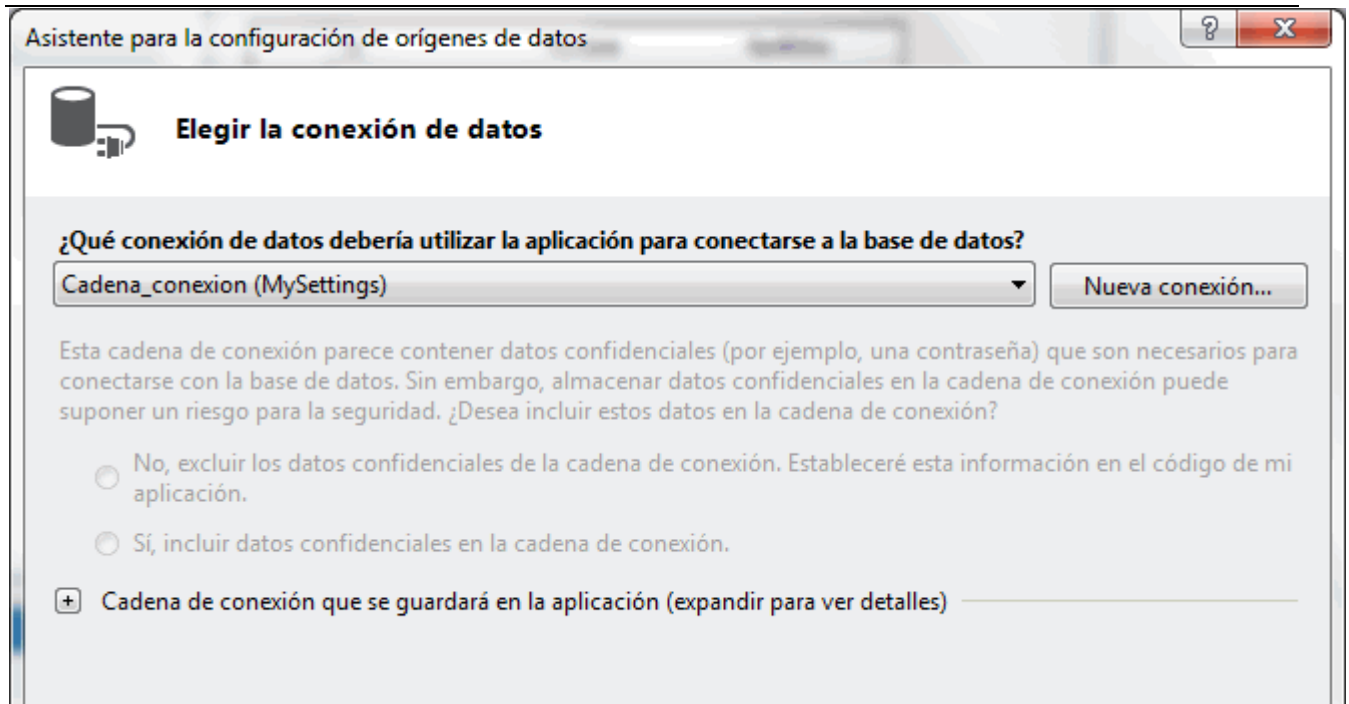
Elegir un modelo de base de datos

¿Qué tipo de modelo de base de datos desea usar?

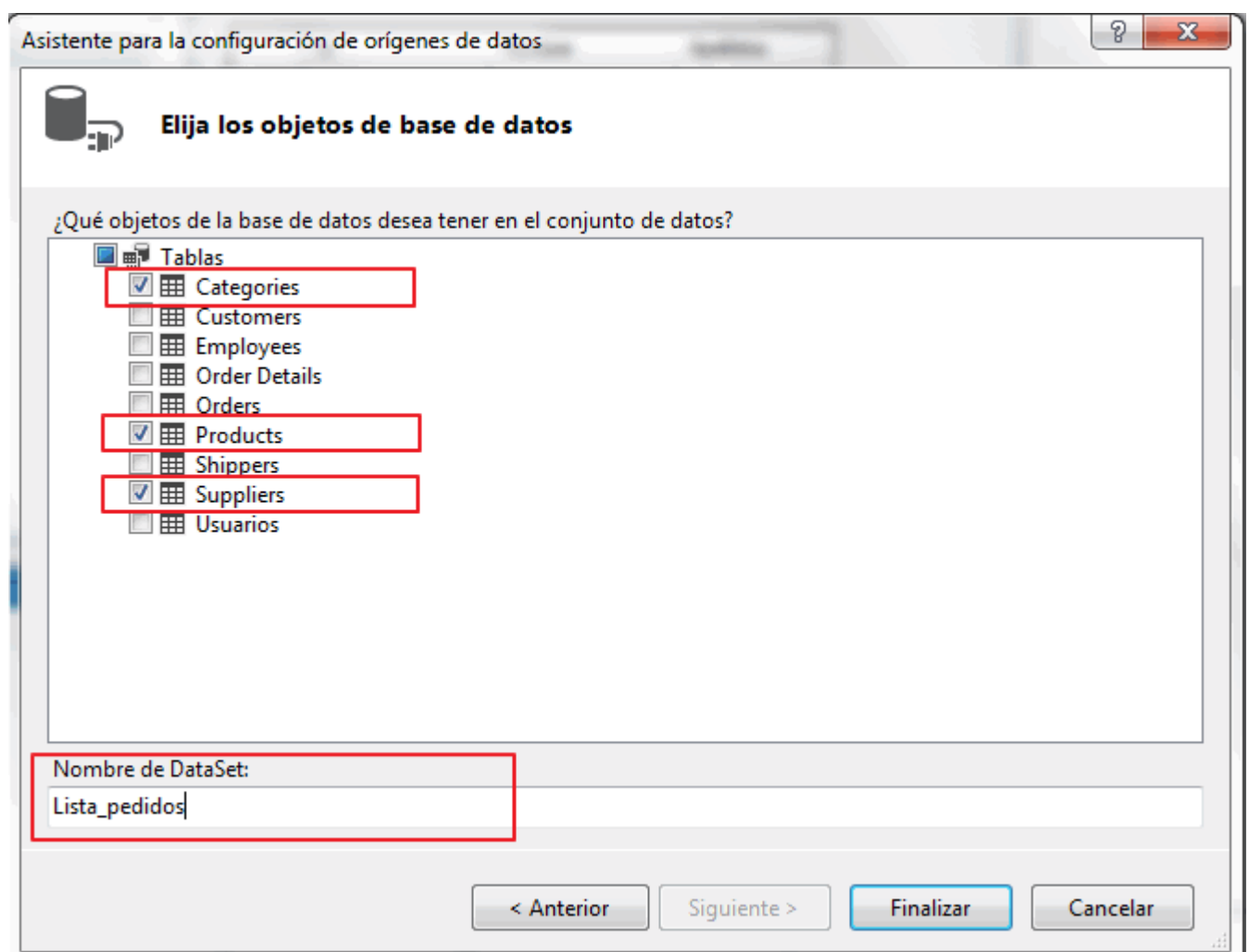
 Conjunto de datos	 Entity Data Model
--	--

El modelo de base de datos que elija determina los tipos de objetos de datos que utiliza el código de la aplicación. Se agregará un archivo de conjunto de datos al proyecto.

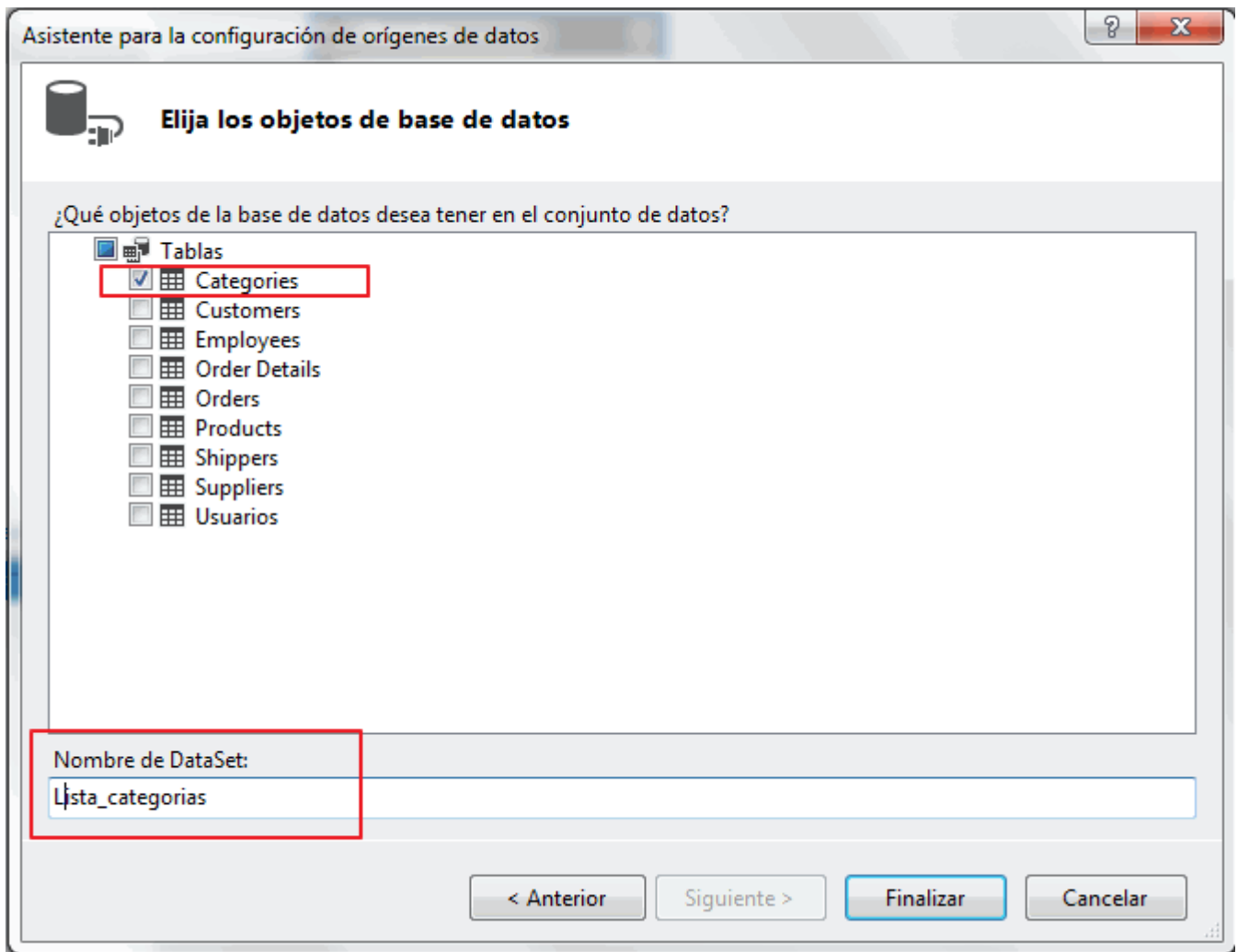
Seleccionamos posteriormente la conexión que está ya almacenada en nuestro proyecto:



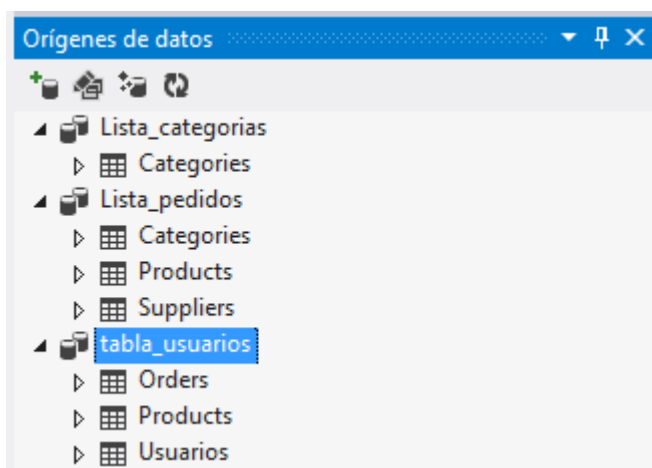
Y finalmente le decimos las tres tablas de antes:



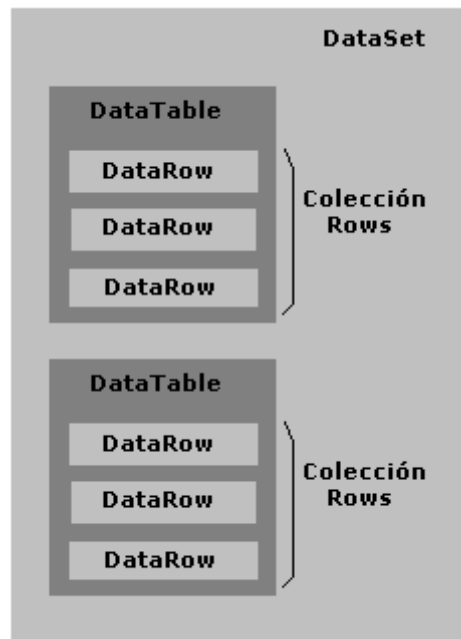
Le ponemos un nombre debajo, en el Dataset y terminamos. Ahora creamos uno más con las categorías:



Así que tendremos estos tres DataSet definidos:



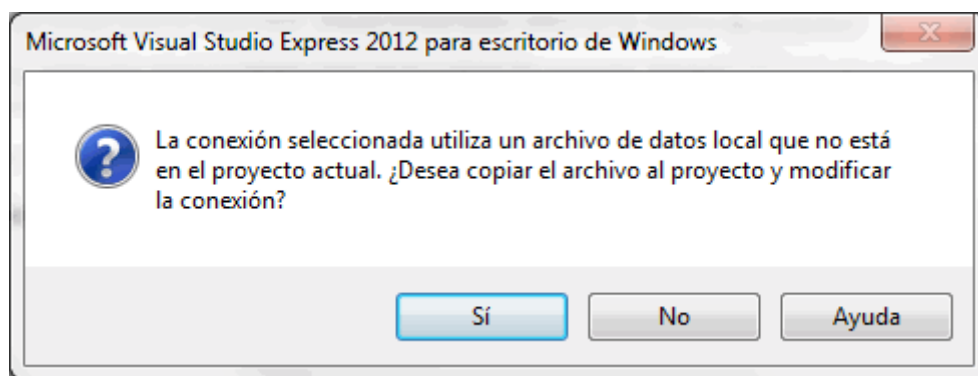
¿Recuerdas el gráfico de los objetos en los que se componía un DataSet? :



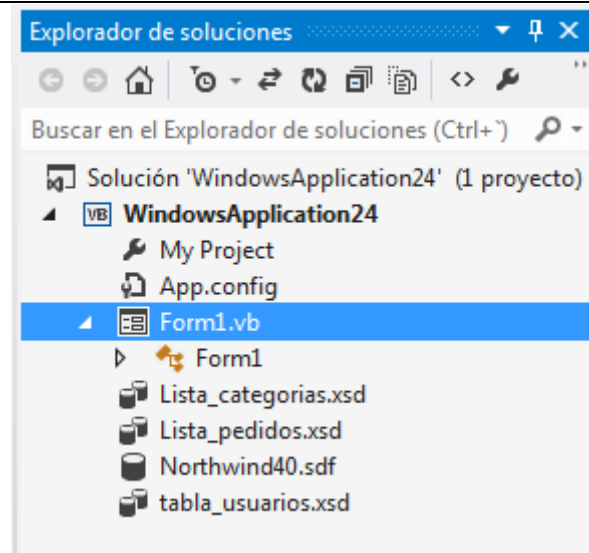
Ya ves que se parece bastante ya que hay dos que solo tienen una tabla, pero hay otro dataset que tiene tres tablas (**DataTable**). Y cada una de ellas los datos en colecciones de filas o "Rows".

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:

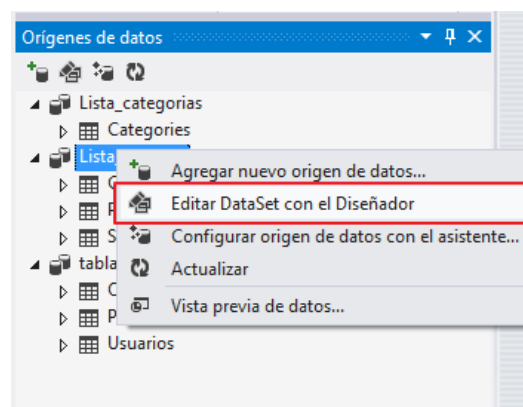


Ahora nuestro proyecto se compone de:

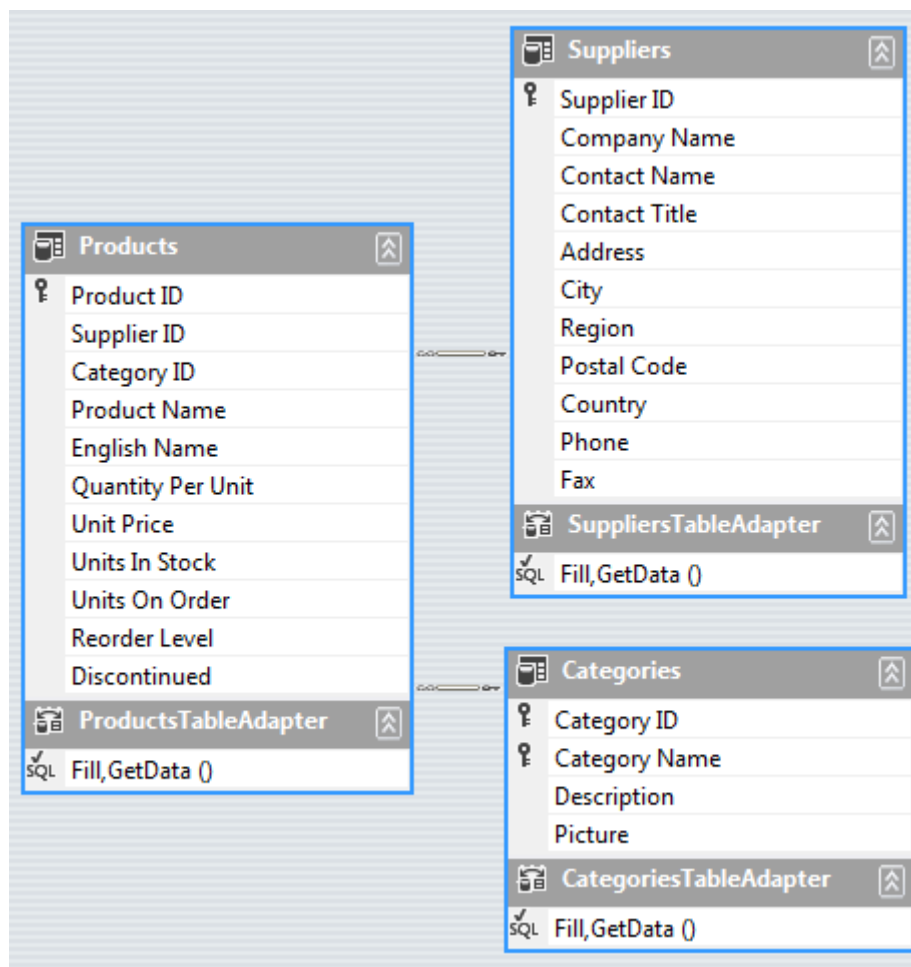
- "My Project". Fichero de configuración del entorno del proyecto: formulario de inicio, opciones como "Option Explicit", "Exstrict", ...
- App.config. Fichero en formato XML con información adicional del proyecto: conexiones de bases de datos, ... si le das doble clic verás entre todas las etiquetas 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:

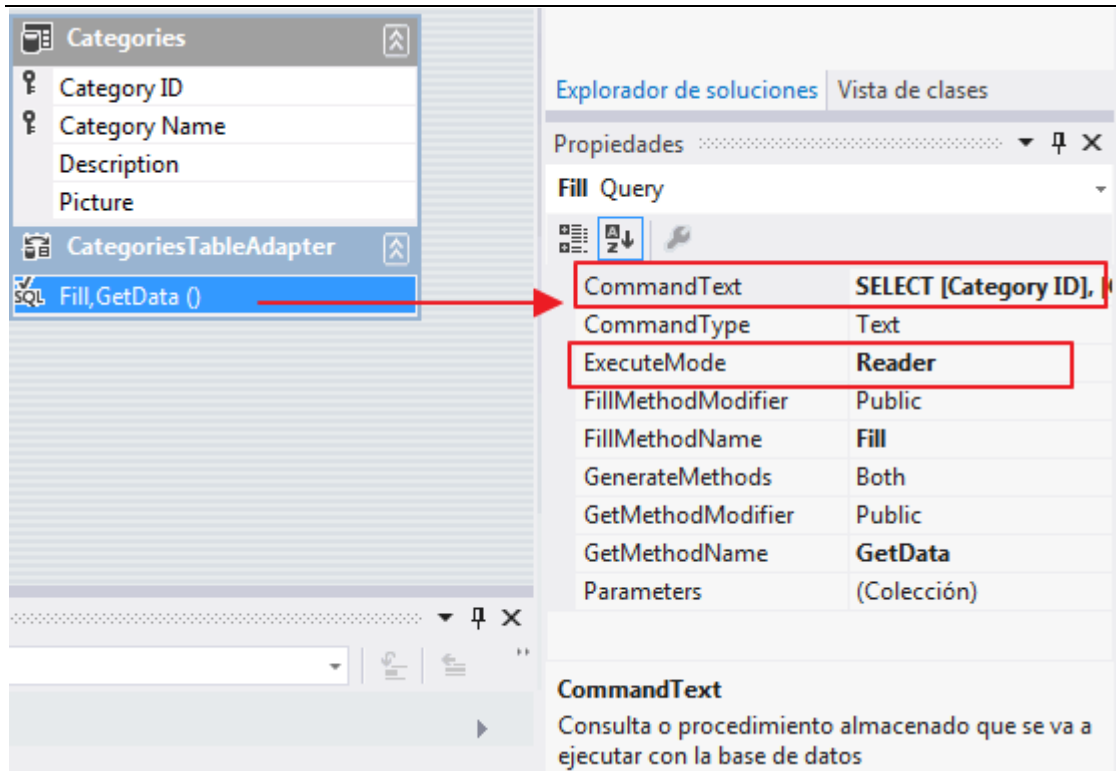


Vemos que el aspecto es parecido al del editor:

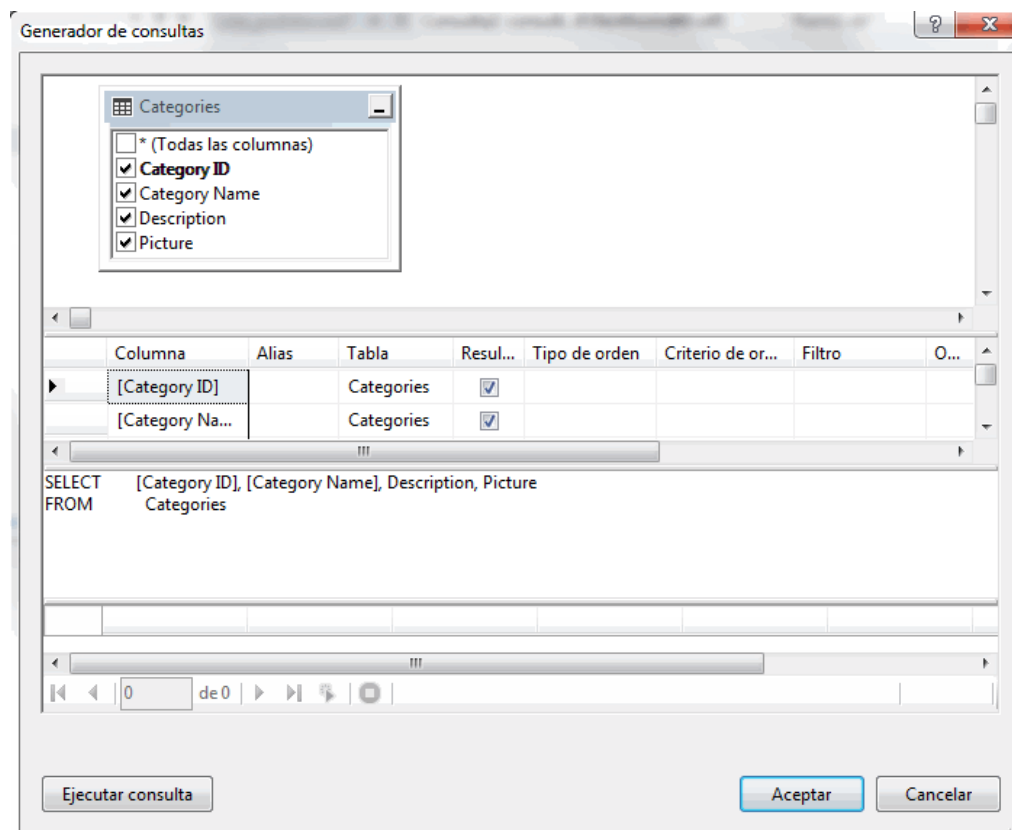


Pero no te confundas, aquí estamos editando los orígenes de datos en forma de Datasets, no las bases de datos con sus tablas. Son pues, cosas muy distintas. Fíjate que nos muestra también las relaciones y nos indica la dirección de la relación "1 a muchos" ya que aparece el símbolo de infinito en la 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:



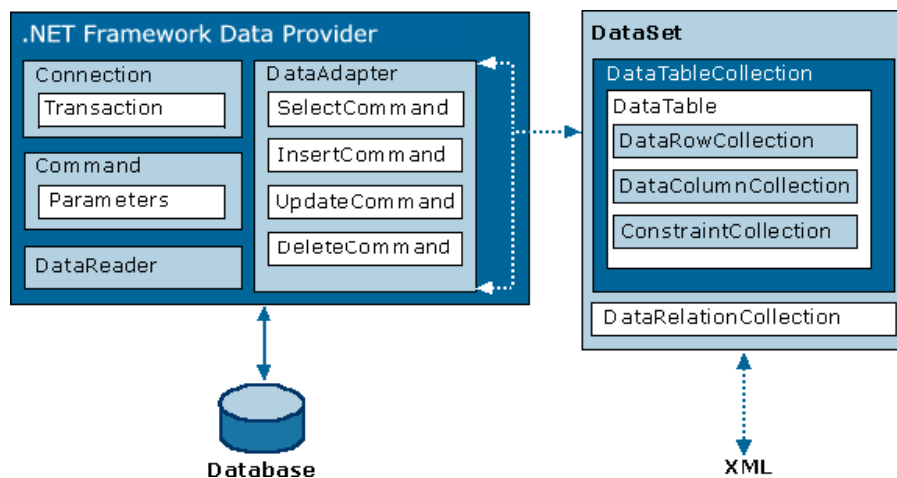
Es decir, este adaptador tiene ya definida un comando para consultar datos. Se ejecuta como un "reader", que veremos más adelante. En un CommandText tenemos la sentencia SQL asociada, que si pulsamos para ver los detalles de este campo de la ventana de propiedades nos mostrará:



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.

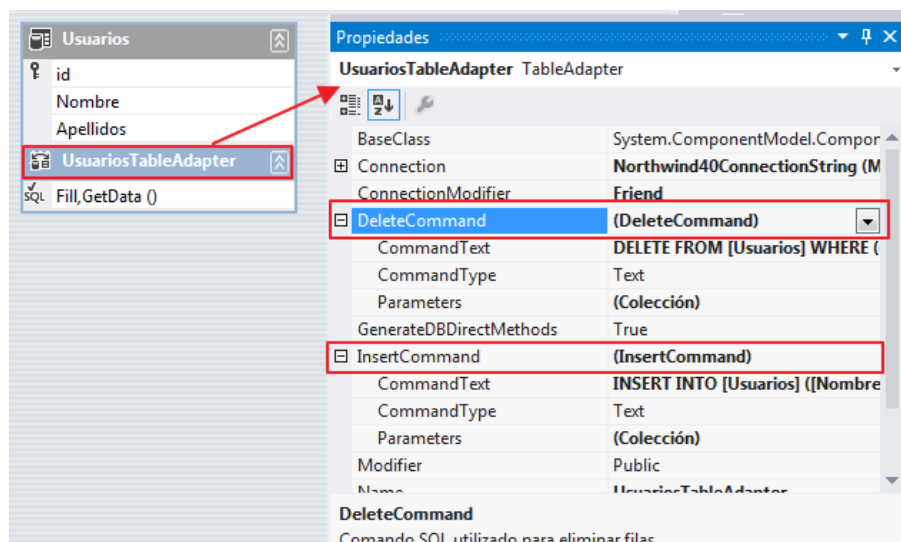
Pero aquí hay algo que falla, o por lo menos no entiendo bien... si el adaptador es el que contiene los comandos para conectar la base de datos con nuestro Dataset ¿por qué aparece sólo un comando para consultar? ¿no deberían aparecer los demás comandos de borrar (delete), modificar (update) e insertar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

Veamos este gráfico:

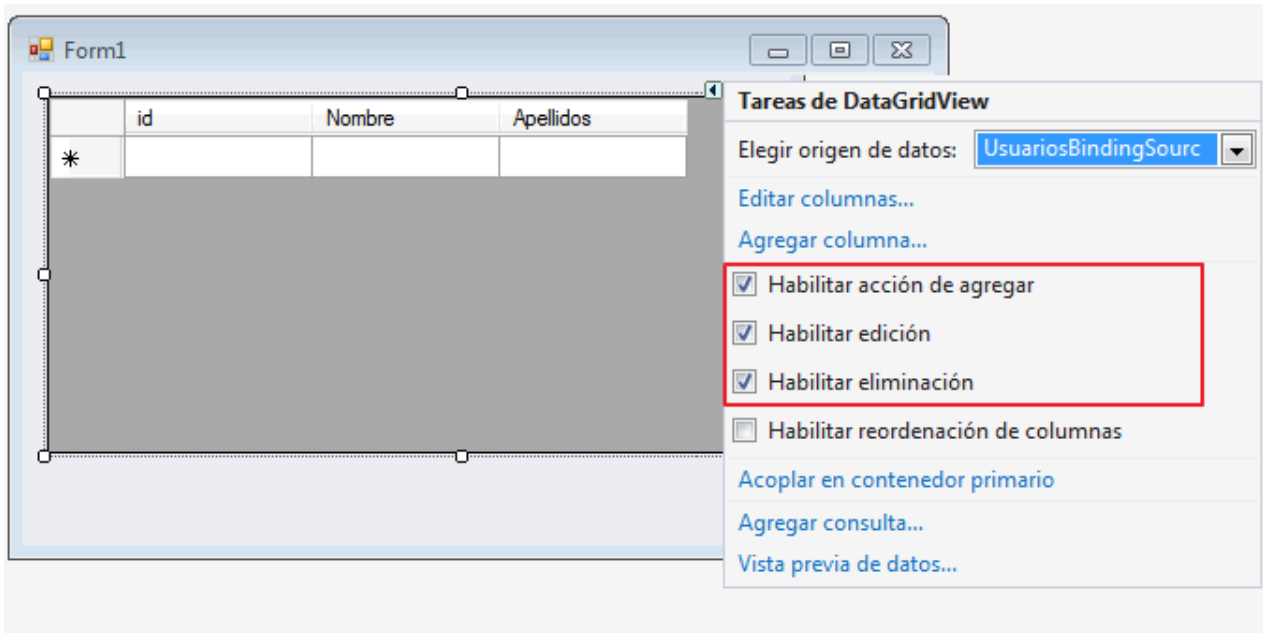


Que viene a ser lo mismo de siempre pero visto de otra forma. Vemos en la parte de la izquierda que tenemos una conexión y a su derecha el adaptador con los cuatro operaciones que puede contener y que de momento solo tenemos la de la consulta.

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:

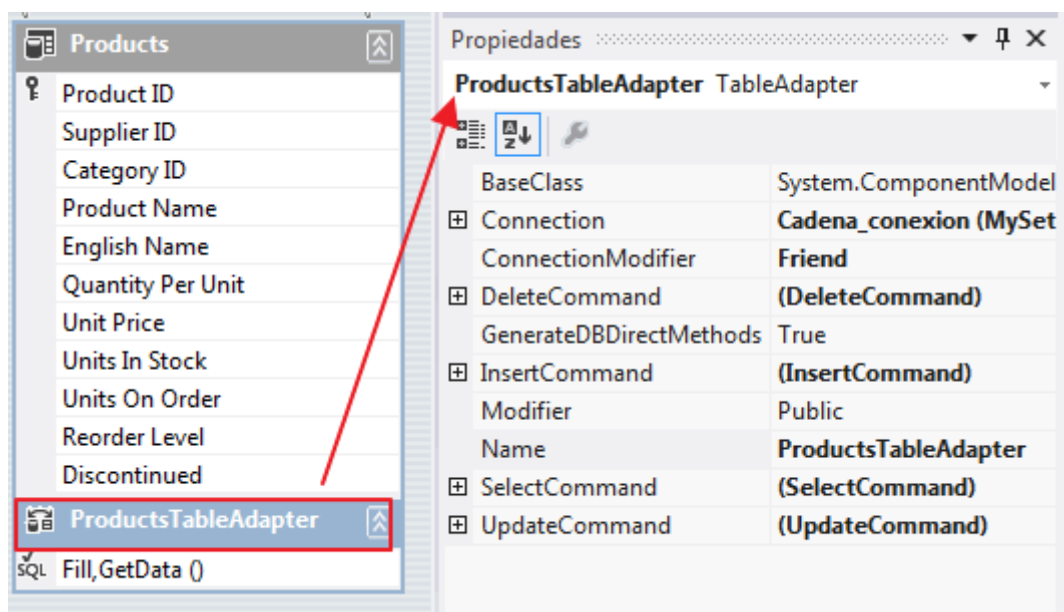


Vemos, haciendo clic en "UsuariosTableAdapter" que a la derecha nos muestra en las propiedades que, efectivamente están incluidas las instrucciones para hacer estas operaciones. ¿y por qué me las ha puesto? Pues porque cuando hicimos el enlace del origen de datos con el control de cuadrícula le indicamos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



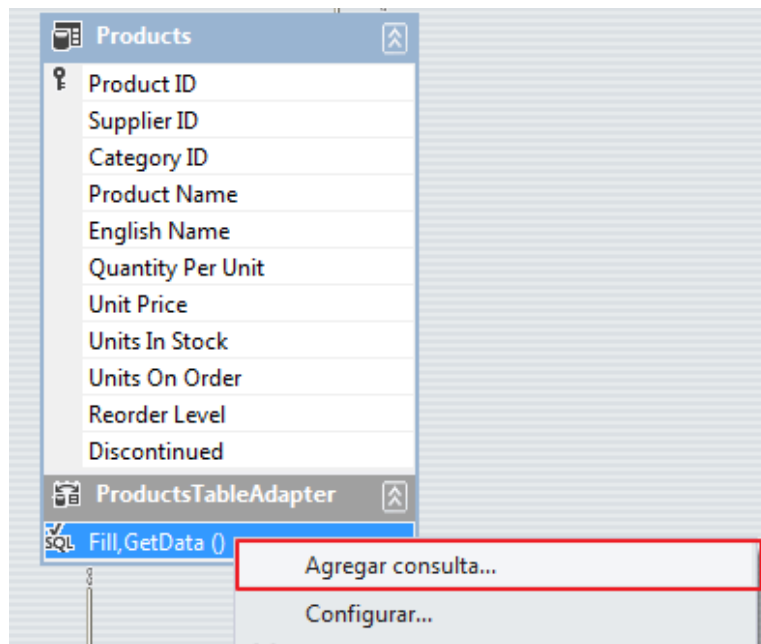
Recuerda que hemos asociado una tabla a un control DataGrid y además en el IDE le hemos dicho que se pueda editar. El precio de todo esto es crear un dataset y un adaptador con los comandos correspondiente.

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

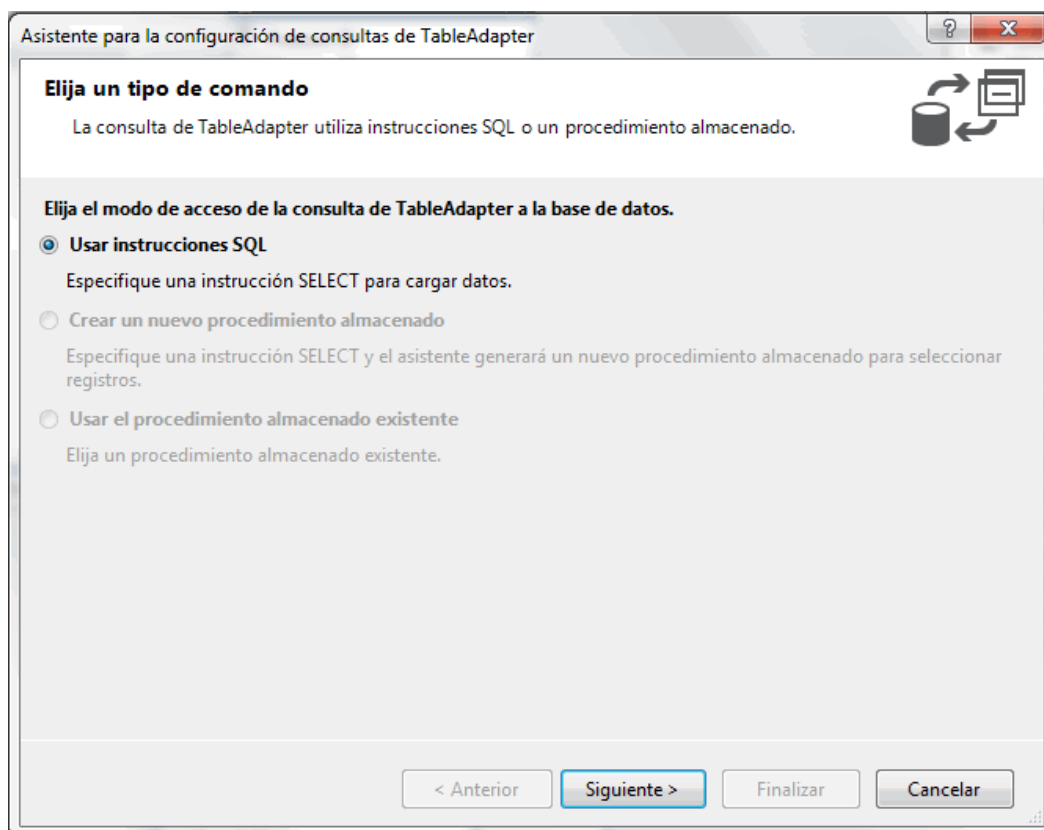


Bien, vemos de nuevo los comandos de mantenimiento de la tabla: Insert, Delete, Update y Select.

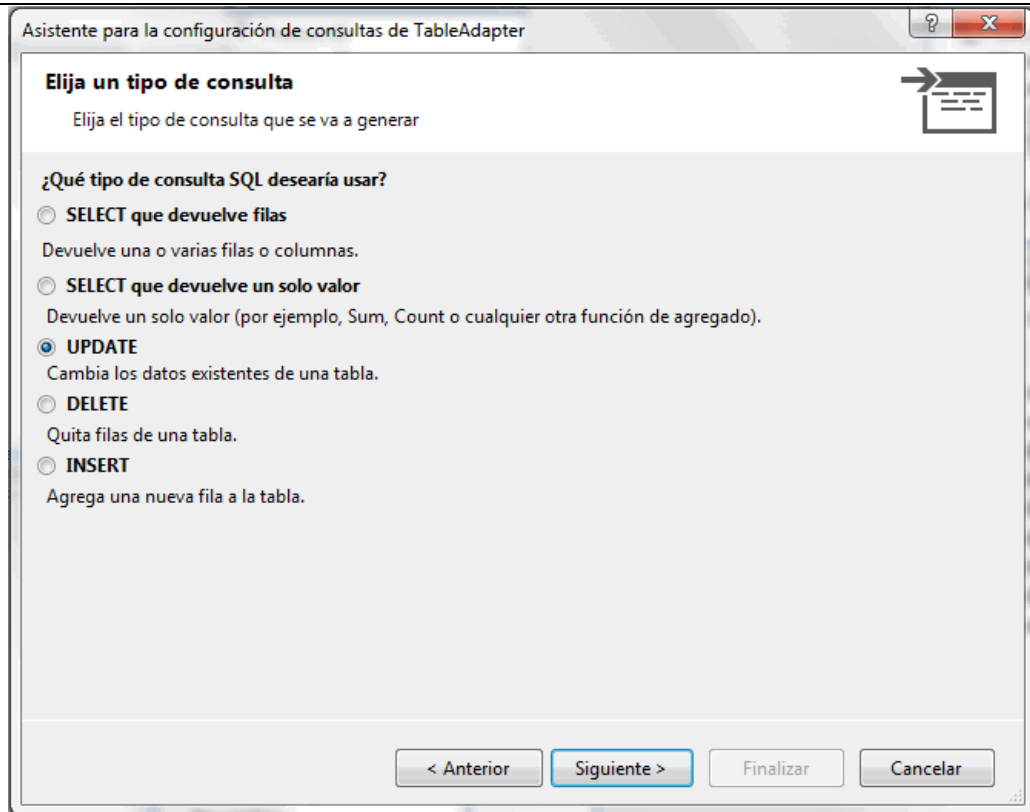
Aunque están puestos ya de forma automática, vamos a ver cómo añadirlos. Seleccionamos "Agregar consulta":



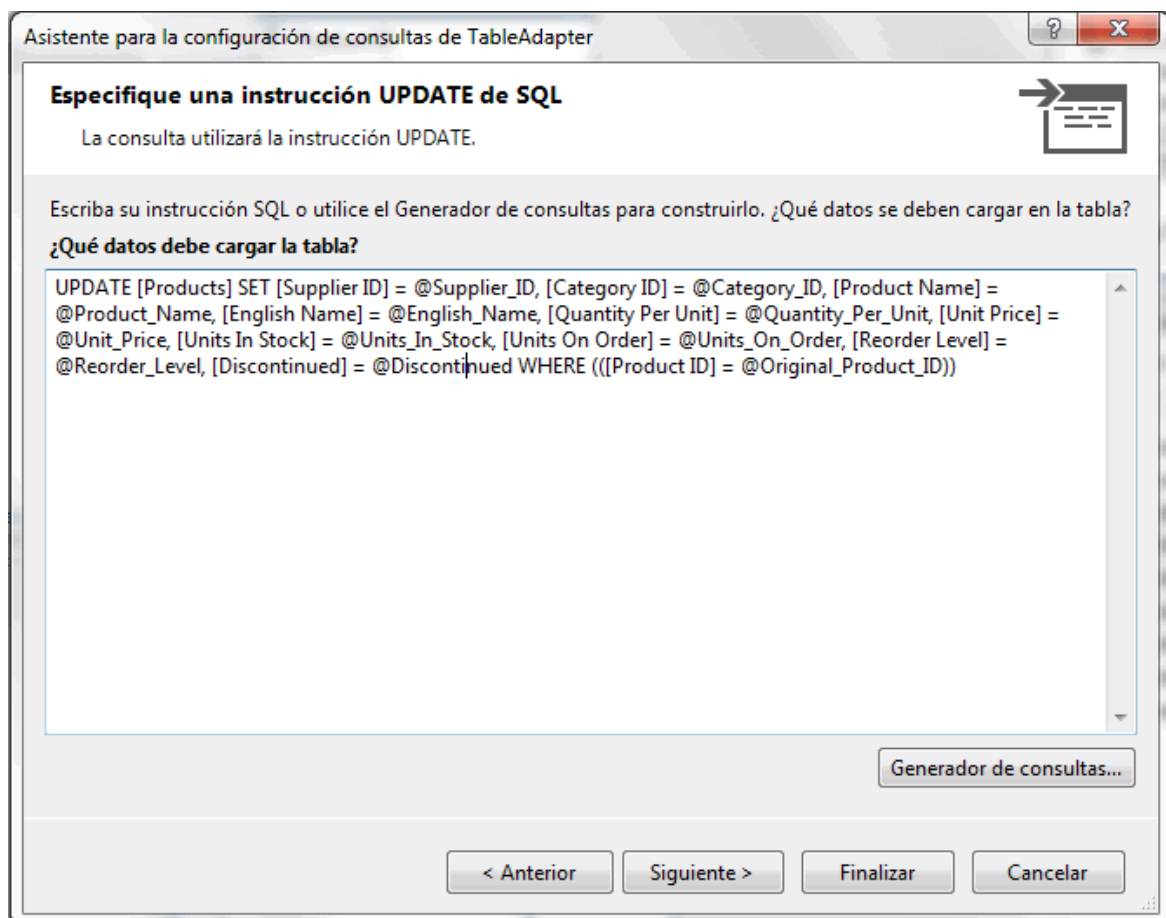
Se pone en marcha un asistente para la creación de la consulta:



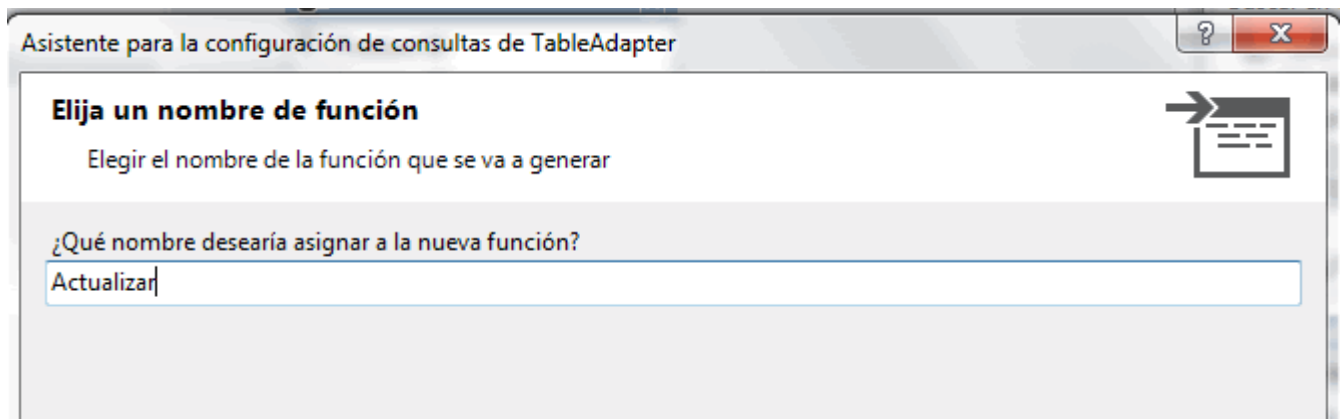
Solo nos dejar hacer esa operación. Las otras dos dependen del origen de datos, en un SQL Server completo podríamos seleccionar las otras para ejecutar un procedimiento almacenado. Pulsamos en siguiente:



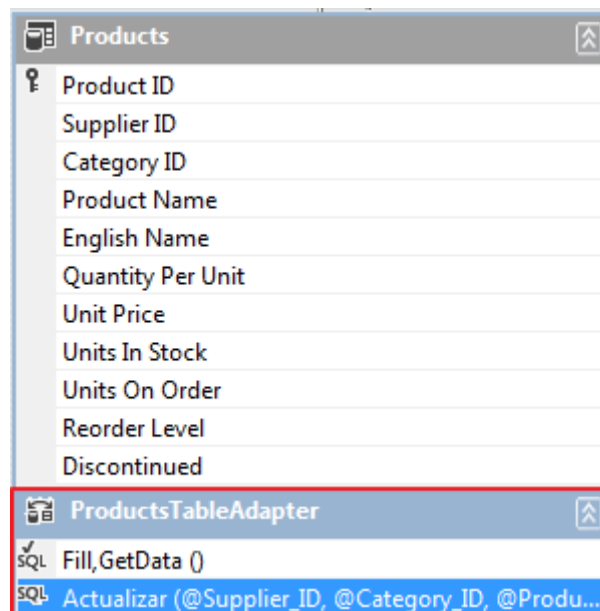
Y queremos añadir una instrucción para actualizar (Update):



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:

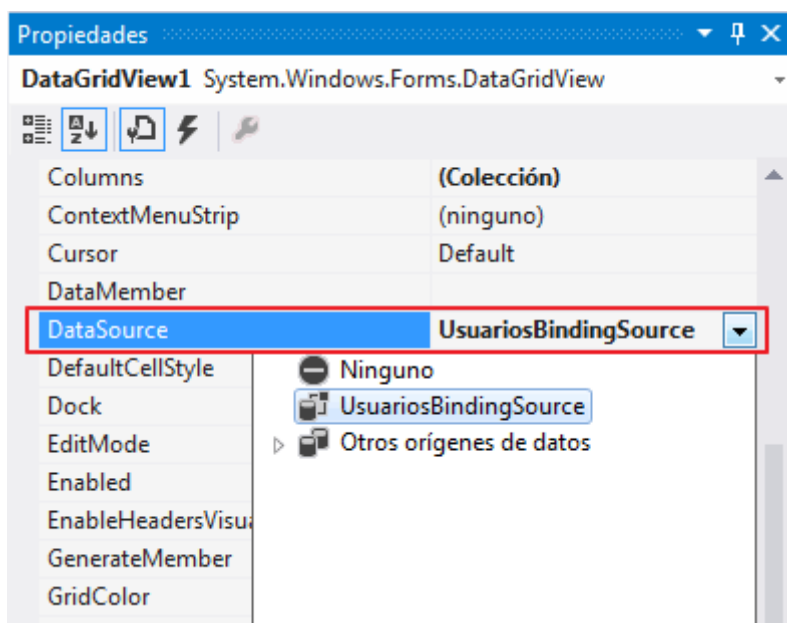


Los comandos (altas, bajas, consultas y modificaciones) ya están añadidos en el adaptador por defecto al utilizar este asistente. Lo que podemos hacer con esto último es añadir más comandos para aplicarse unos u otros en el código.

7.3. Controles enlazados a datos

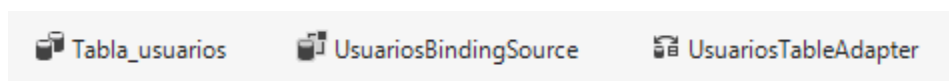
En prácticamente todos los proyectos de aplicaciones de Windows se utilizan los controles enlazados a datos. El ejemplo más sencillo lo hemos visto antes cuando con dos clics asociamos una tabla a un origen de datos. Su uso es realmente sencillo porque el "trabajo sucio" ya está realizado al tener los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



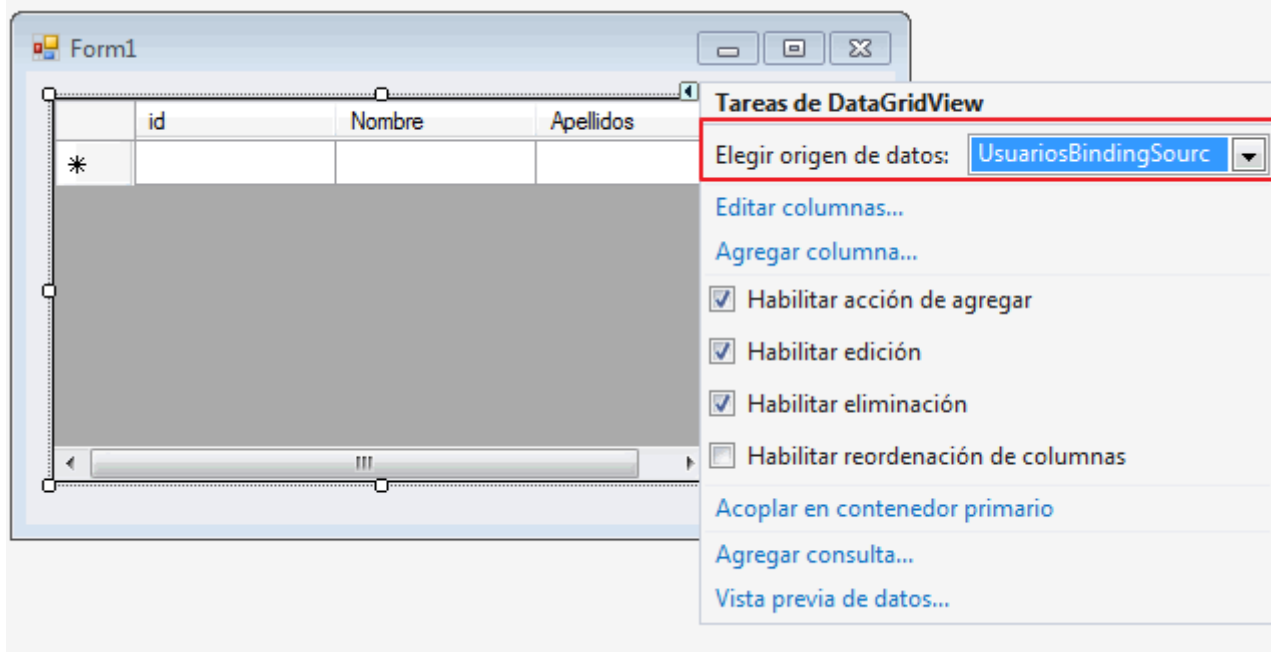
Es decir, se trata de la conocida propiedad "DataSource", a la que se puede acceder desde el menú rápido que aparece junto al control o desde la ventana de propiedades. Como vemos nos indica que al seleccionar un origen de datos se crea una instancia en el proyecto y se enlaza a través de un nuevo "BindingSource". 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:

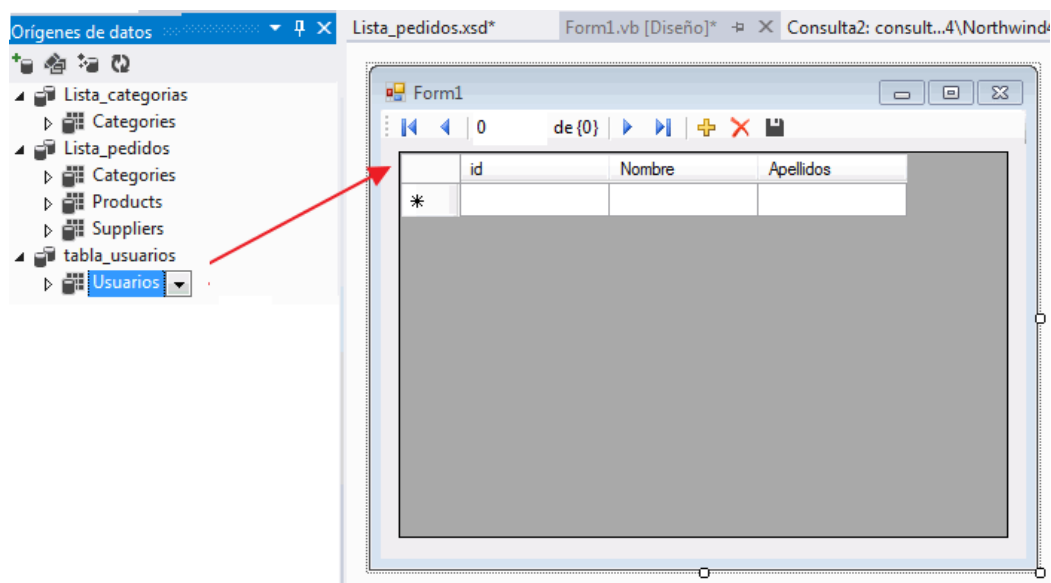


Podemos decirle las funcionalidades que queremos añadir para poderse editar y añadir filas.

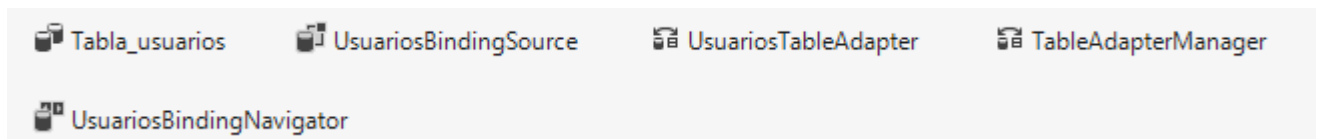
Más sobre el enlace de datos

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.

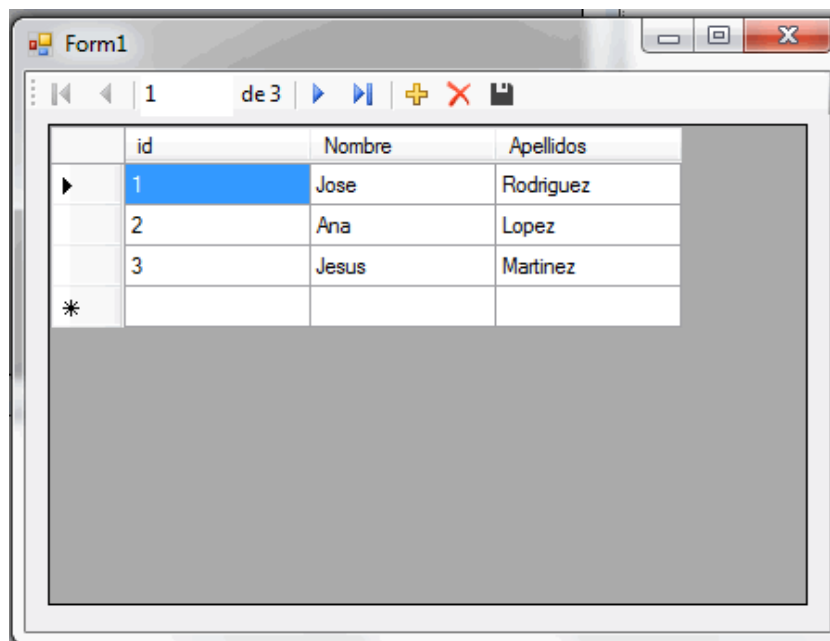
Coge un "dataset" cualquiera de los que tenemos en la lista de orígenes de datos y arrástralo a un formulario vacío. Veremos que nos coloca todo esto:



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:

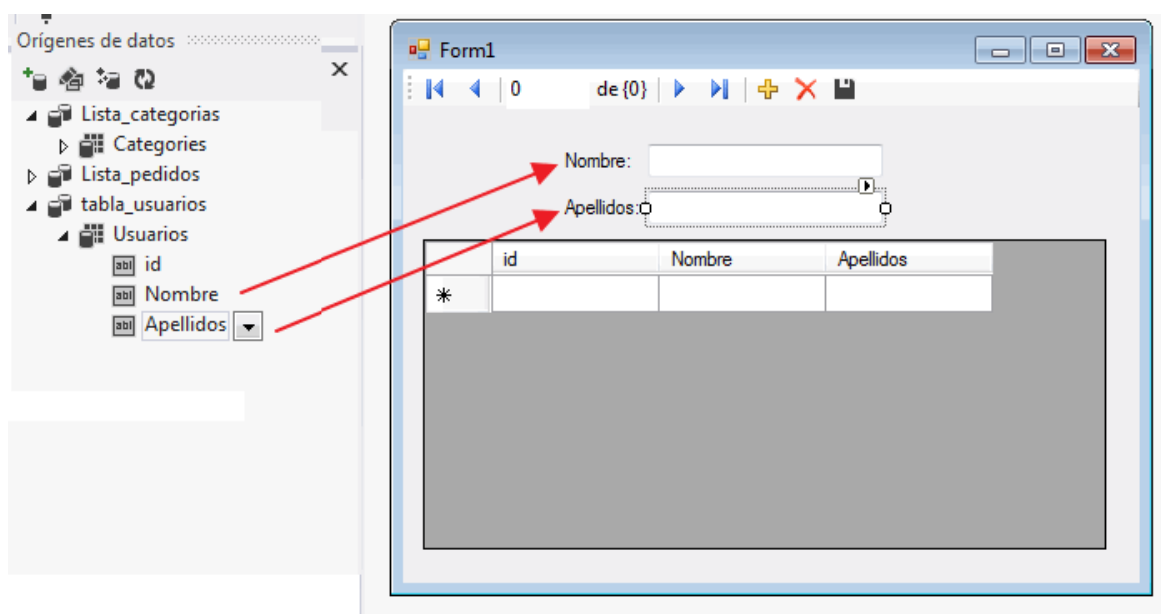
```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'TODO: esta línea de código carga datos en la tabla 'Tabla_usuarios.Usuarios' Puede moverl
    Me.UsuariosTableAdapter.Fill(Me.Tabla_usuarios.Usuarios)
    'TODO: esta línea de código carga datos en la tabla 'Tabla_usuarios.Usuarios' Puede moverl
    Me.UsuariosTableAdapter.Fill(Me.Tabla_usuarios.Usuarios)
    'TODO: esta línea de código carga datos en la tabla 'Tabla_usuarios.Usuarios' Puede moverl
    Me.UsuariosTableAdapter.Fill(Me.Tabla_usuarios.Usuarios)
End Sub

Private Sub UsuariosBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles U
    Me.Validate()
    Me.UsuariosBindingSource.EndEdit()
    Me.TableAdapterManager.UpdateAll(Me.Tabla_usuarios)
End Sub
```

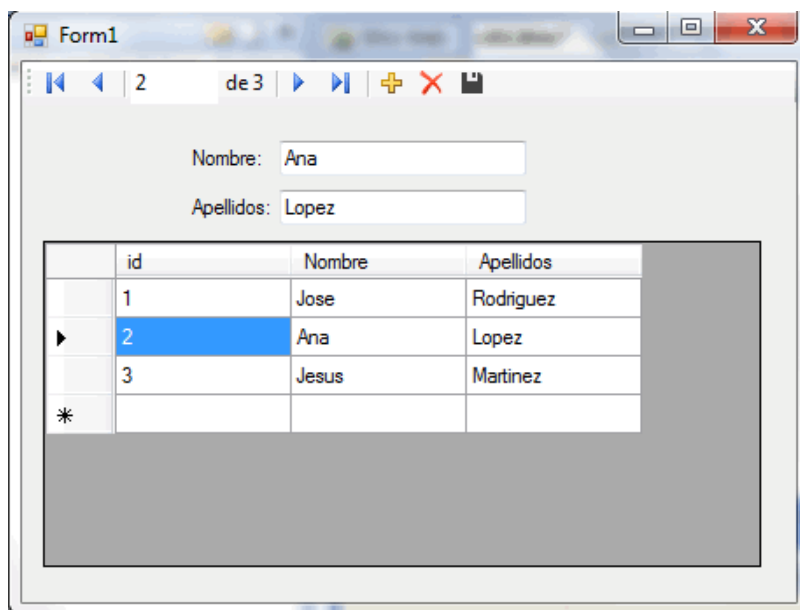
Un procedimiento para rellenar los datos (Load) y otro para almacenar los datos al pulsar el botón de guardar (UsuariosBindingNavigationSaveItem). Luego como vemos, los Dataset cumplen la definición de que están desconectados, podemos modificarlos, añadir, borrar pero hasta que no hagamos un "Update"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

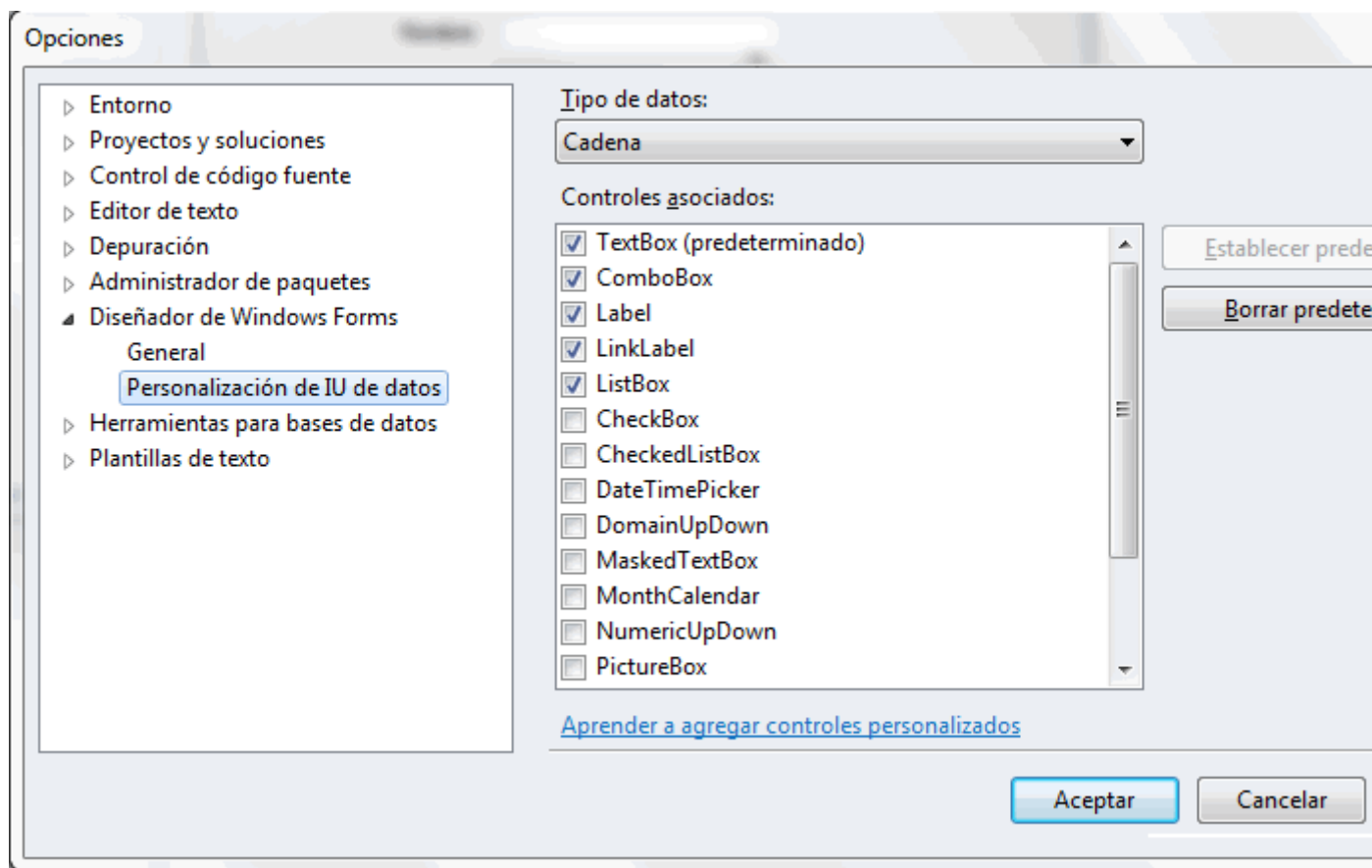
Veamos otro ejemplo. Sobre lo que ya tenemos, arrastra individualmente los dos campos de nombre y apellidos:



Así que si ejecutamos ahora:

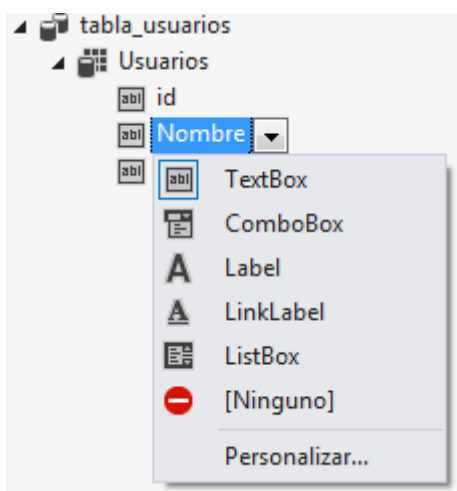


Vemos que todo está perfectamente sincronizado. Pero aún tenemos cosas más curiosas todavía, si nos vamos al menú de opciones del IDE tenemos:

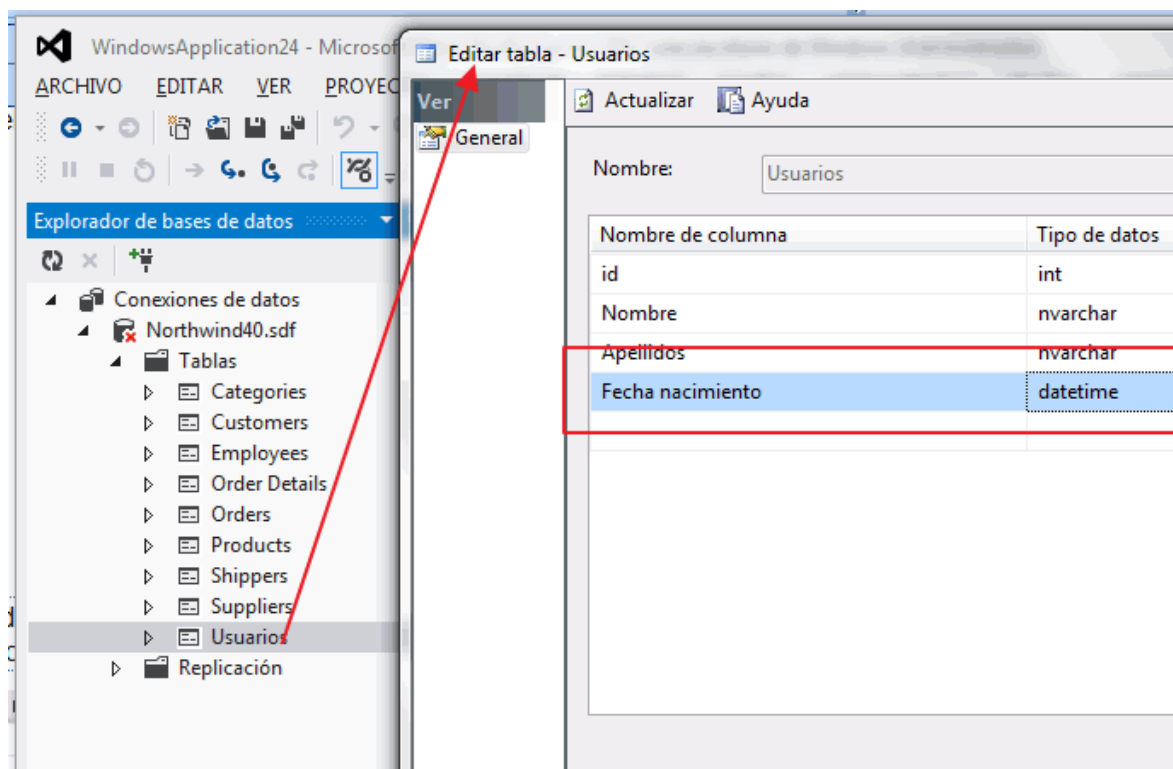


En esta pantalla podemos definir los controles que nos va a poner automáticamente dependiendo del tipo de datos que tengamos. Esto es, si en la base de datos el campo es de tipo fecha, al arrastrar el campo, lo podemos asociar con un control de tipo fecha, por ejemplo.

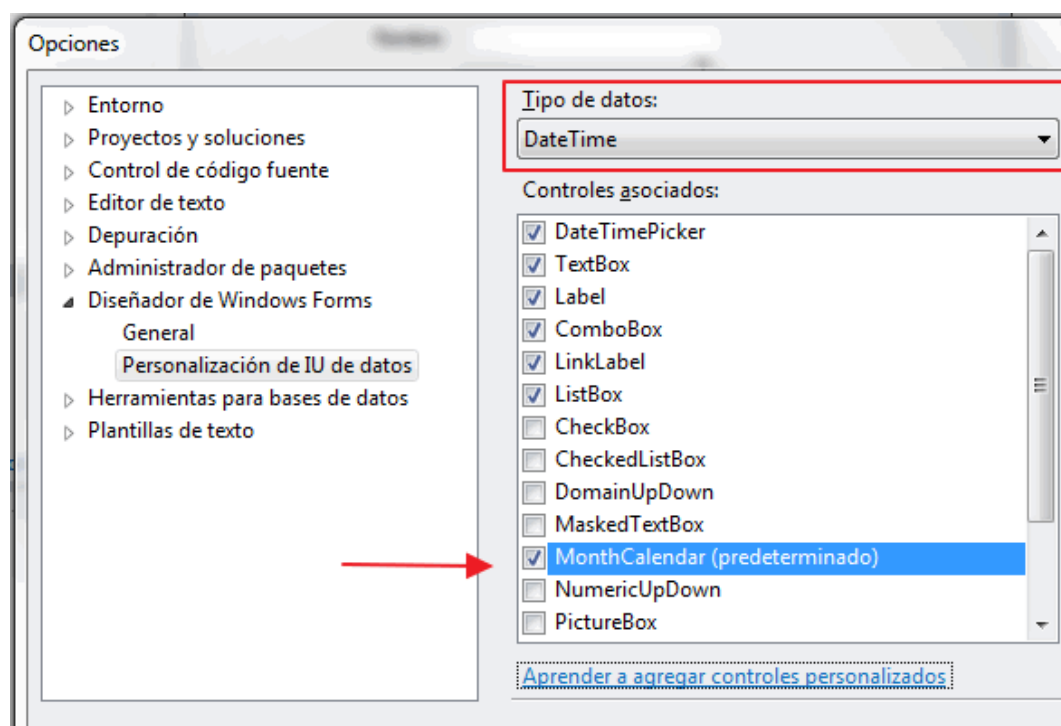
Antes de arrastrar los campos podemos indicarle el control que queremos utilizar para la edición, seleccionándolo de:



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:

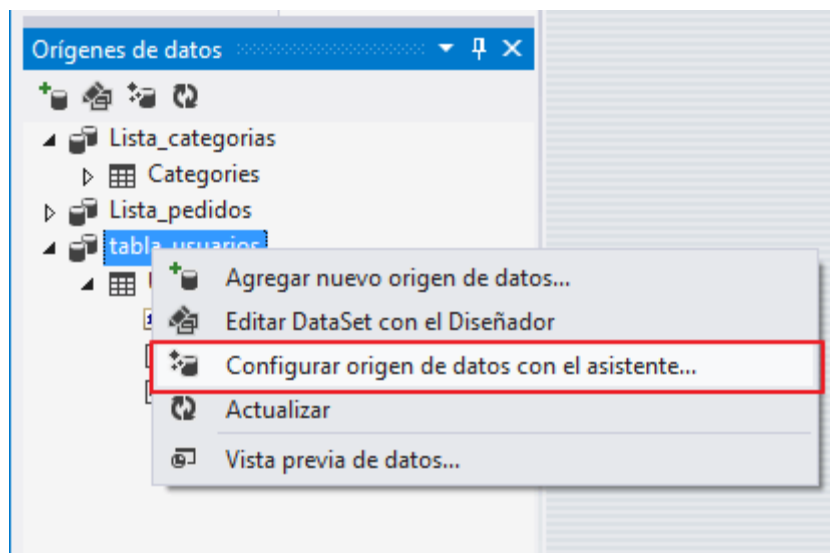


Ahora en las opciones de los campos, de vuelta a Visual Studio, veamos que tenemos para los campos de tipo fecha:

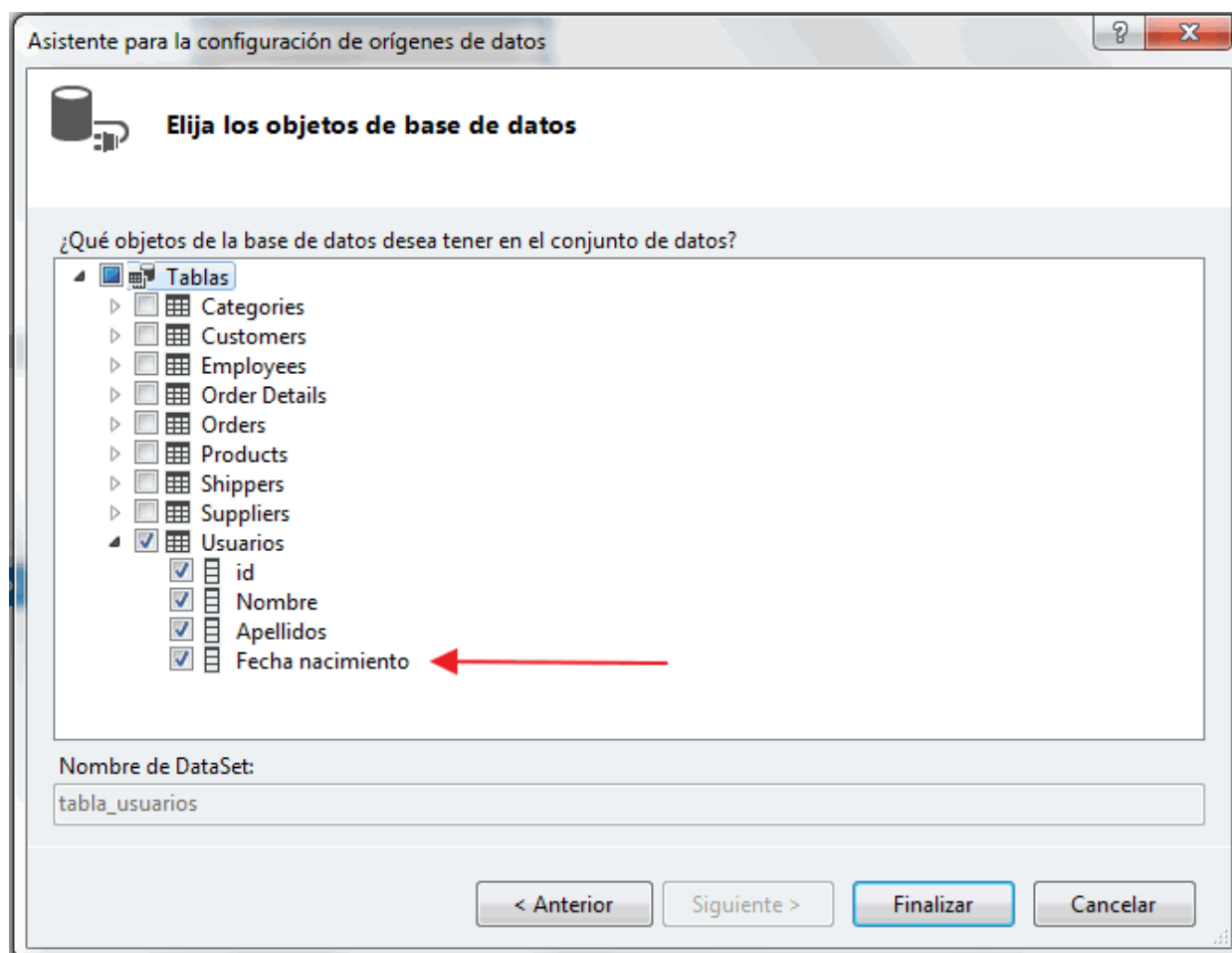


Bien, teníamos como predeterminado el "DateTimePicker" que es el de calendario desplegable. Es el mejor para estos casos pero le vamos a poner también el de la vista del mes completo y lo marcamos como predeterminado.

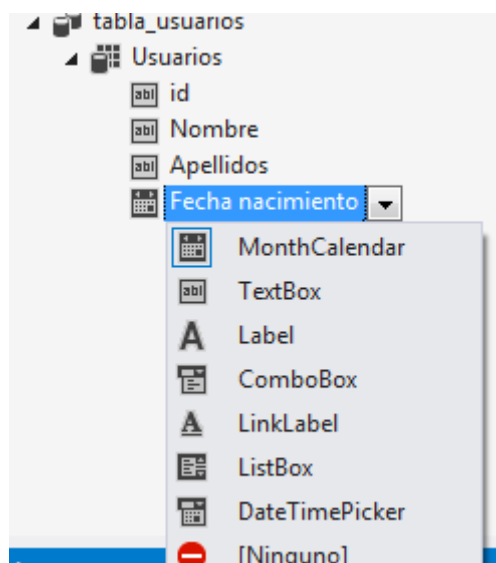
Como hemos añadido un campo a la tabla debemos actualizar el Dataset:



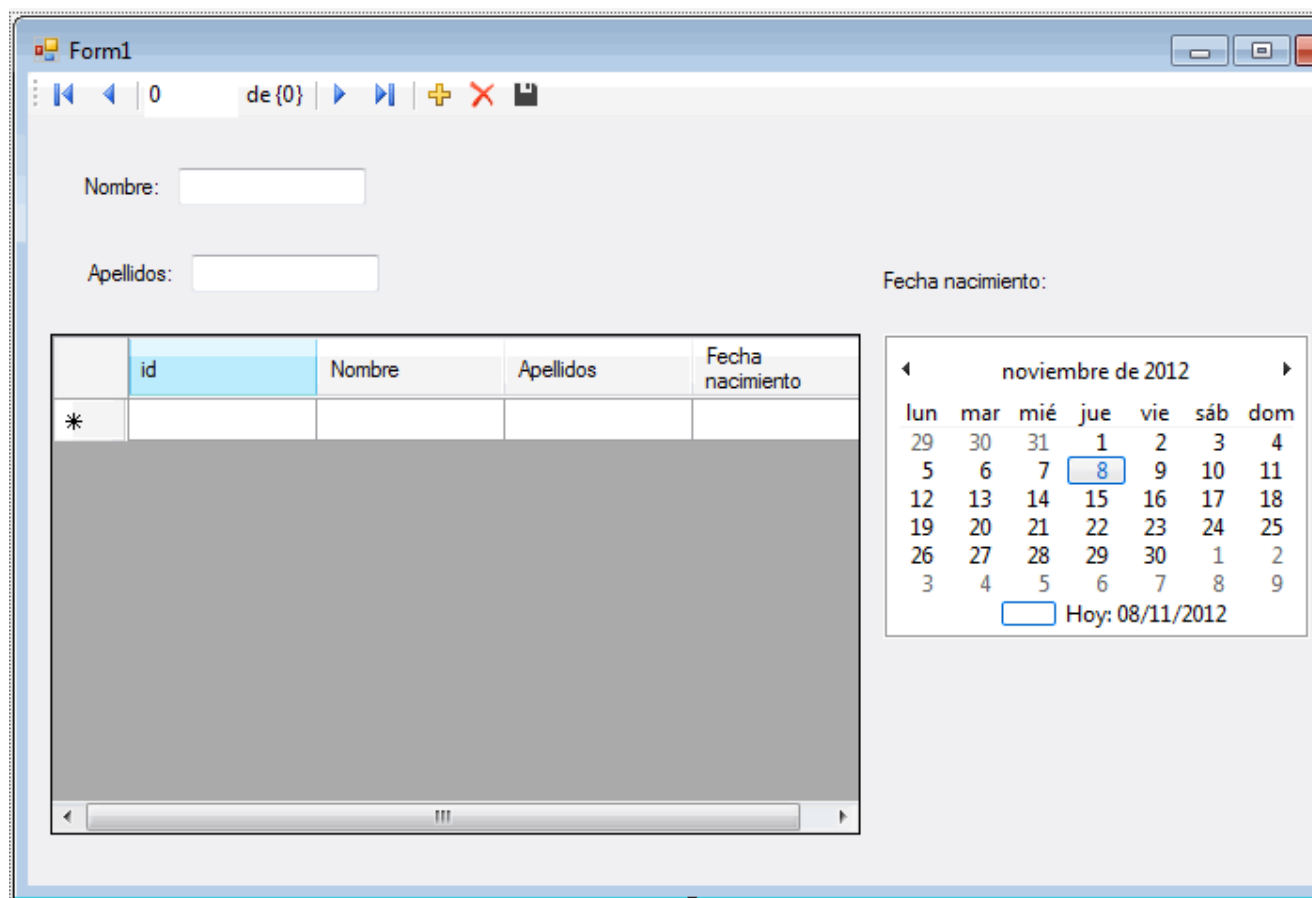
Expandimos y marcamos el nuevo campo para que lo incluya en el método "Fill" del "Dataset":



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

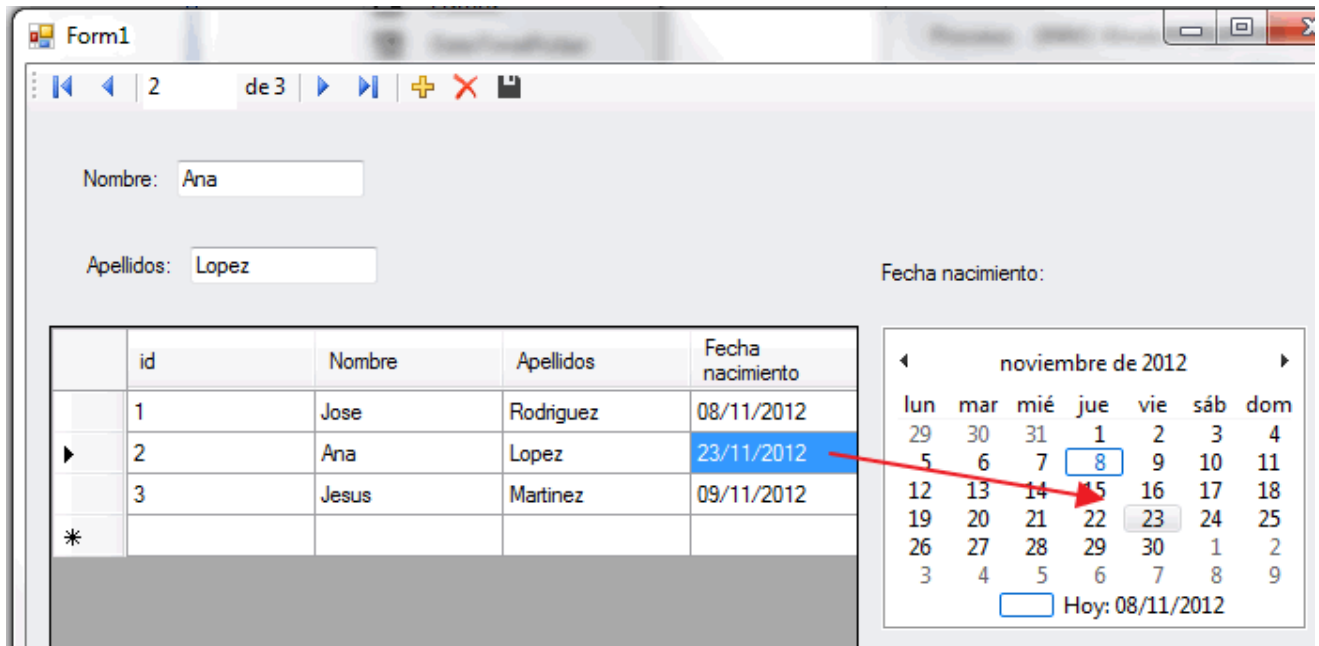


Hasta la posición que queramos. Por ejemplo:

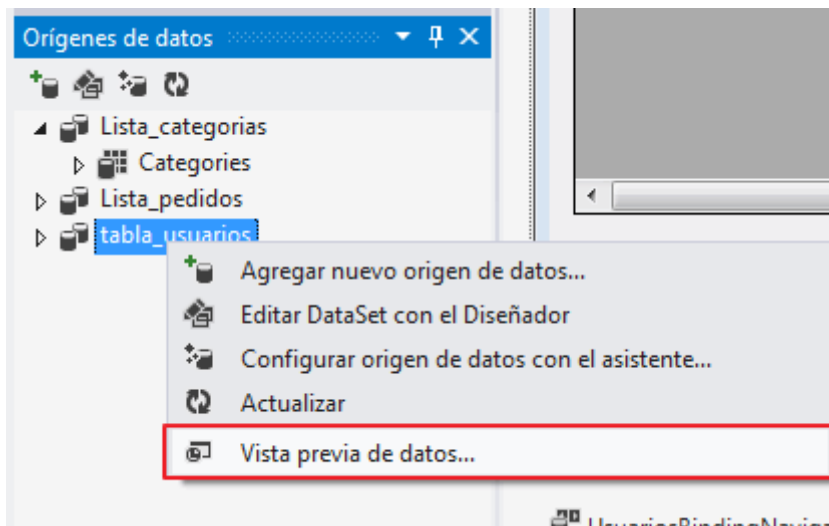


Como hemos variado los campos, lo mejor es volver a colocar los controles de un momento, eliminando también los controles ocultos de enlaces a datos. Ahora funcionará con el nuevo

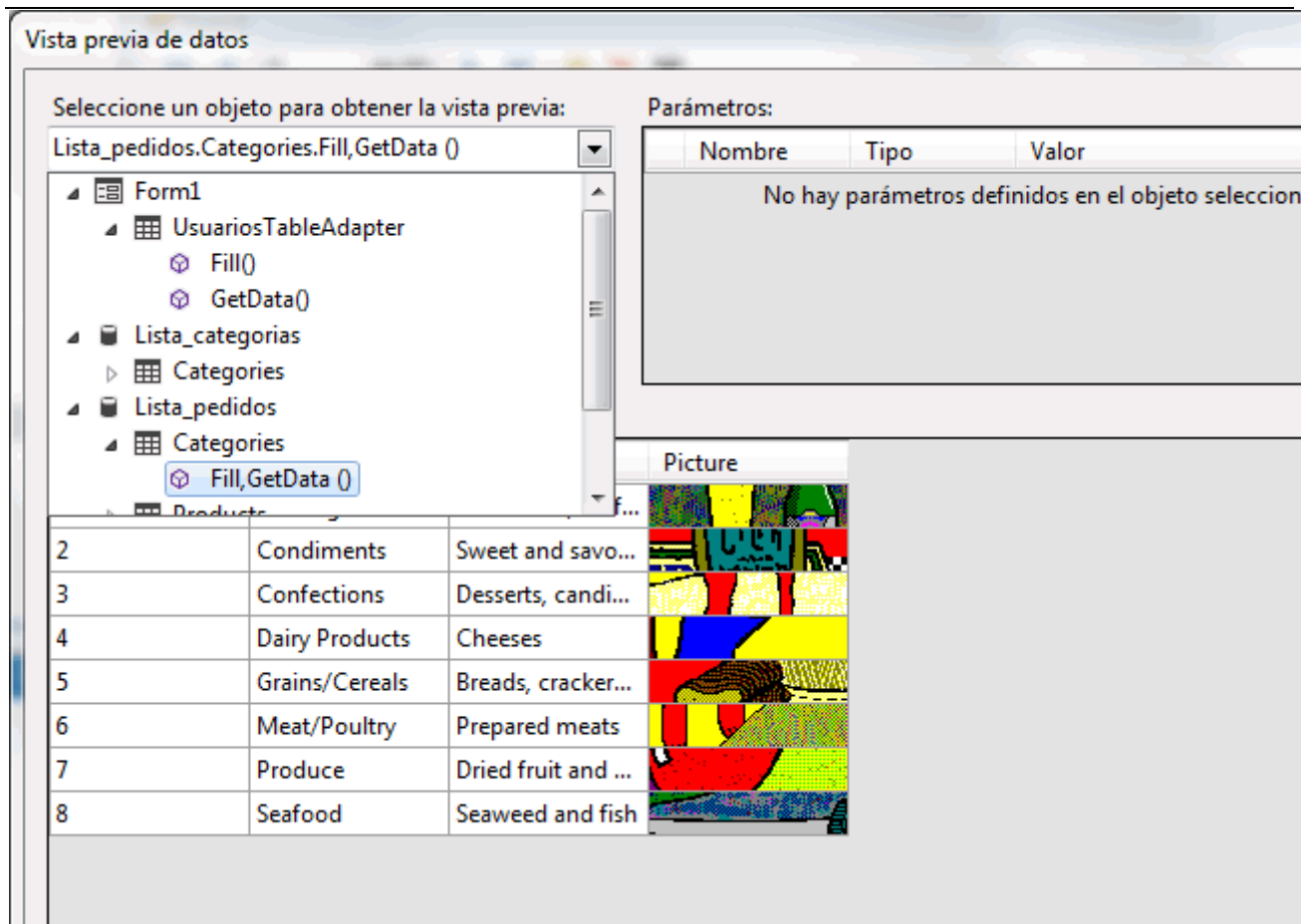
control y su tipo de datos de fecha totalmente operativo. Si seleccionamos una fecha en un registro y le damos al botón superior de grabar, se almacenarán los datos:



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:



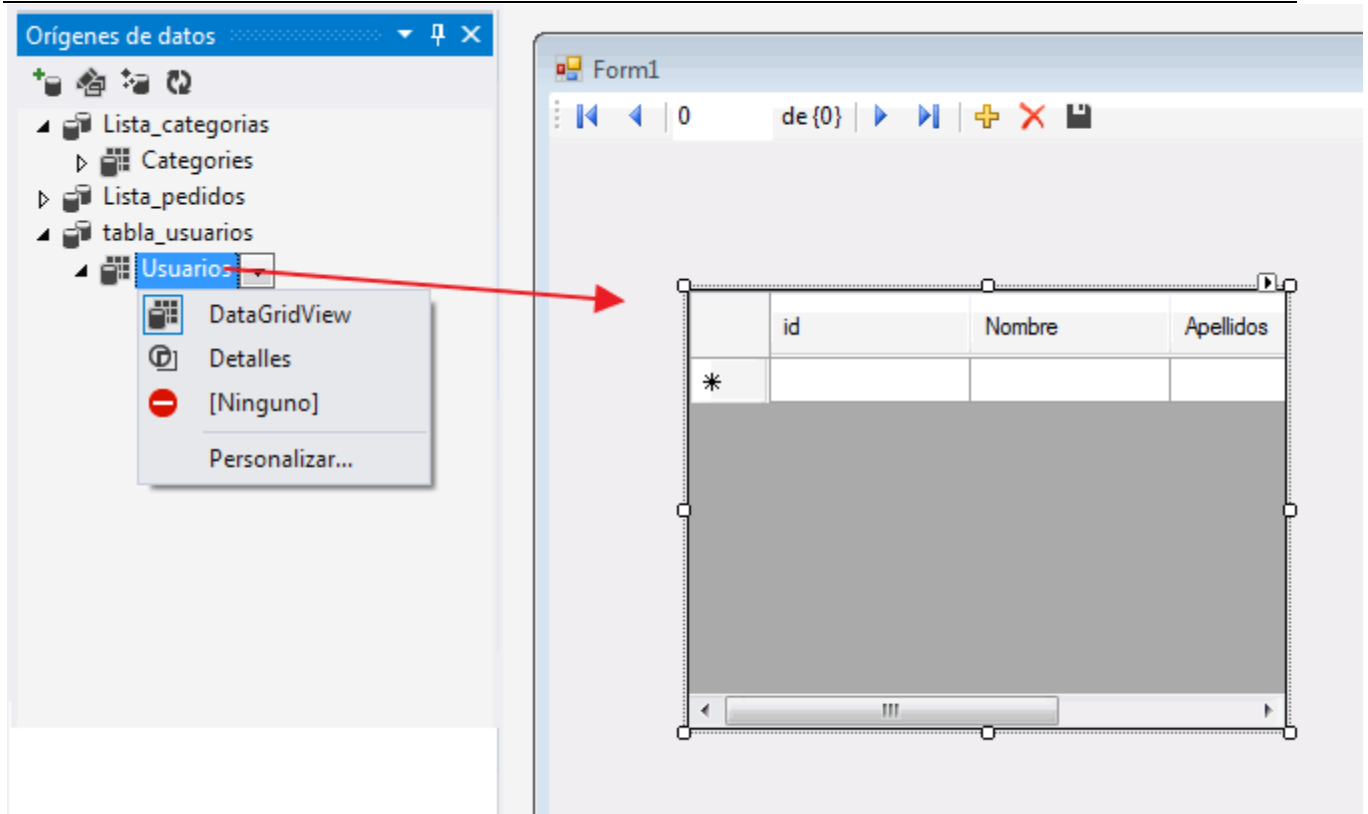
Se ejecuta el método Fill para rellenar esa cuadrícula de prueba. En ese ejemplo, la tabla de categorías tiene un campo gráfico, así que le asocia la vista previa con un "picturebox" para que muestre el contenido.

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:



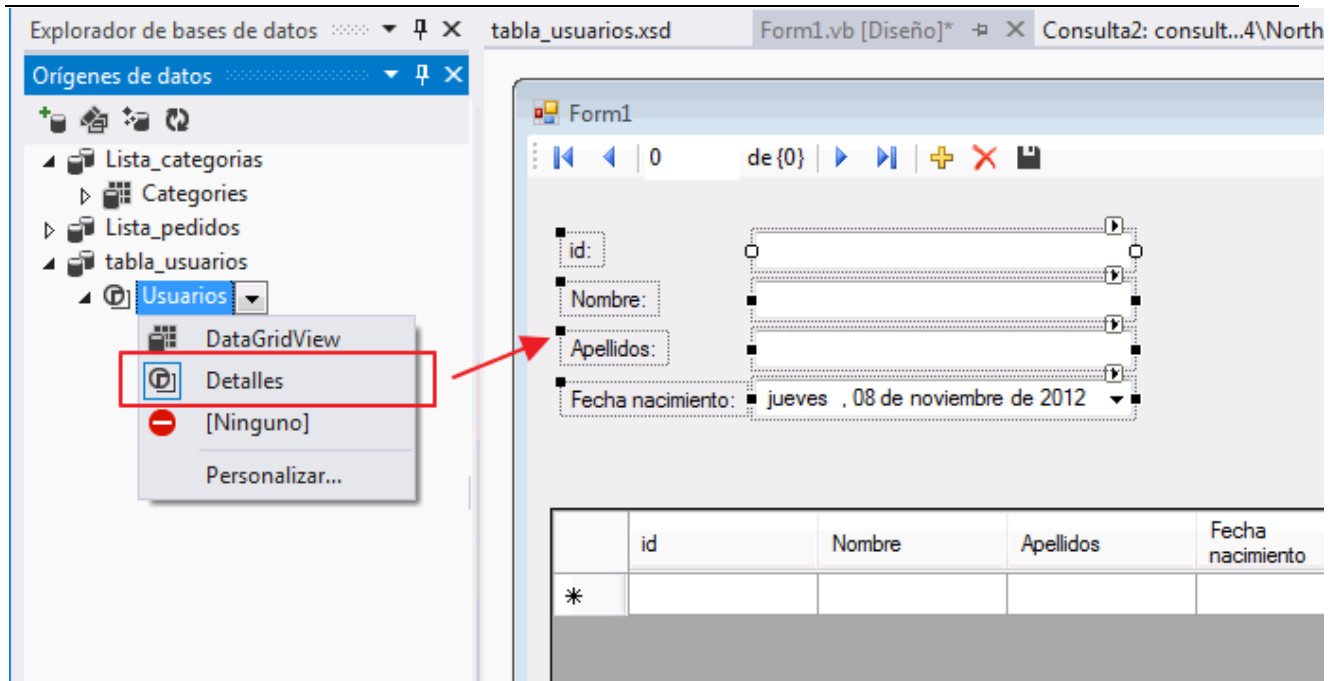
Y con las dos instrucciones, primero para cargar del servidor de base de datos el Dataset a nuestro equipo local:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'TODO: esta línea de código carga datos en la tabla 'NorthwindDataSet1.Users'
    Me.UsuariosTableAdapter.Fill(Me.NorthwindDataSet1.Usuarios)
End Sub
```

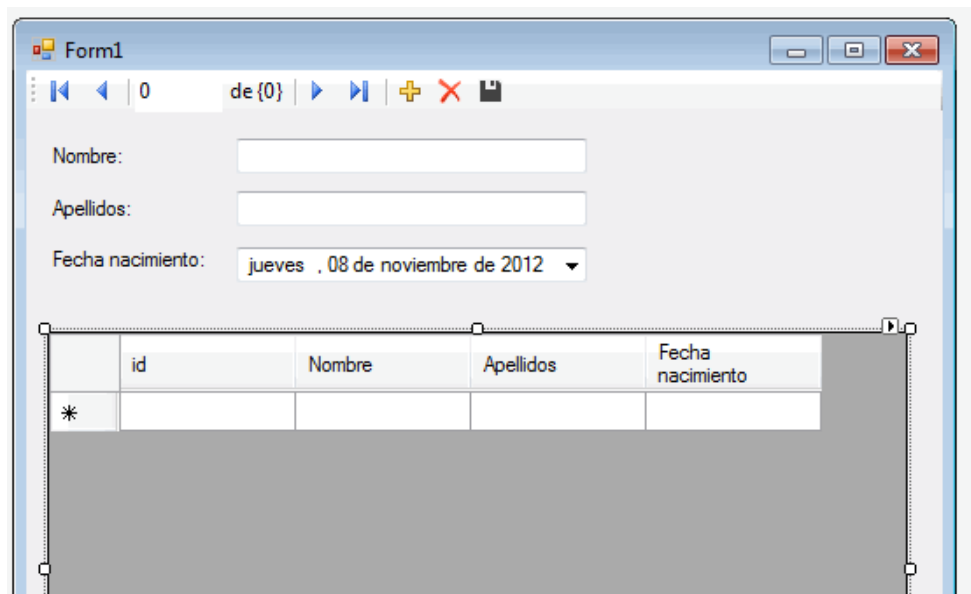
Y dentro del botón de grabar la actualización del Dataset local con el servidor:

```
Private Sub UsuariosBindingNavigatorSaveItem_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles UsuariosBindingNavigator.SaveItem.Click
    Me.Validate()
    Me.UsuariosBindingSource.EndEdit()
    Me.TableAdapterManager.UpdateAll(Me.NorthwindDataSet1)
End Sub
```

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:

	id	Nombre	Apellidos	Fecha nacimiento
▶	1	Jose	Rodriguez	02/11/2012 15:56
	2	Ana	Lopez	15/11/2012 15:56
	3	Jesus	Martinez	
*				

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

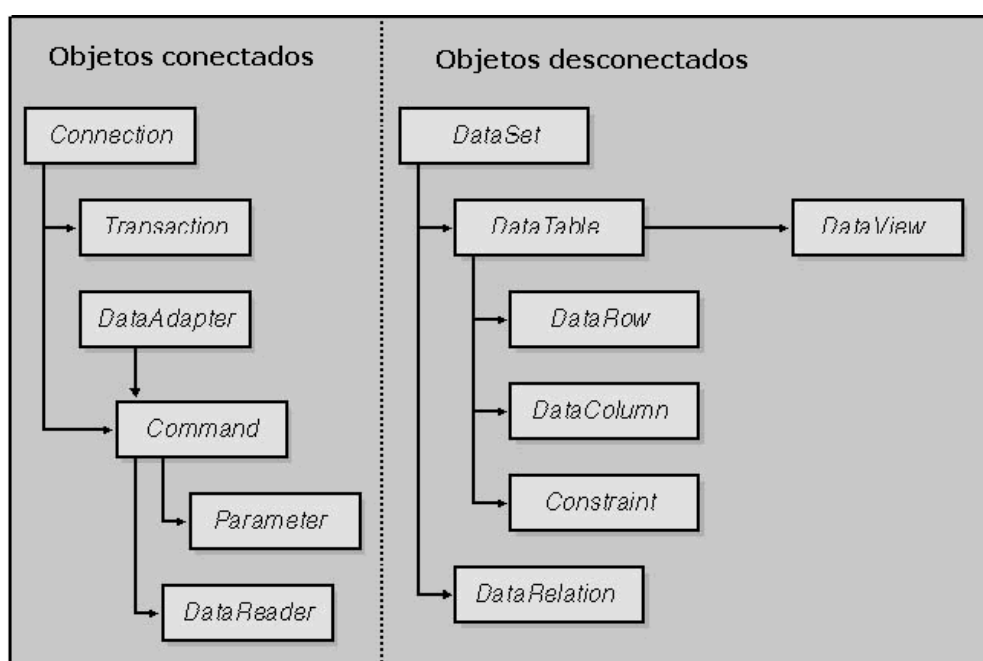
Es la forma con la que hemos trabajado hasta ahora. Después de muchos estudios se llegó a la conclusión de que mantener conexiones y consultas permanentemente abiertas afectaba mucho al rendimiento de la aplicación así que se diseñaron este tipo de objetos. Se conectan lo justo para actualizar los datos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

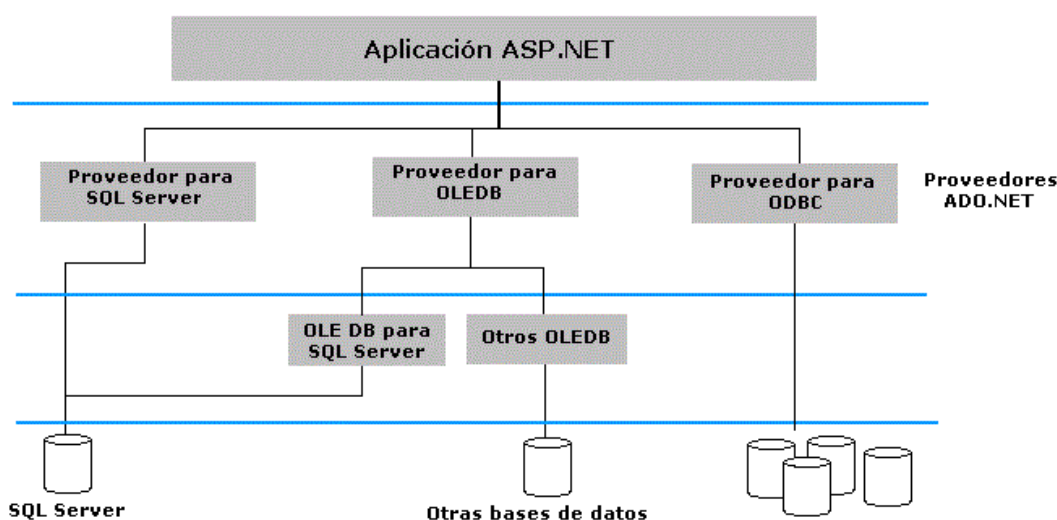
En este cuadro podemos ver los dos grupos de objetos:



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.

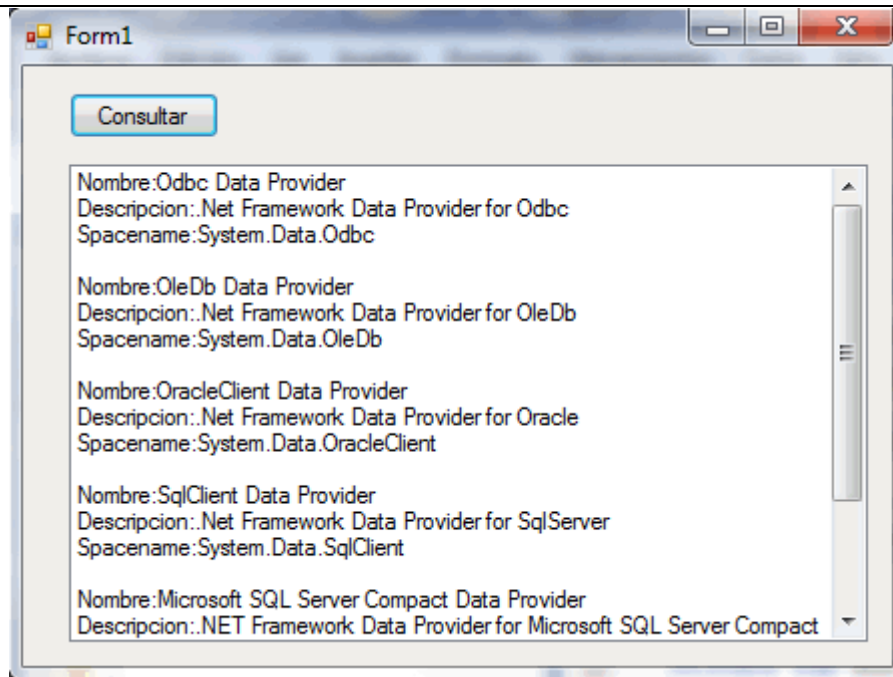
Creamos una aplicación nueva de Windows, ponemos un cuadro de lista y un botón. Importamos el espacio de nombres de "System.Data" y ponemos el siguiente código en el evento clic del botón:

```
Imports System.Data.Common
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim lista_clases As DataTable = DbProviderFactories.GetFactoryClasses

        For Each clase In lista_clases.Rows
            ListBox1.Items.Add("Nombre:" & clase("Name"))
            ListBox1.Items.Add("Descripcion:" & clase("Description"))
            ListBox1.Items.Add("Spacename:" & clase("InvariantName"))
            ListBox1.Items.Add(" ")
        Next
    End Sub
End Class
```

Lo ejecutamos y obtendremos:

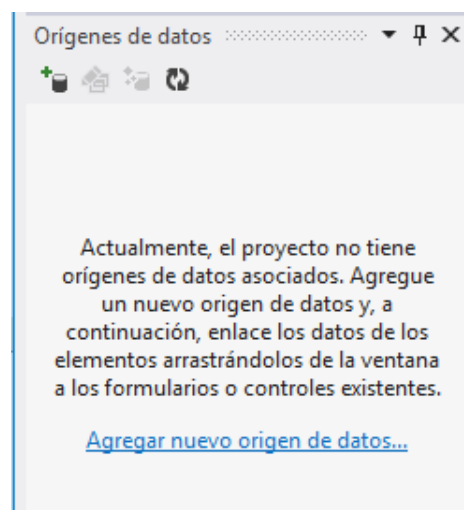


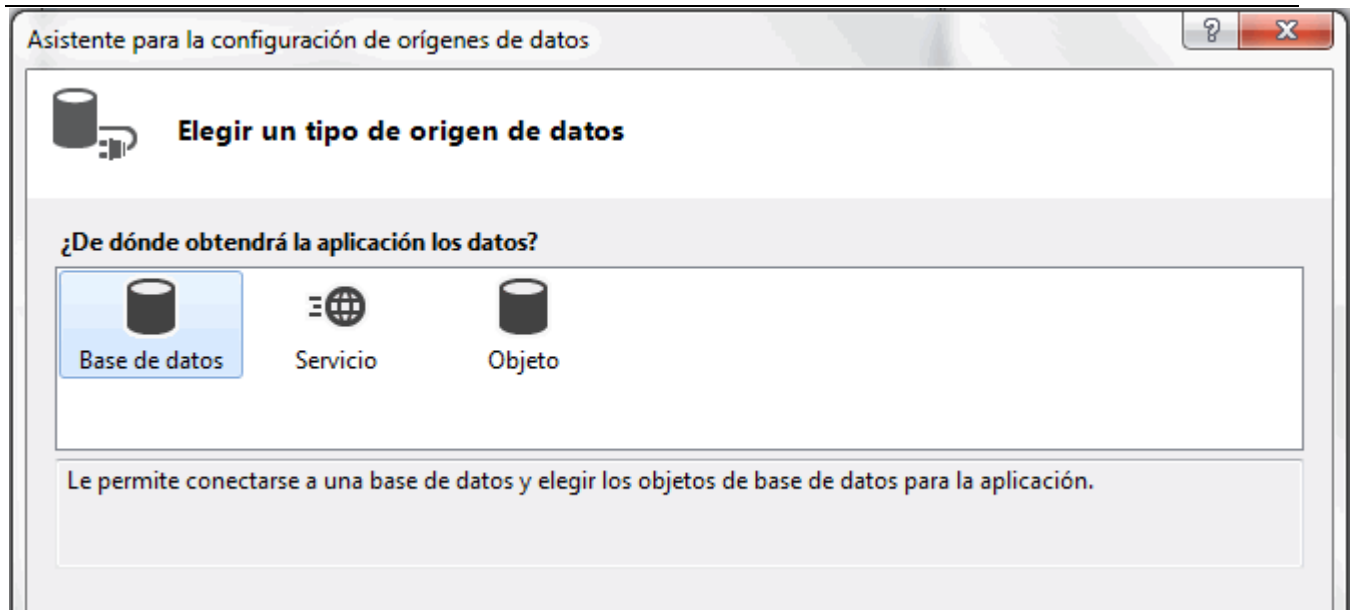
Una interesante lista de los proveedores. Si nos fijamos en los dos últimos, uno para un servidor SQL estándar y el otro es para el mini servidor SQL que nos ha instalado con Visual Basic. Vamos ya a conectarnos a bases de datos y a realizar sencillas operaciones.

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

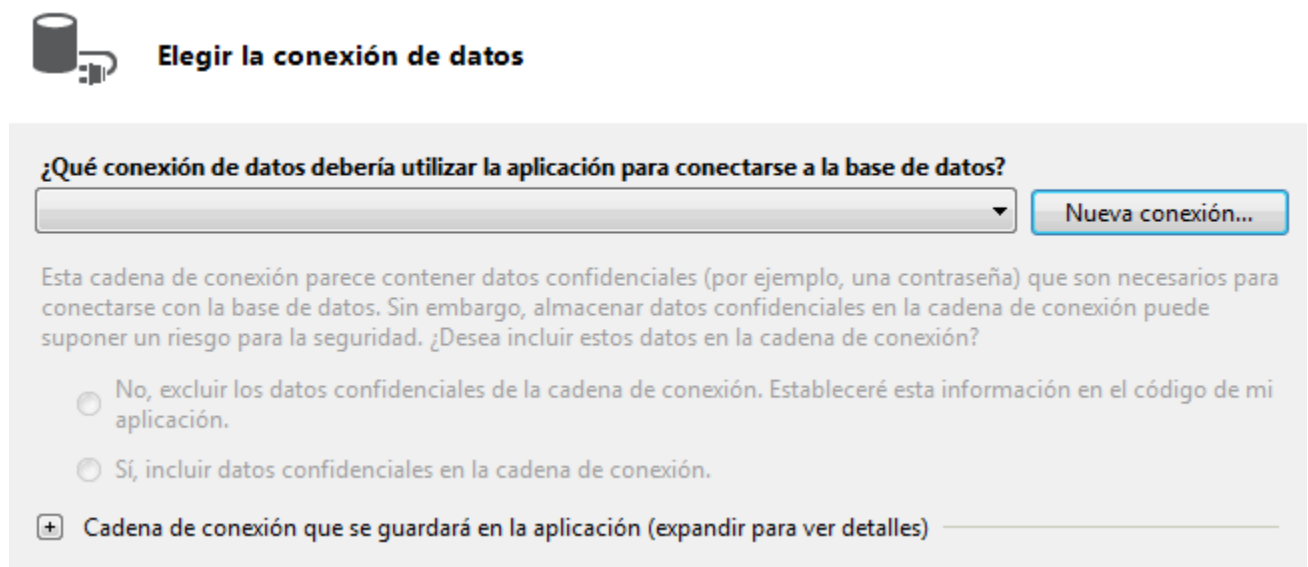
Ya sabemos con qué base de datos nos vamos a conectar y sabemos el proveedor que utilizaremos. Ahora haremos ejemplos con éstos. Para realizar la conexión tendremos que proporcionar una serie de datos que dependen del proveedor indicado. En SQL Server debemos indicar el servidor con su usuario y contraseña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

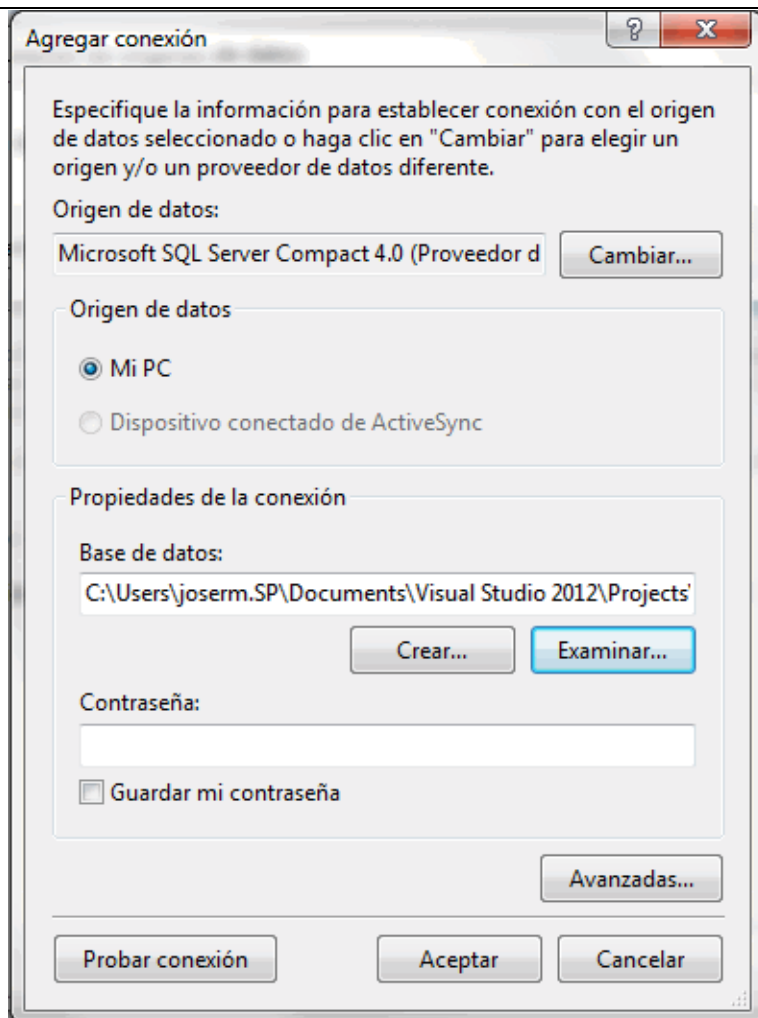




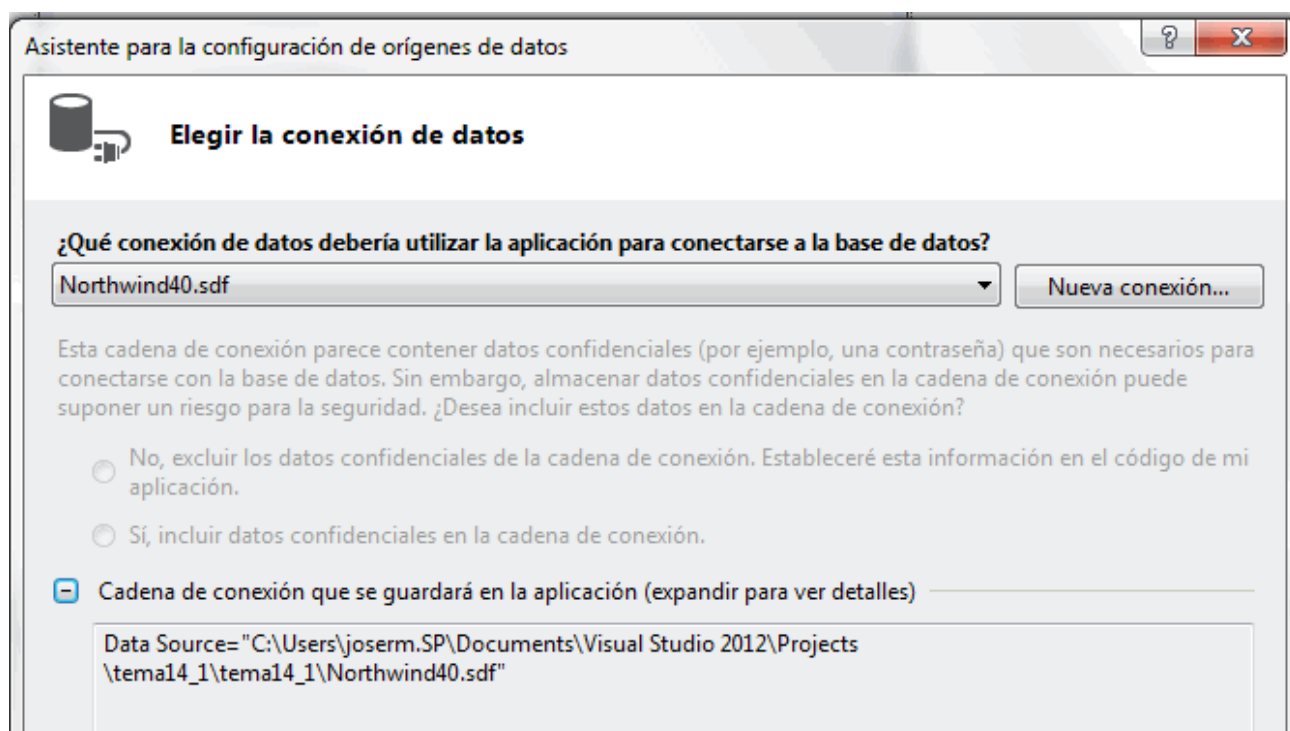
No tenemos ninguna conexión. Como es un proyecto nuevo no hay ninguna definida, así que pulsaremos en nueva conexión



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:

El almacenamiento de las cadenas de conexión del archivo de configuración de aplicación facilita el mantenimiento y la implementación. Para guardar la cadena de conexión en el archivo de configuración de la aplicación, escriba un nombre en el cuadro y, a continuación, haga clic en Siguiente.

¿Desea guardar la cadena de conexión en el archivo de configuración de la aplicación?

☒ Sí, guardar la conexión como:

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



Elija los objetos de base de datos

¿Qué objetos de la base de datos desea tener en el conjunto de datos?

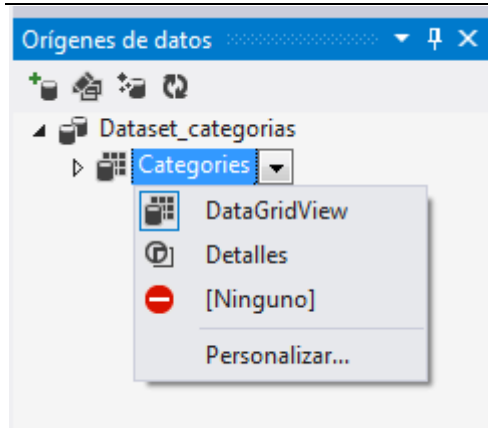
☒ Tablas

- ☒ Categories
- ☐ Customers
- ☐ Employees
- ☐ Order Details
- ☐ Orders
- ☐ Products
- ☐ Shippers
- ☐ Suppliers
- ☐ Usuarios

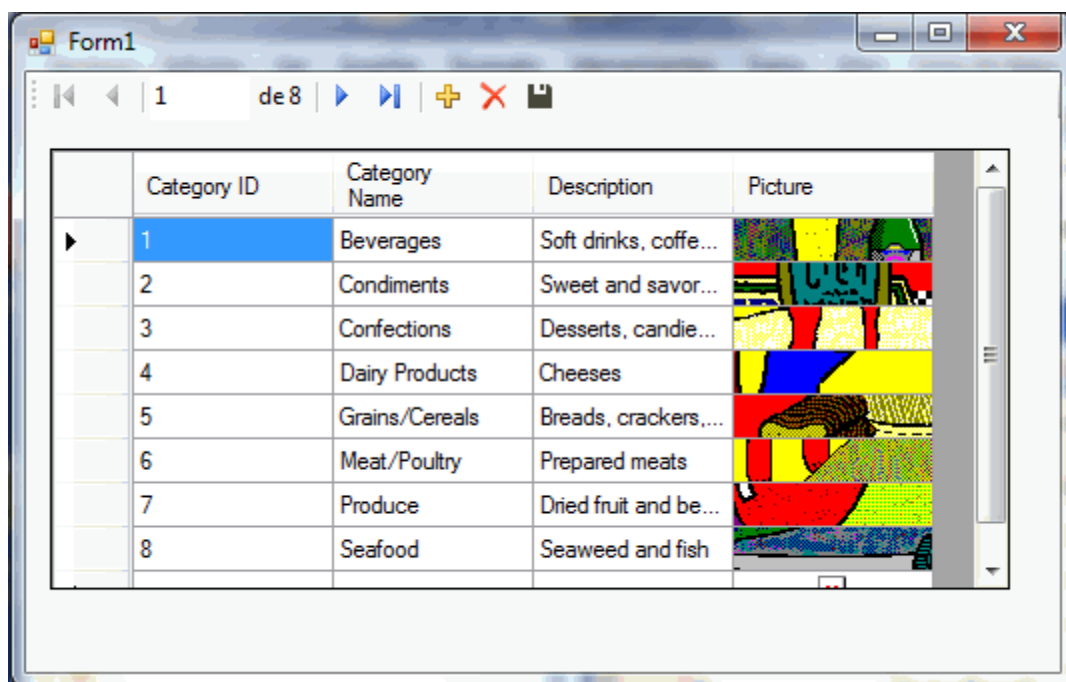
9.1. Consultar la base de datos con el IDE

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="WindowsApplication28.My.MySettings.Cadena_SQLCompact"
        connectionString="Data Source=|DataDirectory|\Northwind40.sdf"
        providerName="Microsoft.SqlServerCe.Client.4.0" />
</connectionStrings>
```

Y ahora veamos el código autogenerated en el programa:

```
Private Sub CategoriesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CategoriesBindingNavigator.SaveItem_Click
    Me.Validate()
    Me.CategoriesBindingSource.EndEdit()
    Me.TableAdapterManager.UpdateAll(Me.Dataset_categorias)
End Sub

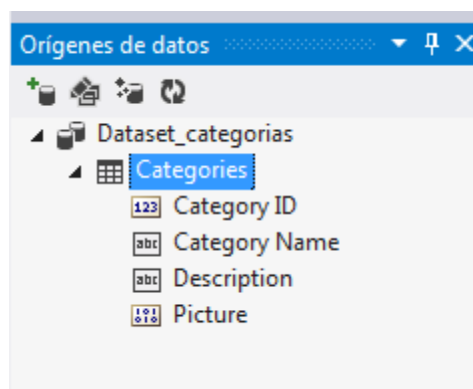
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'TODO: esta línea de código carga datos en la tabla 'Dataset_categorias.Categories' Puede moverse o eliminarse según corresponda.
    Me.CategoriesTableAdapter.Fill(Me.Dataset_categorias.Categories)
End Sub
```

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. Consultar la base de datos con el IDE y código.

Enseguida haremos un ejemplo íntegramente con código, sin la ayuda de arrastrar controles. Pero antes vamos a realizar una consulta por código pero utilizando los objetos que nos ha añadido antes el IDE. Nos puede ayudar un poco para familiarizarnos con los ejemplos. Creamos una aplicación Windows y le añadimos el mismo origen de datos de antes:



Hasta aquí todo igual, ahora vamos a realizar lo demás por código. En el evento Load del formulario vamos a escribir esto, donde leeremos la tabla y mostraremos un mensaje con el primer valor de la tabla:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'Creamos un dataset
    Dim test_ds As Dataset_categorias = New Dataset_categorias

    'El adaptador
    Dim adaptador As Dataset_categoriasTableAdapters.CategoriesTableAdapter = _
    New Dataset_categoriasTableAdapters.CategoriesTableAdapter

    'y lo rellenamos con los datos
    adaptador.Fill(test_ds.Categories)

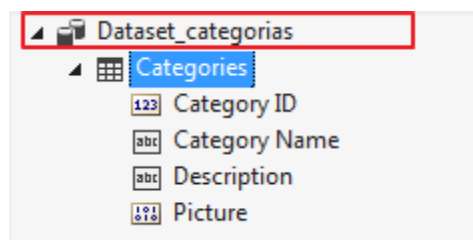
    'Preparamos para leer la primer fila:
    Dim fila As Dataset_categorias.CategoriesRow = _
    CType(test_ds.Categories.Rows(0), Dataset_categorias.CategoriesRow)
    MessageBox.Show(fila.Category_Name)

End Sub
```

Ahora vamos a ver los pasos que hemos realizado porque es una forma un poco compleja de trabajar que luego iremos simplificando:

1. Primero creamos una instancia de Dataset_categorias llamada test_ds. Dataset_categorias. Lo entiende porque al crear la conexión, crea un objeto de este tipo:

```
'Creamos un dataset
Dim test_ds As Dataset_Categorias = New Dataset_Categorias
```



2. El segundo paso es crear una instancia del objeto adaptador autogenerado por el IDE. Así que tenemos un adaptador ya creado para esta conexión y lo que vamos a hacer es crear un nuevo adaptador para este objeto.

```
'El adaptador
Dim adaptador As
Dataset_CategoriasTableAdapters.CategoriesTableAdapter = _
New Dataset_CategoriasTableAdapters.CategoriesTableAdapter
```

3. El tercer paso es utilizar el adaptador para rellenar el dataset que hemos creado.


```
'y lo rellenamos con los datos  
adaptador.Fill(test_ds.Categories)
```

4. El último paso y ya con los datos en el Dataset es extraerlos, donde estamos consultado la columna del nombre de la categoría:

```
'Preparamos para leer las filas:  
Dim fila As Dataset_Categorias.CategoriesRow = _  
CType (test_ds.Categories.Rows(0), Dataset_Categorias.CategoriesRow)  
MessageBox.Show(fila.Category_Name)
```

Esta forma de trabajar es más flexible que la anterior ya que podemos manejar los objetos de una forma más fácil. Aun así sigue siendo algo complicado porque los objetos que nos ha creado no son muy intuitivos y, la verdad, es que se aleja un poco de la facilidad que siempre hemos dicho que tiene ADO.NET.

Lo más fácil va a ser hacerlo todo manualmente ya que seguiremos de una forma más secuencial y sencilla los pasos de: conectar, consultar, crear conjunto de resultados y mostrar resultado. Eso es lo que veremos en el tercer ejemplo.

9.3. Realizar una consulta con el código

El último ejemplo es 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:

```
Imports System.Data.SqlClient
```

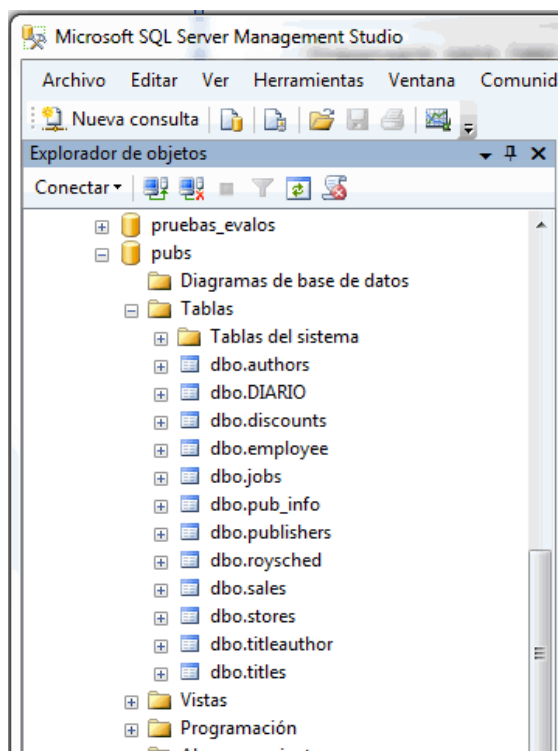
Si la base de datos es de Microsoft Access importaremos:

```
Imports System.Data.OleDb
```

Esta vez vamos a conectarnos con un servidor SQL Server. Se trata de la versión SQL Server Express que puedes descargar desde la página de Microsoft. Cuando lo hagas asegúrate de descargar 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.

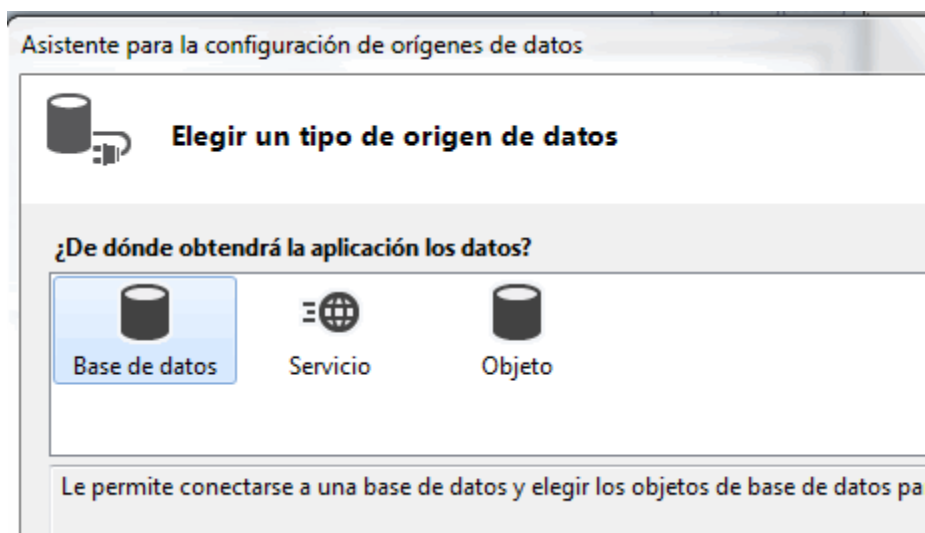
A continuación se realiza el ejemplo con la bbdd pubs. Nosotros trabajaremos en clase con la bbdd Northwind, que importamos al principio del tema.

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:

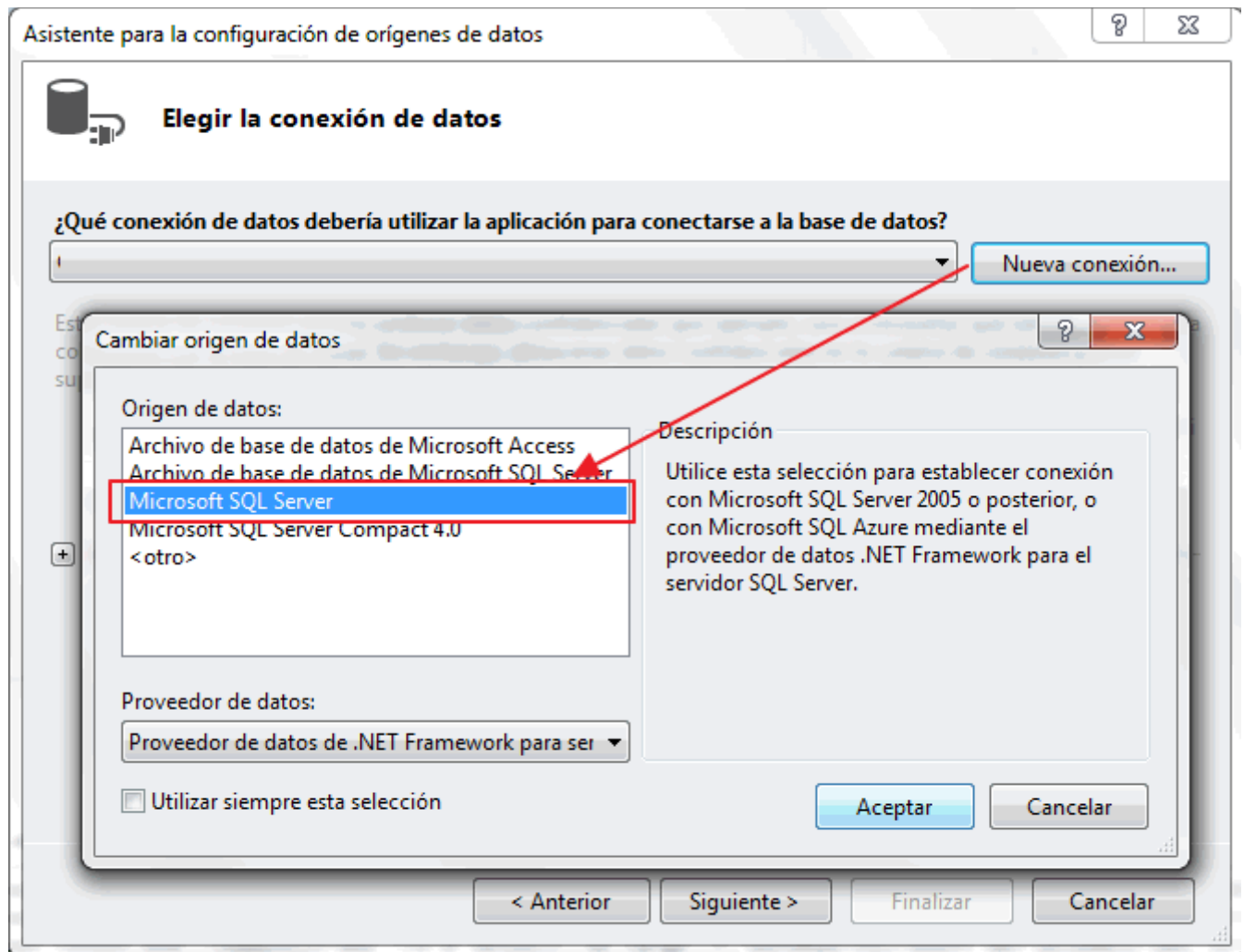


En la tabla "Authors" hay una lista de autores de libros, donde queremos extraer los campos de nombre y apellidos, pero todo de forma manual. Vamos con ello.

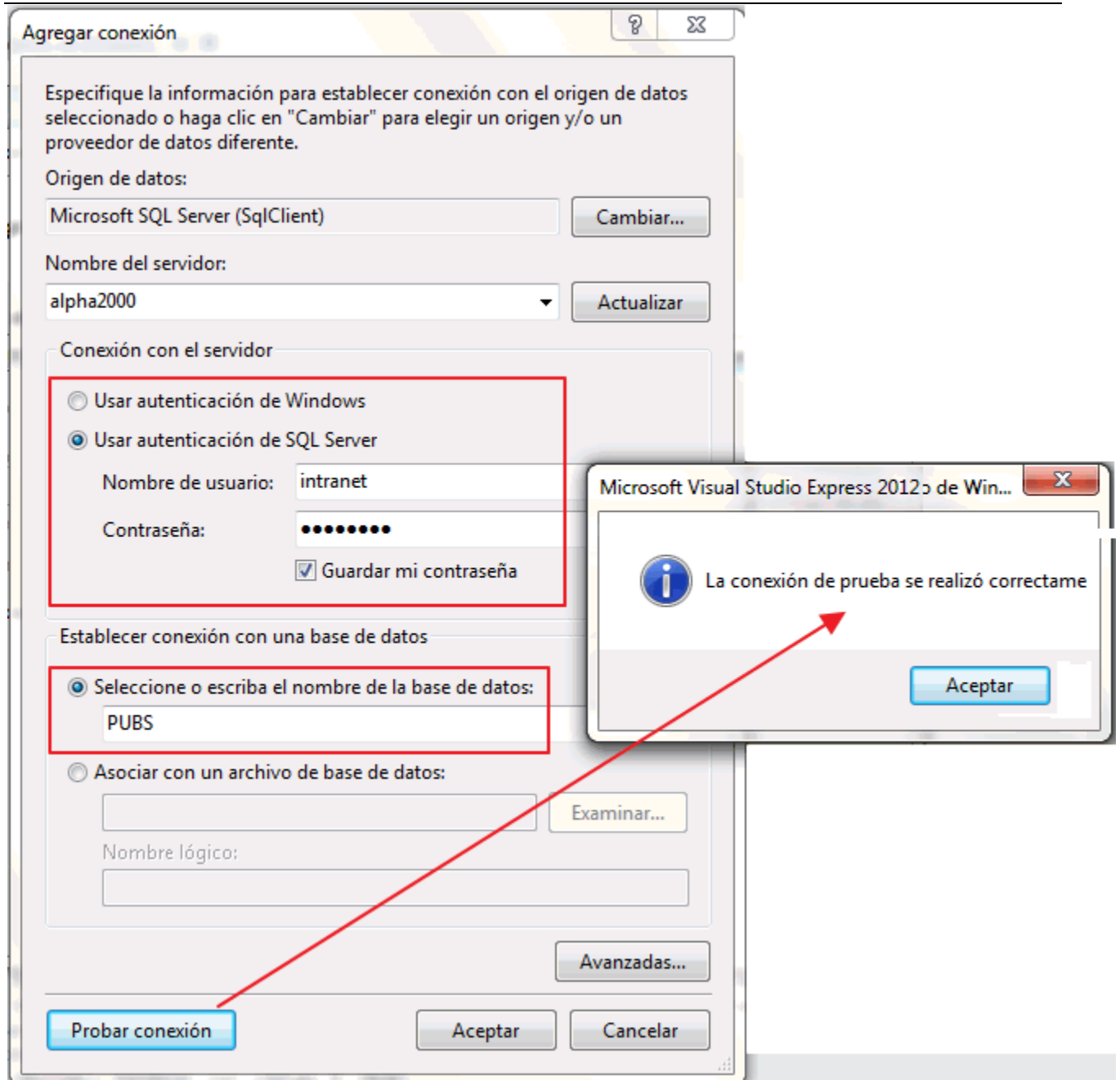
Creamos una aplicación Windows y esta vez no incluiremos ninguna conexión de base de datos, lo vamos a definir todo desde el código. Para empezar vamos a definir la cadena de conexión. No vamos a utilizar ningún objeto ni nada automático, haremos lo mismo que antes pero de forma manual. Para empezar 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:



Elegir la conexión de datos

¿Qué conexión de datos debería utilizar la aplicación para conectarse a la base de datos?

alpha2000.PUBS.dbo1 Nueva conexión...

Esta cadena de conexión parece contener datos confidenciales (por ejemplo, una contraseña) que son necesarios para conectarse con la base de datos. Sin embargo, almacenar datos confidenciales en la cadena de conexión puede suponer un riesgo para la seguridad. ¿Desea incluir estos datos en la cadena de conexión?

☐ No, excluir los datos confidenciales de la cadena de conexión. Estableceré esta información en el código de mi aplicación.

☐ Sí, incluir datos confidenciales en la cadena de conexión.

☒ Cadena de conexión que se guardará en la aplicación (expandir para ver detalles)

Data Source=alpha2000;Initial Catalog=PUBS;Persist Security Info=True;User ID=intranet

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

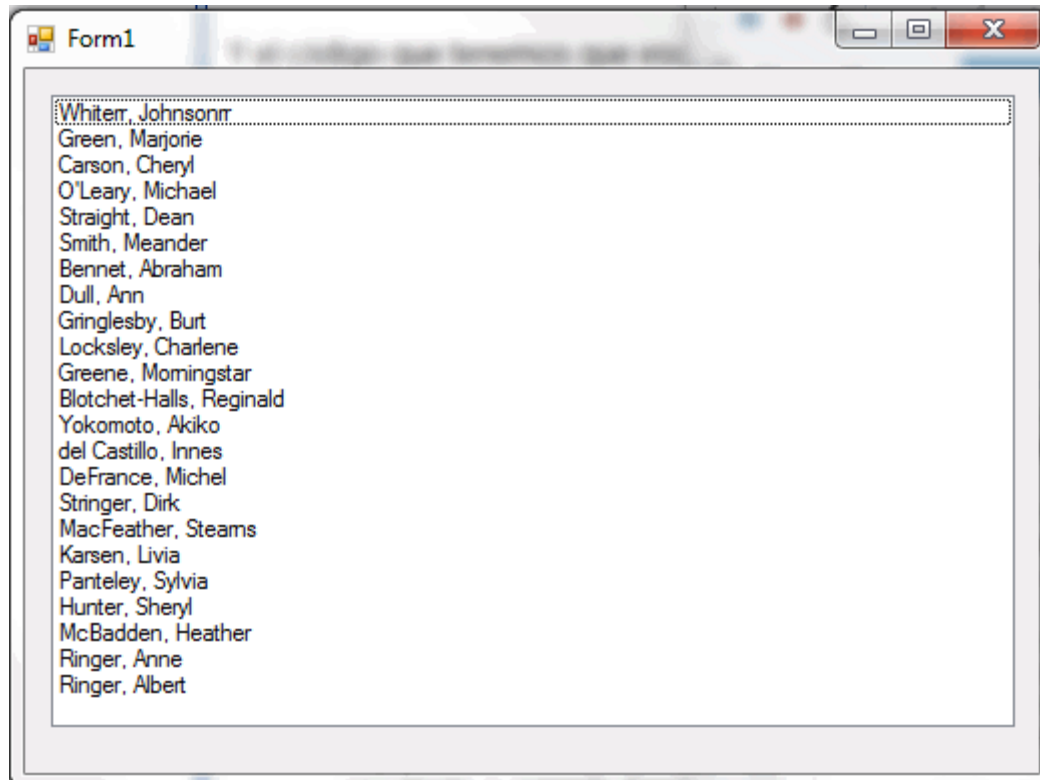
```
Public Class Form1
```

```
Private cadena_conexion = _  
"Data Source=Alpha2000;Initial Catalog=pubs;Persist Security Info=True;User ID=intranet; Password=intranet"
```

Y el código que tenemos que escribir es:

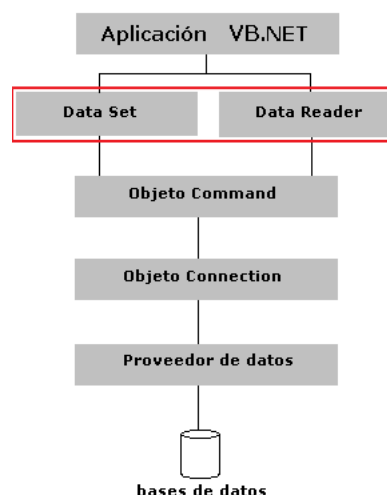
```
Imports System.Data.SqlClient  
Public Class Form1  
  
    Private cadena_conexion = _  
        "Data Source=Alpha2000;Initial Catalog=pubs;Persist Security Info=True;User ID=intranet"  
  
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load  
        'Creamos un objeto conexion  
        Dim conecta As SqlConnection = New SqlConnection(cadena_conexion)  
  
        'Creamos un comando para realizar la consulta  
        Dim comando As SqlCommand = conecta.CreateCommand  
        comando.CommandText = "Select * from Authors"  
  
        'Abrimos la conexión  
        conecta.Open()  
        'Creamos un DataReader para almacenar el resultado de la consulta  
        Dim resultado As SqlDataReader  
  
        'Ejecutamos la consulta  
        resultado = comando.ExecuteReader  
  
        'Hacemos un bucle por el datareader para escribirlo  
        Do While resultado.Read  
            ListBox1.Items.Add(resultado("au_lname") & ", " & resultado("au_fname"))  
        Loop  
  
        conecta.Close()  
    End Sub  
End Class
```

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 y apellidos de la tabla de autores:



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. Tanto hablar de los famosos adaptadores y aquí no aparece por ningún lado. Pues no 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":

```
Private cadena_conexion = _  
"Data Source=Alpha2000;Initial Catalog=pubs;Persist Security Info=True;User ID=intr;
```

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load  
    'Creamos un objeto conexion  
    Dim conecta As SqlConnection = New SqlConnection(cadena_conexion)
```

Nota: La cadena de conexión para la base de datos de Access sería:

```
cadena_conexion="Provider=Microsoft.Jet.OLEDB.4.0;Data Source="C:\Northwind.mdb""
```

Para la conexión con Access necesitaremos objetos OleDbConnection, OleDbCommand, OleDbDataReader

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

```
'Creamos un comando para realizar la consulta  
Dim comando As SqlCommand = conecta.CreateCommand  
comando.CommandText = "Select * from Authors"
```

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  
conecta.Open()  
'Creamos un DataReader para almacenar el resultado de la consulta  
Dim resultado As SqlDataReader  
  
'Ejecutamos la consulta  
resultado = comando.ExecuteReader
```

Las instrucciones con la ejecución del 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 datareades para escribirlo  
Do While resultado.Read  
    ListBox1.Items.Add(resultado("au_lname") & ", " & resultado("au_fname"))  
Loop
```

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:

```
Dim conexion as New SqlConnection
Dim cadena_conexion as String="Data Source=nombre_servidor;" & _
    "Initial Catalog=PUBS; ... "

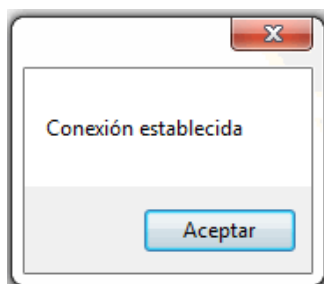
conexion.ConnectionString= cadena_conexion
```

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:

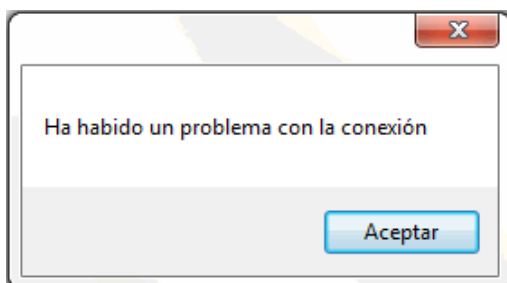
```
'Creamos un objeto conexion
Dim conecta As SqlConnection = New SqlConnection(cadena_conexion)

Try
    'Abrimos la conexión
    conecta.Open()
    If conecta.State = ConnectionState.Open Then
        MessageBox.Show("Conexión establecida")
    End If
Catch
    If conecta.State <> ConnectionState.Open Then
        MessageBox.Show("Ha habido un problema con la conexión")
    End If
Finally
    If conecta.State = ConnectionState.Open Then
        conecta.Close()
    End If
    conecta.Dispose()
End Try
```

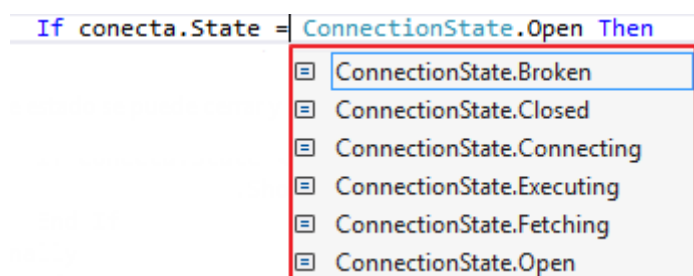

Hemos hecho una ejecución controlada de la conexión utilizando un controlador de errores "Try...Catch" que deberíamos utilizar siempre porque no sabemos si la base de datos destino está operativa. En el caso de que haya sido correcta la apertura obtendremos:



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 Sql Server.

```
Dim cadena_conexion as String="Data Source=localhost\sqlexpress;" & _  
    "Initial Catalog=pubs; Integrated Security=SSPI"
```

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 SSPI indica que 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=SSPI. 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:

```
Dim cadena_conexion As SqlConnectionStringBuilder = _
    New SqlConnectionStringBuilder()

cadena_conexion.DataSource = "localhost"
cadena_conexion.InitialCatalog = "pubs"
cadena_conexion.IntegratedSecurity = True

Using conexion As SqlConnection = _
    New SqlConnection(cadena_conexion.ToString())|
    Try
        conexion.Open()
        If conexion.State = ConnectionState.Open Then
            MessageBox.Show("Conexión abierta correctamente")
        End If
    Catch
```

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:

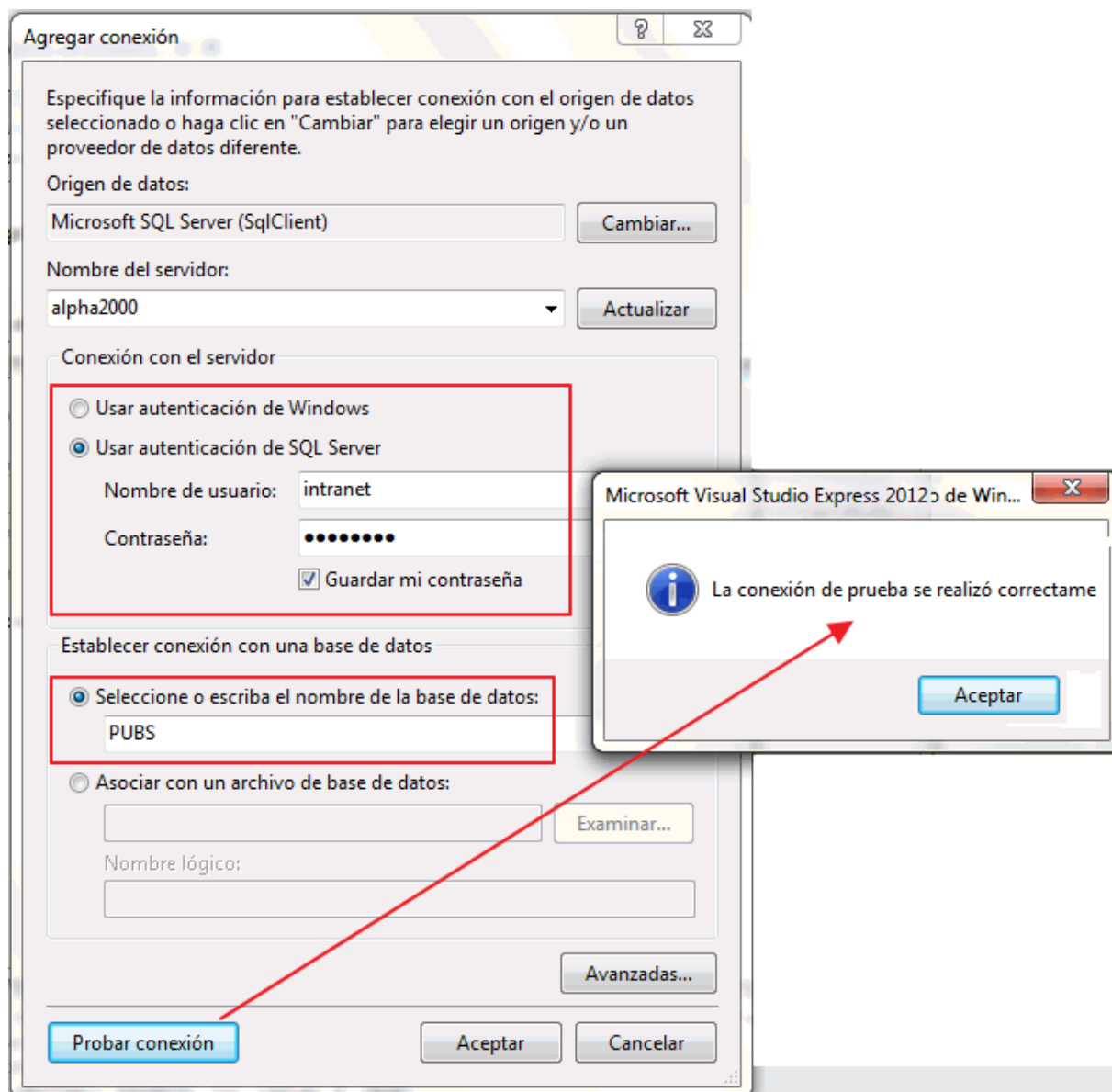
Propiedad	Descripción
ConnectionString	Cadena utilizada para conectar con la fuente de datos
ConnectionTimeout	Número de segundos después de cual una conexión fallida de interrumpe. El valor predeterminado es de 15 segundos
Database	Devuelve el nombre de la base de datos especificada en la propiedad 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.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="tema15_conexion.My.MySettings.Cadena_SQLCompact"
          connectionString="Data Source=|DataDirectory|\Northwind40.sdf"
          providerName="Microsoft.SqlServerCe.Client.4.0" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

Es el fichero XML que incorporan los proyectos y dónde el asistente de la cadena de conexión nos 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 DateReader 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
Dim cn As New SqlConnection(cadenaconexion)
cn.Open()

'Definimos el comando para insertar una fila en la base de datos:
Dim sql As String = "insert into Autores (Autor,Libro) values ('Camilo Jose Cela','La colmena')"
```



```
'Creamos un comando de ejecución:
Dim cmd As New SqlCommand()
cmd.Connection = cn
cmd.CommandText = sql
```



```
'Ejecuta la consulta y obtiene el número de registros afectados
Dim registros As Integer = cmd.ExecuteNonQuery()
MessageBox.Show("Registros afectados:" & registros)
```



```
'Cierra la conexión
cn.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
    'Ejecuta la consulta
    Dim registros As Integer = cmd.ExecuteNonQuery()
Catch ex As Exception
    'Procesamos el error
Finally
    'Cerramos la conexión
    cn.Close()
End Try
```

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:

```
Dim cn As New SqlConnection()

cn.ConnectionString = "Provider=SQLOLEDB;Data Source=Servidor" & "Initial
Catalog=Northwind;Trusted_Connection=Yes;"

cn.Open()

Dim cmd As SqlCommand = cn.CreateCommand()

cmd.CommandText = "UPDATE Clientes SET NombreCompañía = 'Nuevo valor' WHERE
IdCliente = 'ALFKI'"

cmd.ExecuteNonQuery()

cn.Close()
```

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

```
Dim strConn As String
```

```
strConn = "Data Source=Servidor;InitialCatalog=Northwind;Trusted_Connection=Yes;"
```

```
Dim cn As New SqlConnection(strConn)
```

```
cn.Open()
```

```
Dim cmd As SqlCommand = cn.CreateCommand()
```

```
cmd.CommandText = "UPDATE Clientes SET NombreCompañía = 'Nuevo valor' WHERE  
IdCliente = 'ALFKI'"
```

```
Dim intRegistrosAfectados As Integer = cmd.ExecuteNonQuery()
```

```
If intRegistrosAfectados = 1 Then
```

```
    Console.WriteLine("Actualización correcta...")
```

```
Else
```

```
    'Assume intRecordsAffected = 0
```

```
    Console.WriteLine("Se ha producido un error en la actualización.")
```

```
End If
```

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:

```
Dim id_autor as string = txt_id.text.Replace ("'", "'")
```

Es decir creamos otra variable donde hemos sustituido ese carácter ' por una comilla doble ", entonces si 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 cliente_id= 'ABCD'
```

que sustituiremos por:

```
Select * from Clientes where cliente_id= @cliente_id
```

En esa variable @cliente_id 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 digamos 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:

```
Dim sql As String
sql = "Insert into authors (au_id,au_fname,au_lname,phone,address,city,state,zip,contract)
sql &= " values ("
sql &= " @au_id,@au_fname,@au_lname,@phone,"
sql &= " @address,@city,@state,@zip,@contract) "
```

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

```
'Definimos los objeto ADO.NET:
Dim conexion As New SqlConnection(cadena_conexion)
Dim comando As New SqlCommand(sql, conexion)

comando.Parameters.AddWithValue("@au_id", txt_id.Text)
comando.Parameters.AddWithValue("@au_fname", txt_nombre.Text)
comando.Parameters.AddWithValue("@au_lname", txt_apellidos.Text)
comando.Parameters.AddWithValue("@phone", txt_telefono.Text)
comando.Parameters.AddWithValue("@address", txt_direccion.Text)
comando.Parameters.AddWithValue("@city", txt_ciudad.Text)
comando.Parameters.AddWithValue("@state", txt_estado.Text)
comando.Parameters.AddWithValue("@zip", txt_codigopostal.Text)
comando.Parameters.AddWithValue("@contract", Val(chk_contract.Checked))

Dim operacion As Integer = 0
'Abrimos la base de datos y leemos: _
```

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
Dim sql As String
sql = "update authors set"
sql &= " au_fname=@au_fname, au_lname=@au_lname,"
sql &= " phone=@phone, address=@address,"
sql &= " city=@city,state=@state,"
sql &= " zip=@zip,contract=@contract"
sql &= " where au_id=@au_id_original"

'Definimos los objeto ADO.NET:
Dim conexion As New SqlConnection(cadena_conexion)
Dim comando As New SqlCommand(sql, conexion)

comando.Parameters.AddWithValue("@au_fname", txt_nombre.Text)
comando.Parameters.AddWithValue("@au_lname", txt_apellidos.Text)
comando.Parameters.AddWithValue("@phone", txt_telefono.Text)
comando.Parameters.AddWithValue("@address", txt_direccion.Text)
comando.Parameters.AddWithValue("@city", txt_ciudad.Text)
comando.Parameters.AddWithValue("@state", txt_estado.Text)
comando.Parameters.AddWithValue("@zip", txt_codigopostal.Text)
comando.Parameters.AddWithValue("@contract", Val(chk_contract.Checked))
comando.Parameters.AddWithValue("@au_id_original", lista_autores.SelectedItem.Value)
```

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 au_id=@au_id_original"
```

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

```
Dim sql As String
sql = "Delete from authors "
sql &= " where au_id=@au_id_original"
```

'Definimos los objeto ADO.NET:

```
Dim conexion As New SqlConnection(cadena_conexion)
Dim comando As New SqlCommand(sql, conexion)
```

```
comando.Parameters.AddWithValue("@au_id_original", lista_autores.SelectedItem.Value)
```

```
Dim operacion As Integer = 0
```

Nos funciona pero nos da un problema ya que como este autor tiene libros suyos en la tabla de los libros, nos avisa 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`" o "`OleDbCommand`" que es muy fácil de definir:

```
Dim comando as SqlCommand
O
Dim comando as OleDbCommand
```

Veamos un ejemplo de cómo crearlo de tres formas distintas:

```
'Método 1. Creandolo en dos pasos
Dim comando As New SqlCommand
comando.Connection = conexion

'Metodo 2. Creandolo en un solo paso
Dim comando As 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:

```
dim num_registros as Integer = Cint (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:

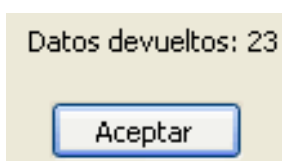
```
Using conexion = New SqlConnection(cadena_conexion)

    'Metodo 2. Creandolo en un solo paso
    Dim comando As New SqlCommand("select count(*) from authors", conexion)
    'Abrimos conexión
    conexion.Open()
    Dim total As Integer = CInt(comando.ExecuteScalar())

    MessageBox.Show("Datos devueltos: " & total)

End Using
```

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

```
Dim sql_dr as SqlDataReader(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 <code>ExecuteReader</code> 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 y la propiedad Item
Datos.Item("Apellidos")
Accediendo a un campo mediante su nombre (la propiedad Item está marcada como
Default)
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 sql_dr.hasrows then
    While sql_rd.Read
        Listbox1.additem(sql_dr.getvalue(0) & sql_dr.getvalue (1))
    End While
End If
```

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 cualquier 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:

```
Imports System.Data.SqlClient
Imports System.Data.Common

Public Class Form1

    Private cadena_conexion As String = "Data Source=SERVIDOR\SQLEXPRESS;Initial
Catalog=northwind;Integrated Security=True"

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load
        Dim matriz_resultado As New ArrayList
        Using conexion As New SqlConnection(cadena_conexion)

            Dim comando As New SqlCommand("Select * from suppliers", conexion)
            conexion.Open()
            Dim resultado As SqlDataReader =
comando.ExecuteReader(CommandBehavior.CloseConnection)
            If resultado.HasRows Then
                For Each registro As DbDataRecord In resultado
                    matriz_resultado.Add(registro)
                Next
            End If
            End Using 'En este momento se hace un conexion.dispose automático
            cuadrícula.DataSource = matriz_resultado 'Enlazamos con la cuadrícula
        End Sub
    End Class
```

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 "arralist". 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 nuestra pantalla nos muestra 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
...
end Using
```

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:

```
Imports System.Data.SqlClient
Public Class Form1

    Private cadena_conexion As String = "Data Source=SERVIDOR\SQLEXPRESS;Initial
Catalog=northwind;Integrated Security=True"

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load
        Dim conexion As New SqlConnection(cadena_conexion)
        Try
            Using conexion
                'Intentamos abrir la conexion
                conexion.Open()
                lbl_estado.Text = "Versión del servidor: " & conexion.ServerVersion
                lbl_estado.Text &= "La conexión es:" & conexion.State.ToString()
            End Using
        Catch err As Exception
            'Vemos el error producido:
            lbl_estado.Text = "Error en la conexión de la base de datos..."
            lbl_estado.Text &= err.Message
        End Try
        'Ha ido todo bien cerramos la conexión
        conexion.close()
        lbl_estado.Text = "La conexión está ahora: " & conexion.State.ToString()

    End Sub
End Class
```

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

12.2. Consultar conjuntos de datos grandes de forma asíncrona.

Para estas consultas de muchos registros utilizaremos procesos "Asíncronos", en los cuales llamamos el proceso pero sin quedarnos a esperar una respuesta. En lugar de rellenar el arraylist como hicimos antes utilizaremos el método "beginExecuteReader" del objeto SqlCommand que pasa a ejecutar el comando de forma asíncrona. Le estamos indicando... "haz la ejecución de ese comando y cuando hayas terminado me avisas con un evento". El cambio que tendremos que hacer respecto al código anterior es sustituir "ExecuteReader" por "BeginExecuteReader". Además de ponerle un par de instrucciones para realice el proceso de forma asíncrona:

```
Dim terminado as AsyncCallback=New AsyncCallback (resultado_completado)
Dim resultado as IAsyncResult =comando.beginexecutereader(terminado,comando)
```

Esto es un código un poco avanzado pero así es el punto de partida para la ejecución asíncrona. De esta forma lanzamos la consulta y podemos tener el control del programa para realizar otras opciones mientras termina de llegar el resultado que sería:

```
Private sub resultado_completado (byval resultado as IAsyncResult)
    messagebox.show ("Carga de datos completada")
    Dim comando As sqlcommand= CType(result.AsyncState, SqlCommand)
    Dim resultado As SqlDataReader=comando.EndExecuteReader (resultado)
    if resultado.hasrows then
        For each registro as dbDataRecord in resultado
            matriz_resultado.add (registro)
        Next
    End if

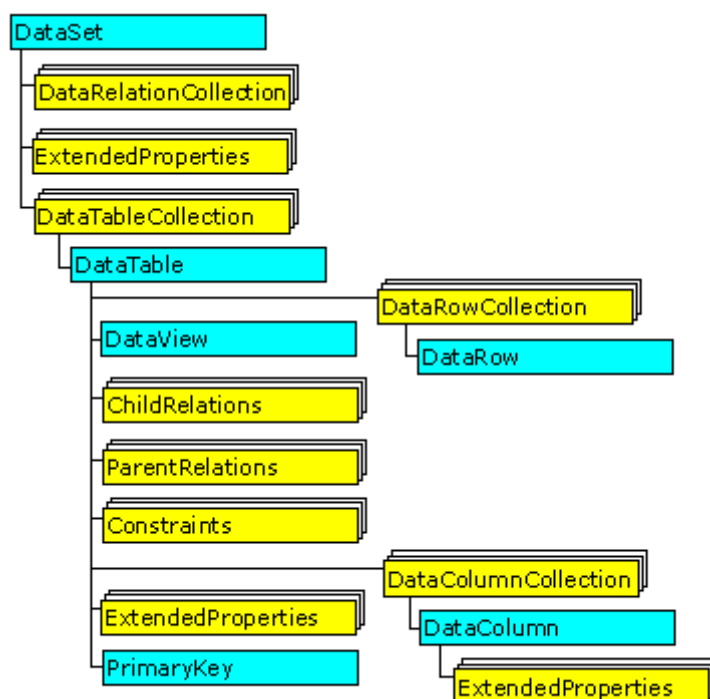
    sql_dr.close ()
    comando.connectionDispose()
    'Enlazamos con la cuadrícula
    cuadrícula.datasource=matriz_resultado
```

La ejecución será aparentemente igual que antes porque tenemos pocos registros en nuestro ejemplo pero con tablas grandes el resultado es que mientras se recuperan los datos podemos ir realizando otras operaciones con el programa, indicando por ejemplo en una barra de estado "recuperando datos...".

13. DataSets

Una vez visto el conjunto de resultados obtenido en los Datareaders en un modelo conectado con el servidor vamos a pasar ahora a ver la misma tecnología pero con objetos desconectados. Trabajar en modo sin conexión es la técnica más escalable que se puede adoptar, ya que demanda recursos del cliente (en lugar del servidor) y sobre todo, no impone ningún bloqueo en las tablas de las bases de datos (excepto para los bloqueos de pequeña duración que se crean durante las operaciones de actualización).

El objeto DataSet es esencial para admitir escenarios de datos distribuidos de ADO.NET sin mantener una conexión. El DataSet es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos. Se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación. El DataSet representa un conjunto completo de datos entre los que se incluyen tablas relacionadas, restricciones y relaciones entre las tablas.



Cuando utilizamos datos desconectados lo que hacemos es mantener en memoria una copia de los datos utilizando un "DataSet". Cuando haya cambios nos volveremos a conectar y actualizaremos los datos de este DataSet en la base de datos.

Antes simplemente hacíamos una lectura de un DataReader que no almacena nada en memoria. Según se lee se vuelca en formulario y no volvemos a acceder a él (aunque mantiene viva una conexión). Con los DataSet mantendremos en memoria un conjunto de resultados que se desconecta completamente de la bbdd, que podremos modificar y luego volcarlos otra vez al

servidor de base de datos. Tenemos varias razones por las que hay que utilizar esta otra forma de trabajar con los datos en memoria:

- Necesitamos utilizar mucho tiempo la base de datos. Con un DataSet conseguimos volcar desde la base de datos a nuestra aplicación y viceversa de una forma rápida y así reducimos el tiempo de uso.
- Necesitamos rellenar un control de datos, por ejemplo, de DataGridView de .NET. El DataReader es muy limitado y no nos permitirá muchas de las funciones que proporcionan los controles de cuadrícula. Por ejemplo, la paginación de resultados, con el DataReader no se puede y con los DataSet sí
- Navegación por los resultados en los dos sentidos. Con los DataReader solo podemos hacer un bucle desde el primer registro al último. Con los DataSet podemos movernos libremente por los registros en los dos sentidos.
- Podemos movernos de una tabla a otra. En un DataSet podemos almacenar varias tablas con datos e incluso definir relaciones entre ellas para mejorar la navegación.
- Con los DataSet podemos almacenar los datos para su uso posterior. Por ejemplo, exportar una consulta a XML.
- Proporciona una forma de empaquetar datos que podemos enviar a otros objetos.
- Podemos almacenar datos que pueden ser utilizados más adelante.

Los DataSet almacenan los cambios que se realizan en él (modificar un nombre de un empleado en la tabla de empleados cargada en el DataSet). Cuando hemos hecho los cambios provocaremos un "Update" que actualizará los cambios en la base de datos. Por eso se llama acceso desconectado, porque se carga un DataSet, se cierra la conexión de la bbdd, se realizan los cambios necesarios y se vuelcan a la bbdd.

De todas formas en la práctica se utilizan mucho los DataSet pero a la hora de modificar datos se ejecuta la SQL necesaria. En una aplicación Windows, el DataSet permanece en memoria mucho tiempo y la sincronización es sencilla. En Web se carga el DataSet y se desconecta hasta que el usuario hace alguna solicitud en la página. Esto hace que el DataSet es distinto en los momentos en los que se sirve al usuario y cuando vuelve a solicitarlos para una operación. En el programa Windows está permanentemente en memoria pero aquí no, por lo tanto puede haber estas diferencias. Por eso te comento lo de que se utiliza mucho para enlazarlo con controles complejos porque proporcionan mucha potencia y a la hora de modificar datos se utiliza SQL. Recuerda que en un DataSet podemos almacenar varias tablas:

Cada tabla contenida en un objeto DataSet se encuentra disponible a través de su propiedad Tables que es una colección de objetos System.Data.DataTable. A su vez cada objeto DataTable contiene las filas de datos en DataRow. Las columnas también tienen su objeto que es DataColumn.

El objeto DataAdapter es el objeto que se conecta a la base de datos para llenar el objeto DataSet. A continuación, se vuelve a conectar a la base de datos para actualizar los datos de dicha base de datos a partir de las operaciones realizadas en los datos contenidos en el objeto DataSet. En el pasado, el procesamiento de datos se basaba principalmente en la conexión. Ahora, con el fin de proporcionar a las aplicaciones multinivel mayor eficacia, se está adoptando para el procesamiento de datos un enfoque basado en mensajes que manipulan fragmentos de información. En el centro de este enfoque se sitúa el objeto DataAdapter, que proporciona un puente entre un objeto DataSet y un almacén de datos de origen para recuperar y guardar datos. Para ello, envía solicitudes a los comandos SQL apropiados que se ejecutan en el almacén de datos.

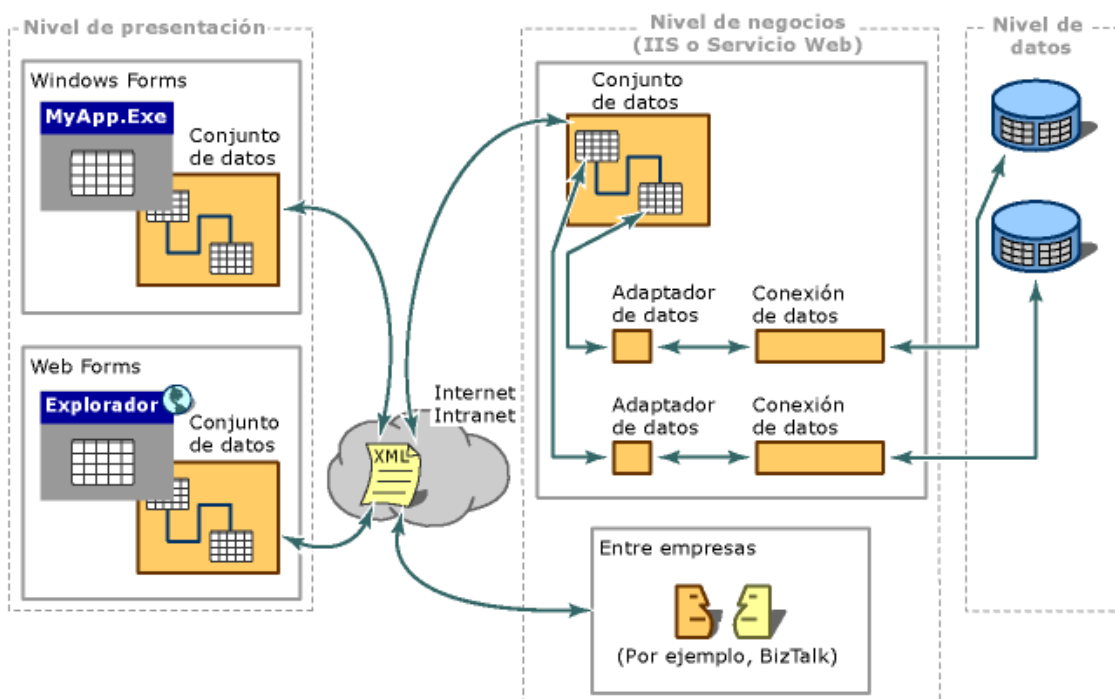
De momento tenemos:

- Una conexión (connection)
- Un comando (command)
- Donde recuperamos los datos (DataSet)
- El objeto para sincronizar estos datos (DataAdapter)

Puesto que estamos hablando de objetos desconectados, es decir, realizamos una consulta y se desconecta de la base de datos, si quiero actualizar un registro debe haber algún tipo de enlace entre esos datos y su origen: DataAdapter.

Observemos detenidamente este gráfico:

Componentes de datos ADO.NET



La parte de la izquierda tenemos los programas de usuario. De ellos en la parte superior los Windows Forms (programas Windows) y en la parte inferior los Web Forms (páginas web de ASP.NET). En el centro tenemos el formato XML que es el formato estándar para trabajar con bases de datos.

En la siguiente capa tenemos la forma de trabajar con los datos, tenemos los datos por una parte y debajo puedes ver el "adaptador de datos" y la "conexión de datos", finalmente la base de datos física a la derecha. Así que tenemos el conjunto de datos (DataSet), la forma de conectarlo con el origen (DataAdapter), el objeto conexión (Connection) y finalmente la base de datos.

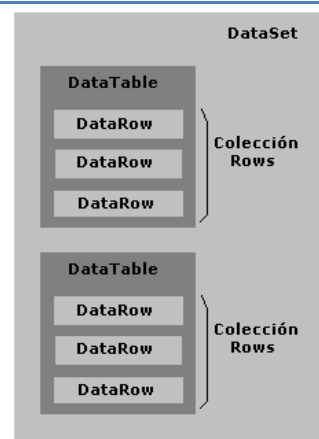
Sigamos ahora con los detalles de los DataSet, en concreto sus propiedades más importantes

Propiedad	Descripción
CaseSensitive	Indica si las comparaciones de texto dentro de las tablas distinguen entre mayúsculas y minúsculas. Por defecto tiene False
DataSetName	Establece o devuelve el nombre del objeto DataSet
HasErrors	Devuelve un verdadero o falso indicando si hay algún error en las tablas DataSet
Relations	Devuelve la colección de objetos DataRelation que representan todas las relaciones existentes entre las tablas del objeto DataSet.
Tables	Devuelve una colección de objetos DataTable que representa a cada una de las tablas existentes dentro del objeto DataSet

Y ahora sus métodos:

Método	Descripción
Clear	Elimina todos los datos almacenados en el objeto DataSet
AcceptChanges	Confirma todos los cambios realizados en las tablas y relaciones contenidas en el objeto DataSet, o todos los cambios realizados desde la última llamada de AcceptChanges
GetChanges	Devuelve un objeto DataSet que contiene todos los cambios realizados desde que se cargó con datos, o desde la última llamada de AcceptChanges
HasChanges	Devuelve cierto o falso para indicar si se han realizado cambios en el contenido del DataSet desde que fue cargado o desde la última llamada de AcceptChanges
RejectChanges	Abandona todos los cambios realizados en las tablas contenidas en el objeto DataSet desde que se cargó el objeto o desde la última llamada de AcceptChanges
Merge	Toma los contenidos de un Dataset y los mezcla con otro DataSet de forma que contiene ambos datos.

Y ahora algunos detalles de los principales objetos. Fíjate primero en este gráfico para ver la jerarquía de los objetos.



13.1. Objetos del dataset

DataTable

Esos objetos representan tablas de datos en memoria y se compone de una colección de filas (rows) y columnas (columns) como habrás podido imaginar. También tiene una colección de "constraint" que junto con las columnas (que son la definición de los campos) definen el esquema de la tabla o DataTable.

Propiedad	Descripción
Columns	Esos objetos son una instancia de la clase System.Data.DataColumnCollection y contiene una colección de objetos de tipo DataColumn. Éstos definen las propiedades de cada columna (DataColumn) como sus nombres, tipos de datos y clave primaria.
Rows	Es una colección de System.Data.DataRowCollection y contiene una colección de DataRow. Los objetos DataRow contienen los datos del DataTable definidos por las columnas DataTable.Columns. Cada DataRow contiene un elemento para columna DataColumn de la colección Columns
Constraints	Es una instancia de la clase System.Data.ConstraintCollection y se compone de una serie de definiciones para relacionar las tablas y qué acciones realizar cuando se modifican campos que son claves únicas.

DataColumn

Definen los nombres y tipos de datos de las columnas en un DataTable. Podemos crear estos objetos mediante su constructor o añadiendo una columna a la colección de columnas:

```
Dim columna as New DataColumn ("id",GetType(System.Int32))
Tabla_productos.Columns.Add ("id",GetType(System.Int32) )
```

La creación completa quedaría:

```
'Creamos una tabla
Dim tabla_productos as New DataTable ("productos")
'Creamos las columnas
Tabla_productos.Columns.Add ("id",GetType(System.Int32))
Tabla_productos.Columns.Add ("Nombre",GetType(System.String))
Tabla_productos.Columns.Add ("Categoría",GetType(System.Int32))
```

DataRow

Con la definición de la tabla realizada ahora podemos acceder a las filas de datos bien para introducir o bien para recuperarlos. Para crear una nueva fila ya te imaginarás que podremos hacerlo mediante su propia colección de filas. Primero, por tanto, definiremos una fila, le metemos los datos y la añadimos a la tabla:

```
'Creamos un DataRow con el mismo esquema que el DataTable
Dim fila_temporal as DataRow= Table_productos.NewRow()
'Establecemos los valores:
fila_temporal.Item ("Id")=1
fila_temporal.Item ("Nombre")="Nombre del producto..."
fila_temporal.Item ("Categoria")="Categoría..."
'Aquí añadimos la fila a la colección de la tabla:
tabla_productos.rows.add(fila_temporal)
```

Constraints

Este objeto, como hemos dicho antes, obtiene la colección de restricciones mantenidas por esta tabla. Lógicamente estamos hablando ya de tablas relacionadas. Existen dos tipos de restricciones:

- **ForeignKeyConstraint.** Se utilizan para forzar a que se cumpla la integridad referencial. Por ejemplo, si se elimina una clave primaria en una tabla indicará que se deben eliminar todas las filas asociadas a esa clave en las tablas "hijas". Este tipo de restricción está pensado para utilizarlo con columnas de clave principal. En una relación primaria-secundaria entre dos tablas, la eliminación de un valor de la tabla primaria puede afectar las filas secundarias de una de las formas siguientes.
 - **Cascade.** Se pueden eliminar las filas secundarias (acción en cascada)
 - **SetNull.** Los valores de las columnas secundarias se pueden establecer en nulos.
 - **SetDefault.** Los valores de las columnas secundarias se pueden establecer en predeterminados.
 - **None.** No hacer nada.
- **uniqueConstraint.** Fuerza a que el valor de una columna sea único. Este tipo de restricción es automático en claves primarias.

13.2. Eventos que se pueden activar en los DataTables

Evento	Descripción
Initialized	Se produce después de haber llamado al método EndInit de un objeto DataTable. Este evento está concebido principalmente para admitir escenarios en tiempo de diseño.
ColumnChanged	Se produce después de cambiar correctamente un valor en un objeto DataColumn.
ColumnChanging	Se produce cuando se ha enviado un valor para un objeto DataColumn.
RowChanged	Se produce cuando se ha cambiado correctamente un valor de DataColumn o la propiedad RowState de un objeto DataRow en el objeto DataTable.
RowChanging	Se produce cuando se ha enviado un cambio para un valor DataColumn o la propiedad RowState de un objeto DataRow en el objeto DataTable.
RowDeleted	Se produce después de marcar un objeto DataRow de un objeto DataTable como Deleted.
RowDeleting	Se produce antes de marcar un objeto DataRow de un objeto DataTable

	como Deleted.
TableCleared	Se produce después de que una llamada al método Clear del objeto DataTable haya borrado correctamente todos los objetos DataRow.
TableClearing	Se produce después de haber llamado al método Clear pero antes de que se inicie la operación Clear.
TableNewRow	Se produce después de crear un nuevo objeto DataRow mediante una llamada al método NewRow del objeto DataTable.
Disposed	Se produce cuando el objeto DataTable se establece en Disposed. Se hereda de MarshalByValueComponent.

Digamos que es interesante que los conozcas por si quisieras controlar los eventos cuando se activen en el código.

Relacionar datos

Finalmente nos queda establecer la relación de varios objetos DataTable que se alojarán en el DataSet. Para esto disponemos de una propiedad "DataSet.Relations" que es una colección de objetos de tipo DataRelation que se utilizan para crear relaciones del tipo padre-hijo entre tablas del DataSet. Veamos un ejemplo de cómo crearlo por código:

'Crear una relación entre dos tablas: Pedidos y Cliente:

```
midataset.Relations.Add("Clientes_pedidos", midataset.Tables ("clientes").Columns("idcliente"), _  
midataset.Tables("Pedidos").Columns("idcliente"))
```

Este ejemplo invoca al método Add del objeto DataSet.Relation para crear un objeto DataRelation en la colección DataRelations, donde creamos una relación entre las dos tablas.

13.3. Ejemplo

Vamos a poner todo junto y a realizar todo "a mano" en un ejemplo con dos tablas en las que crearemos una relación. Las tablas van a ser estas:

Propiedad	Descripción
id_animal	Clave primaria
Nombre	Nombre del animal

y

Propiedad	Descripción
id_mascota	Clave primaria
id_animal	Clave externa que relaciona con la otra tabla
nombre_mascota	Nombre de la mascota

Crearemos un formulario y en el evento clic de un botón crearemos las tablas, sus relaciones y pondremos algunos datos:

Form1

Crear Tabla

	id_animal	Nombre
▶	1	Perro
	2	Gato
	3	Pez
*		

	id_mascota	id_animal	Nombre_mascota
▶	1	1	pepe
	2	1	Jose
*			

Para empezar definimos un Dataset:

```
Dim datos_mascotas As DataSet
```

Y en el evento clic ponemos las llamadas a crear el Dataset, rellenar datos y enlazar con los DataGridViews.

```
Private Sub btn_crear_Click(...)
    datos_mascotas = Crear_esquema()
    datos_ejemplo()
    cuadrricula_animales.DataSource = datos_mascotas.Tables("Animal")
    cuadrricula_mascotas.DataSource = datos_mascotas.Tables("Mascotas")
End Sub
```

En la creación del esquema ponemos lo que hemos visto hasta ahora para crear las dos tablas con sus campos y una relación al final:

```
Private Function Crear_esquema() As DataSet
    Dim tabla_animales As DataTable = New DataTable("Animal")
    Dim columna As DataColumn
```

```
'Primera columna
columna = New DataColumn("id_animal", GetType(System.Int32))
columna.AutoIncrement = True 'Activamos el identity
tabla_animales.Columns.Add(columna)

'Segunda columna
columna = New DataColumn("Nombre", GetType(System.String))
tabla_animales.Columns.Add(columna)

'Creamos la otra tabla
Dim tabla_mascotas As DataTable = New DataTable("Mascotas")

'Primera columna
columna = New DataColumn("id_mascota", GetType(System.Int32))
columna.AutoIncrement = True 'Activamos el identity
tabla_mascotas.Columns.Add(columna)

'Segunda columna
columna = New DataColumn("id_animal", GetType(System.Int32))
tabla_mascotas.Columns.Add(columna)

'Tercera columna
columna = New DataColumn("Nombre_mascota", GetType(System.String))
tabla_mascotas.Columns.Add(columna)

'Con las dos tablas creadas, creamos ahora el Dataset:
Dim datos As DataSet = New DataSet("Datos_mascotas")
datos.Tables.Add(tabla_animales)
datos.Tables.Add(tabla_mascotas)
'Y ahora la relación entra las dos:
datos.Relations.Add(New DataRelation("Animal_mascotas", _
    tabla_animales.Columns("id_animal"), tabla_mascotas.Columns("id_animal")))
Return datos
End Function
```

Ahora tenemos que rellenar con unos datos de prueba:

```
Sub datos_ejemplo()
    Dim dr As DataRow
    'Tres filas en la primera tabla
    dr = datos_mascotas.Tables("animal").NewRow
```

```
dr(0) = 1
dr(1) = "Perro"
datos_mascotas.Tables("animal").Rows.Add(dr)

dr = datos_mascotas.Tables("animal").NewRow
dr(0) = 2
dr(1) = "Gato"
datos_mascotas.Tables("animal").Rows.Add(dr)

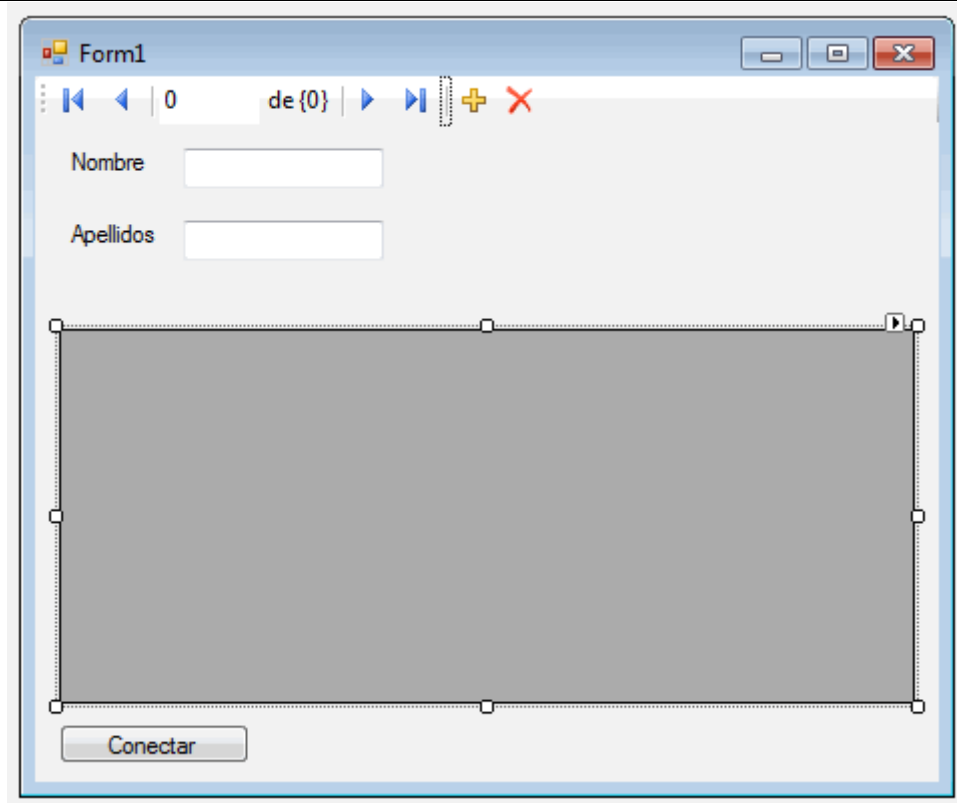
dr = datos_mascotas.Tables("animal").NewRow
dr(0) = 3
dr(1) = "Pez"
datos_mascotas.Tables("animal").Rows.Add(dr)

'Dos filas en la segunda tabla:
dr = datos_mascotas.Tables("mascotas").NewRow
dr(0) = 1
dr(1) = 1
dr(2) = "pepe"
datos_mascotas.Tables("mascotas").Rows.Add(dr)
dr = datos_mascotas.Tables("mascotas").NewRow
dr(0) = 2
dr(1) = 1
dr(2) = "Jose"
datos_mascotas.Tables("mascotas").Rows.Add(dr)
```

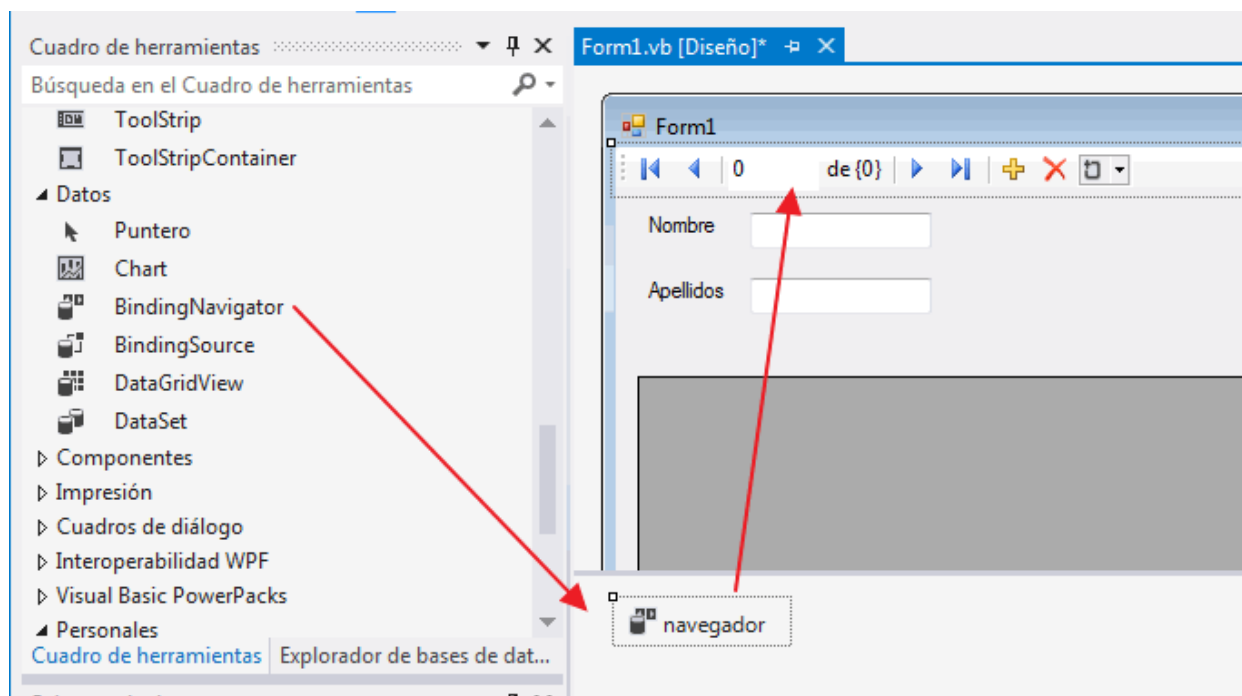
End Sub

Si revisamos el código detenidamente no hay nada que no hayamos comentado y se limita a la creación a mano de un dataset con dos tablas.

Ahora vamos a hacer un sencillo ejemplo donde vamos a ver cómo hacer a mano los formularios que nos crea el IDE:



Hemos puesto un datagridview, dos cuadros de texto y un control de tipo navegación "Databinding"



Haremos todo por código. La primera parte ya la conoces: conexión, adaptador y dataset:

```
'datos de conexión
Dim cadena_conexion As String = "Data Source=ALPHA2000;Initial Catalog=pubs;Use
Dim conexion As SqlConnection = New SqlConnection(cadena_conexion)
Dim resultado As New DataSet

'conectar
conexion.Open()
Dim adaptador As New SqlDataAdapter("Select * from authors", conexion)

'Rellenamos un dataset
adaptador.Fill(resultado, "autores")
```

Ahora viene la parte nueva, cuando trabajamos por primera vez con un control de este tipo que es de **enlace a datos**: BindingSource. Este control se puede enlazar universalmente en todos los controles de formularios Windows Forms a orígenes de datos.

```
'Creamos un enlace con "BindingSource"
Dim enlace As New BindingSource
enlace.DataSource = resultado.Tables(0)

'Enlazamos los botones del navegador y de la cuadrícula con el enlace.
navegador.BindingSource = enlace
Me.DataGridView1.DataSource = enlace

'Enlazamos los cuadros de texto:
txt_nombre.DataBindings.Add(New Binding("Text", enlace, "au_fname", True))
txt_apellidos.DataBindings.Add(New Binding("Text", enlace, "au_lname", True))
```

Primero creamos un objeto de enlace de datos para utilizarlo luego. En este enlace hemos puesto la primera tabla del dataset que es nuestra consulta a la base de datos. Después hemos enlazado el "navegador" que es el conjunto de iconos de la parte superior con ese origen de datos, así vincula todas sus opciones de movimiento de registros. Por último, y como algo accesorio, ver cómo se pueden enlazar campos con controles, en este caso dos campos de la base de datos con dos cuadros de texto mediante un enlace o "Binding".

Esto es lo necesario para comenzar a trabajar de forma avanzada y así saber cómo poder crear pantallas con todo enlazado de una forma más sencilla. Sin duda es mejor esto que todos los controles que nos creó el asistente que vimos. Lo hacemos todo a mano y queda mucho más personalizado, con las mismas funcionalidades.

El resultado es que además de enlazar la cuadrícula hemos enlazado los cuadros de texto, sincronizando todo con el control de navegación de arriba.

	au_id	au_lname	au_fname	phone
	172-32-1176	Whiterr	Johnsonr	408 496-7223
	213-46-8915	Green	Marjorie	415 986-7020
	238-95-7766	Carson	Cheryl	415 548-7723
▶	267-41-2394	O'Leary	Michael	408 286-2428
	274-80-9391	Straight	Dean	415 834-2919
	341-22-1782	Smith	Meander	913 843-0462
	409-56-7008	Bennet	Abraham	415 658-9932

Así, podemos hacer consultas en un dataset que sean combinaciones de otras tablas y enlazarlas en una cuadrícula. Hasta ahora solo podíamos con los asistentes poner tablas pero no consultas. Con este ejemplo podemos poner la sentencia SQL que queramos. Ahora necesitaremos modificar los datos para añadir, borrar y modificar.

Podríamos haber hecho un enlace de una forma más sencilla en el cuadro de texto:

```
txt_nombre.DataBindings.Add("Text", resultado.Tables(0), "au_fname")
```

Pero no quedaría sincronizado con el control de navegación y tendríamos que mover nosotros el registro a mostrar con el objeto de posicionamiento CurrencyManager. En cambio la que hemos puesto:

```
txt_nombre.DataBindings.Add(New Binding("Text", enlace, "au_fname", True))
```

Hemos añadido un enlace a una colección de enlaces que pueden tener los controles. Este nuevo enlace, que con uno nos basta, enlaza la propiedad "Text" que es precisamente el texto del contenido con el origen de datos "enlace" y con el campo "au_fname" de ese DataSet. La última propiedad es para temas de formato.

Lo interesante ahora es hacer un programa para explotar bases de datos sencillas haciendo a mano las cosas. Vamos con la modificación de los datos...

Nota: Hemos visto que la SQL ahora es más asequible para modificarla en el código, pero... ¿hay alguna forma de poder ejecutar sentencias SQL libremente?

Pues sí, con el objeto conexión de esta forma:

```
conexion.Execute("Update Authors Set au_fnombre = 'Jose'", adCmdText, adAsyncExecute)
```

Esto es para que puedas tener más opciones a la hora de construir tu programa y con la versatilidad de ejecutar sentencias SQL.

14. DataAdapter

Técnicamente un adaptador representa un conjunto de comandos SQL y una conexión de base de datos que se utilizan para rellenar el objeto Dataset y actualizar el origen de datos. Ya lo comentamos, cuando hacemos una consulta con un Dataset los datos pasan a ser locales así que de alguna forma luego hay que volcarlos en el servidor. Este objeto es el encargado de realizar esta sincronización de datos entre lo que hemos modificado localmente y el servidor de datos. Esto lo hace mediante un conjunto de comandos que tiene almacenados.

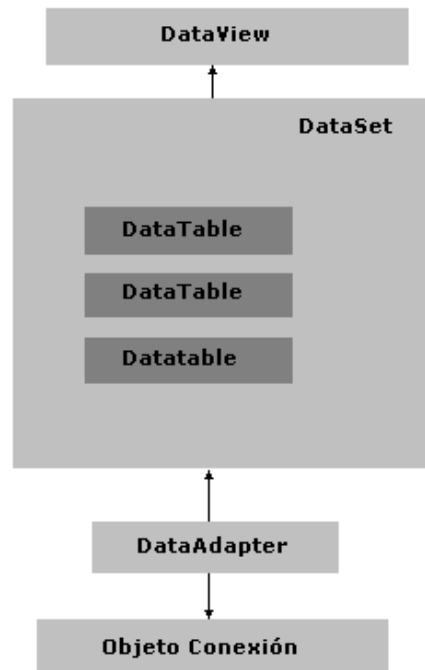
Un objeto **DataAdapter** puede contener desde una sencilla consulta SQL hasta varios objetos Command. Esta clase tiene cuatro propiedades que permiten asignar un objeto **Command** con las operaciones estándar de manipulación de datos:

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

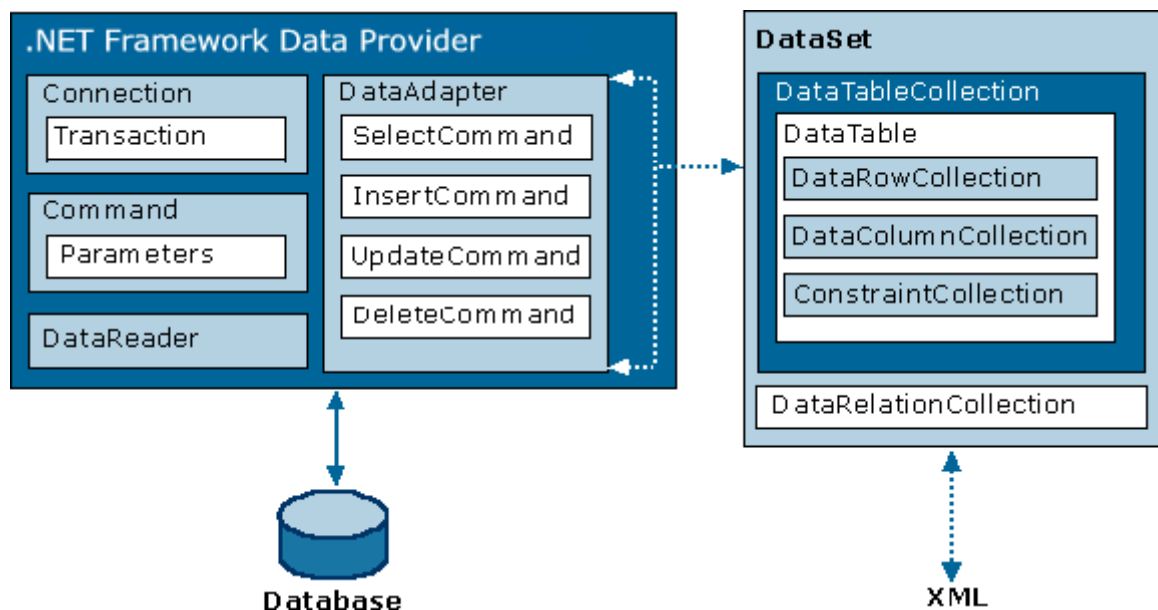
Es decir, todas las operaciones posibles con los datos: altas, bajas, consultas y modificaciones. El método **Fill** ejecuta el comando de selección que se encuentra asociado a la propiedad SelectCommand y los datos se cargan en el objeto DataSet que pasamos como parámetro.

DataAdapter actúa como puente entre DataSet y un origen de datos para la recuperación y el almacenamiento de datos. Para poder actuar como un puente, DataAdapter asigna el método **Fill**, que modifica los datos de DataSet de forma que coincidan con los del origen de datos, y asigna el método **Update**, que modifica los datos del origen de datos para hacerlos coincidir con los de DataSet.

Una forma resumida de ver la posición del DataAdapter en nuestros objetos es:

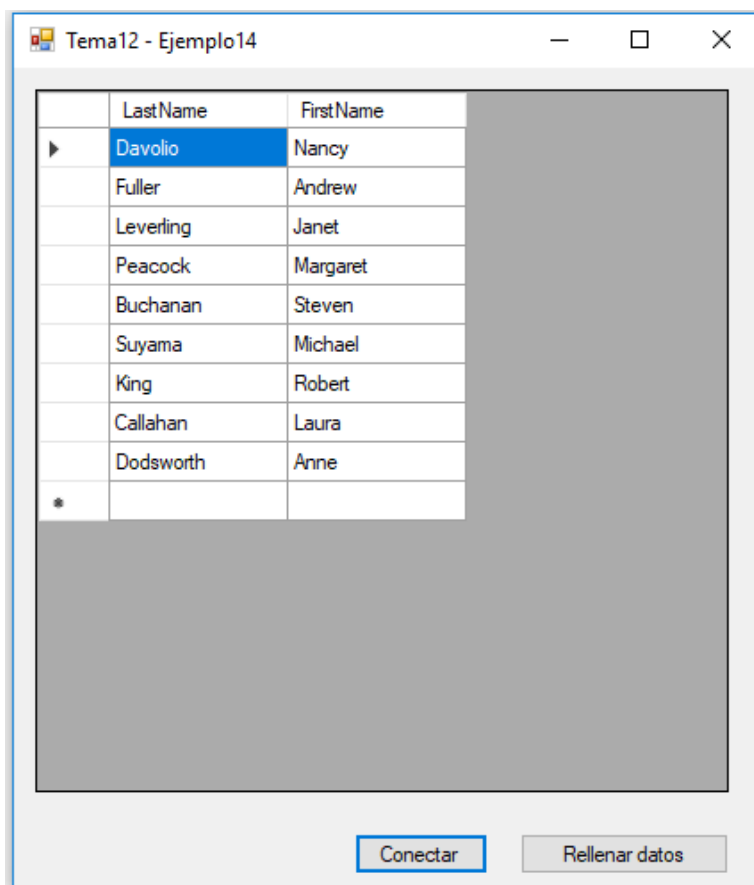


Y de forma más avanzada podemos ver que el adaptador tiene los cuatro comandos para hacer las operaciones de tratamiento de datos.



El objeto **DataAdapter** utiliza comandos para actualizar el origen de datos después de hacer modificaciones en el objeto **DataSet**. Si se utiliza el método **Fill** del objeto **DataAdapter**, se llama al comando **SELECT**; si se utiliza el método **Update** se llama al comando **INSERT**, **UPDATE** o **DELETE** para cada fila modificada. Es posible establecer explícitamente estos comandos con el fin de controlar las instrucciones que se utilizan en tiempo de ejecución para resolver cambios, incluido el uso de procedimientos almacenados.

Comencemos con un sencillo ejemplo, vamos con una aplicación Windows con un control de cuadrícula DataGridView y dos botones:



El código será el siguiente, verás que he creado un datable a nivel global fuera de los eventos de los dos botones.

```
Imports System.Data.SqlClient

2 referencias
Public Class Form1
    Private datos_empleados As New DataTable
    Private cadena_conexion As String = _
        "Data Source=SERVIDOR\SQLEXPRESS;Initial Catalog=northwind;Integrated Security=True"
    0 referencias
    Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles btnConectar.Click
        DataGridView1.DataSource = datos_empleados
    End Sub

    0 referencias
    Private Sub btnRellenar_Click(sender As Object, e As EventArgs) Handles btnRellenar.Click
        Using conexion As SqlConnection = New SqlConnection(cadena_conexion)
            Dim comando As SqlCommand = conexion.CreateCommand()
            comando.CommandText = "Select LastName, FirstName from Employees"
            Dim adaptador As New SqlDataAdapter(comando)
            adaptador.Fill(datos_empleados)
        End Using
    End Sub
End Class
```

El método "Fill" del adaptador es el que rellena los dataset de datos con una tabla. La secuencia "Using conexion... End Using" ya la conoces, dentro de esas líneas estará vigente la conexión y fuera de ellas estará destruida.

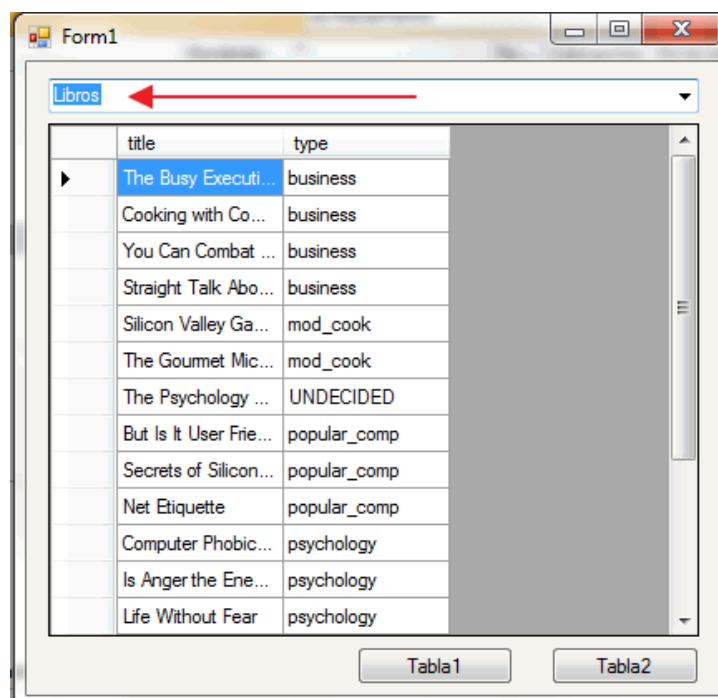
Sabemos que los adaptadores se componen de unos comandos que utilizarán para las operaciones con las bbdd y, como hemos dicho antes, será el método "Fill" para rellenar un adaptador de datos. Luego creamos un comando "conexion.createcomand" y le asignamos una instrucción SQL. Ahora es cuando deberías revisar el esquema que te mostré antes de los objetos en el mundo desconectado de los dataset y adaptadores. Fíjate en la jerarquía: el adaptador lo rellenamos con un comando que a su vez está definido a partir de la conexión.

El método Fill sirve tanto para llenar un Dataset como, en nuestro ejemplo, un datatable. Que es lo que hemos hecho en nuestro sencillo ejemplo porque sabemos que solo va a haber una tabla, sino tendríamos que haber rellenado el dataset con las tablas que hubiéramos querido con este mismo método "Fill". Te habrás dado cuenta de que no hemos abierto ninguna conexión como en otros ejemplos. Es el adaptador quien abre la conexión, realiza la consulta y la cierra inmediatamente.

14.1. Rellenar Dataset con varias tablas

En este caso vamos a rellenar el Dataset con dos consultas. A la tabla de autores y la de publicaciones, de momento las pondremos al realizar el comando.

El resultado va a ser este, donde podremos ver una tabla u otra del Dataset:



Así que pondremos un cuadro desplegable, un control de cuadrícula y dos botones para rellenar dos tablas cualesquiera. Este será el código para rellenar las dos tablas, he suprimido la parte ya repetitiva de la conexión:

```
Using conexion As SqlConnection = New SqlConnection(cadena_conexion)
    Dim comando As SqlCommand = conexion.CreateCommand()
    comando.CommandText = "Select au_fname, au_lname from authors"
    Dim adaptador As New SqlDataAdapter(comando)
    adaptador.Fill(datos, "Autores")
End Using
actualizar_tablas()

MessageBox.Show("Tabla1 rellena en el Dataset")
```

Y para la otra tabla:

```
Using conexion As SqlConnection = New SqlConnection(cadena_conexion)
    Dim comando As SqlCommand = conexion.CreateCommand()
    comando.CommandText = "Select title,type from titles"
    Dim adaptador As New SqlDataAdapter(comando)
    adaptador.Fill(datos, "Libros")
End Using
actualizar_tablas()
```

Cuando rellenamos el Dataset estamos llamando a un procedimiento llamado "actualizar_tablas" que rellena el cuadro desplegable con las tablas que se encuentre en el Dataset:

```
Sub actualizar_tablas()
    lista_tablas.Items.Clear()
    For Each tabla As DataTable In datos.Tables
        lista_tablas.Items.Add(tabla.TableName)
    Next
End Sub
```

Finalmente cuando se selecciona un elemento de la lista actualizaremos el enlace de la cuadrícula con una tabla u otra:

```
Private Sub lista_tablas_SelectedIndexChanged(ByVal sender As System.Object,
    DataGridView1.DataSource = datos.Tables(lista_tablas.SelectedIndex)
End Sub
```

15. Ordenar, buscar y filtrar

Como en los casos anteriores vemos de forma resumida cómo ordenar, buscar y filtrar nuestros conjuntos de resultados para poder mejorar nuestras interfaces y las salidas de datos a los usuarios.

15.1. Trabajar con datatables

Sabemos que un datatable es una tabla de datos en memoria que puede rellenarse como el resultado de cruzar varias tablas. O simplemente con una tabla en la que hemos aplicado algunos filtros en la propia sentencia SQL, por ejemplo:

```
Select * from titles where idtitle=1  
Select * from titles where name like '%the%'  
Select title desc from titles where idtitle>10 order by idtitle
```

Ya que estamos trabajando con datos desconectados deberíamos tener la facilidad de realizar ciertas operaciones con las tablas de nuestro Dataset. Si no dispusiéramos de estos métodos habría que realizar una consulta a la bbdd para obtener los mismos datos ordenados por otra columna, por ejemplo. Y esto sería una pérdida de tiempo si pudiéramos hacerlo internamente en nuestros Dataset.

Además sabemos que podemos hacer modificaciones en estos dataset locales así que una propiedad de cada fila, RowState nos indicará si ha sufrido modificaciones respecto al original. Comencemos por buscar dentro del dataset:

Buscar filas

Muchas veces necesitaremos buscar dentro de las tablas de nuestro dataset. En una base de datos utilizaríamos una sentencia SQL pero en nuestros objetos utilizaremos el método **Find** preparado para este trabajo. Este método de búsqueda necesita obligatoriamente que la tabla tenga una clave primaria. Veamos un sencillo ejemplo, por un lado el proceso de búsqueda con el identificador de la fila. Además y como es obligatorio que exista una clave primaria la añadimos antes de hacer la búsqueda:

```
' Tabla es un datatable con datos  
tabla.primarykey = New DataColumn () {tabla.Columns ("idproveedor")}
```

```
Dim dr as DataRow=tabla.rows.find ("2")  
if dr IsNot Nothing then  
    Muestra_resultado  
end if
```

Y mostramos el resultado:

```
Sub muestra_resultado()  
    For each dc as DataColumn in dr.Table.Columns  
        Console.Write (dr(dc) & " ")  
    Next  
End sub
```

Seleccionar varias filas

Cuando no consultamos sobre un índice único, como el caso anterior, lo normal es que tengamos una consulta que nos devuelva varios resultados como por ejemplo:

```
Select * from titles where name like '%the%'
```

En nuestra consulta en las tablas de los datasets lo haremos de una forma similar utilizando el método "Select":

```
Dim drs() as DataRow=tabla.Select ("name like '%the%'")
```

Que dentro de una consulta en el código quedaría:

```
Sub main()  
    Dim tabla As DataTable = ... 'la cargamos del dataset  
    Dim drs() as DataRow=tabla.Select ("name like '%the%'")  
    If drs is not nothing then  
        For each dr as DataRow in drs  
            Muestra_datos()  
        Next  
    End If  
End Sub
```

Si queremos ordenar el resultado, este método tiene una sobrecarga para indicarlo:

```
Dim drs() as DataRow=tabla.Select ("name like '%the%'", "name ASC")
```

Que nos ordenaría de forma ascendente con nombre el resultado de buscar todos los que tengan el literal "the" en el nombre

Calcular columnas

En ocasiones necesitaremos hacer algunos cálculos en las filas de nuestras tablas de los dataset. Por ejemplo en una sentencia SQL sería:

```
Select idproducto, Precio, IVA, (precio + precio * IVA) as Preciototal
```

Como no podemos utilizar el símbolo "*" para realizar la operación directamente al tratarse de un carácter reservado (indica "todas las columnas") tendremos que hacer uso de:

```
Dim preciototal as DataColumn=New DataColumn ("PrecioTotal")  
preciototal.Expresion="Precio + precio * IVA"  
tabla.Columns.Add (preciototal)
```

Simplemente hemos añadido una columna más a nuestra tabla con la operación.

Funciones de agregado

Son las funciones típicas de SQL, como:

```
Select Sum (precio) from libros
```

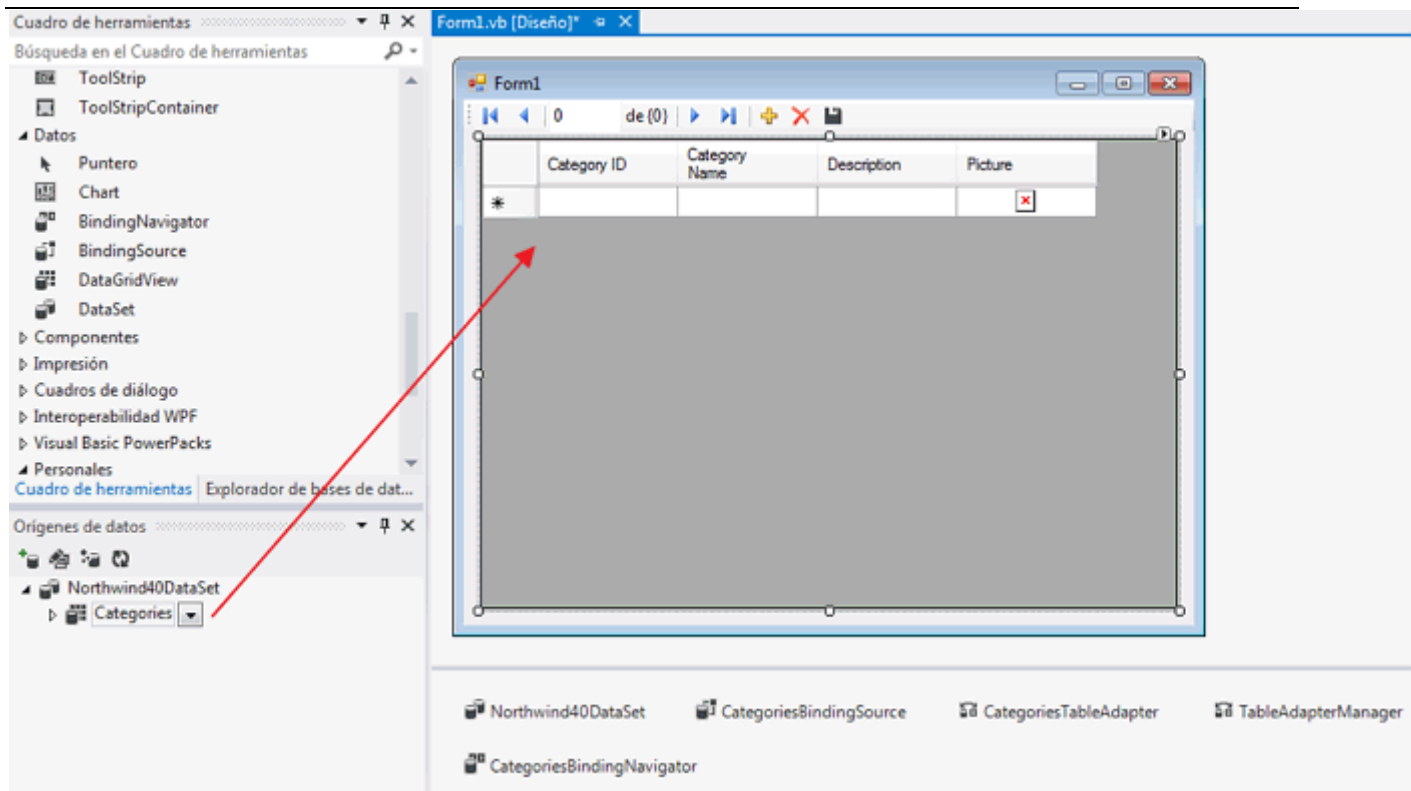
Para estas operaciones también tendremos un método adecuado que se llama "Compute":

```
Dim precio as string= productos.Compute ("Sum (Precio)","").ToString()
```

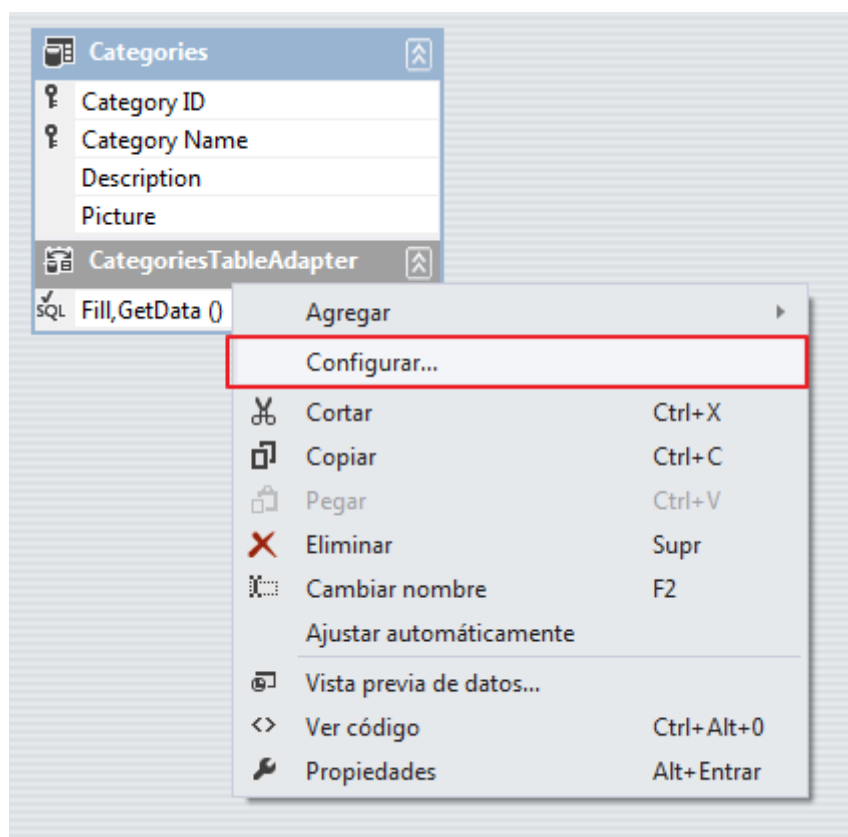
16. Actualizar datos

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.

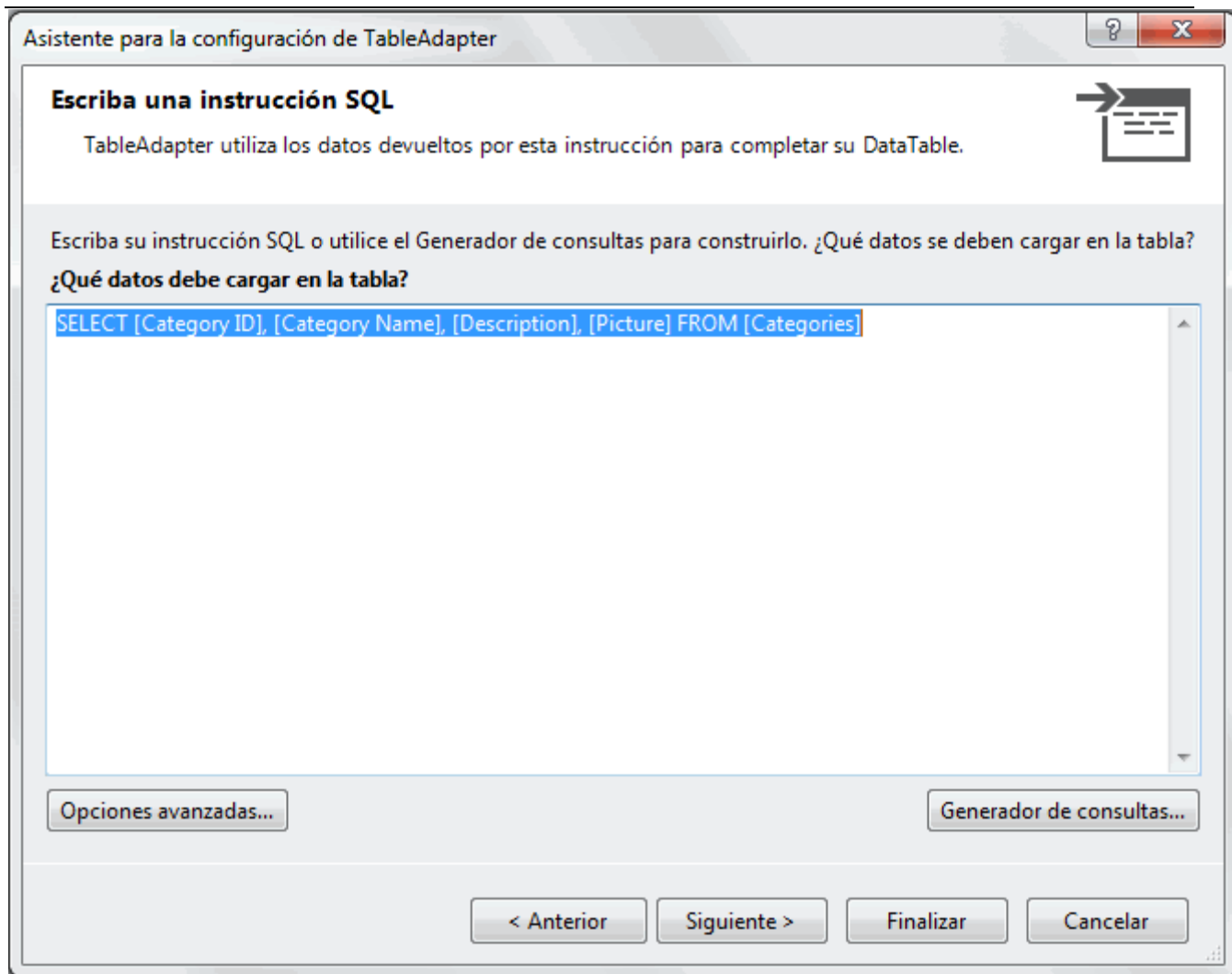
Vamos a repetir como se generan los comandos de actualización con el IDE. Partimos de una tabla sencilla y la arrastramos a nuestro IDE, por ejemplo, la de Categorías.



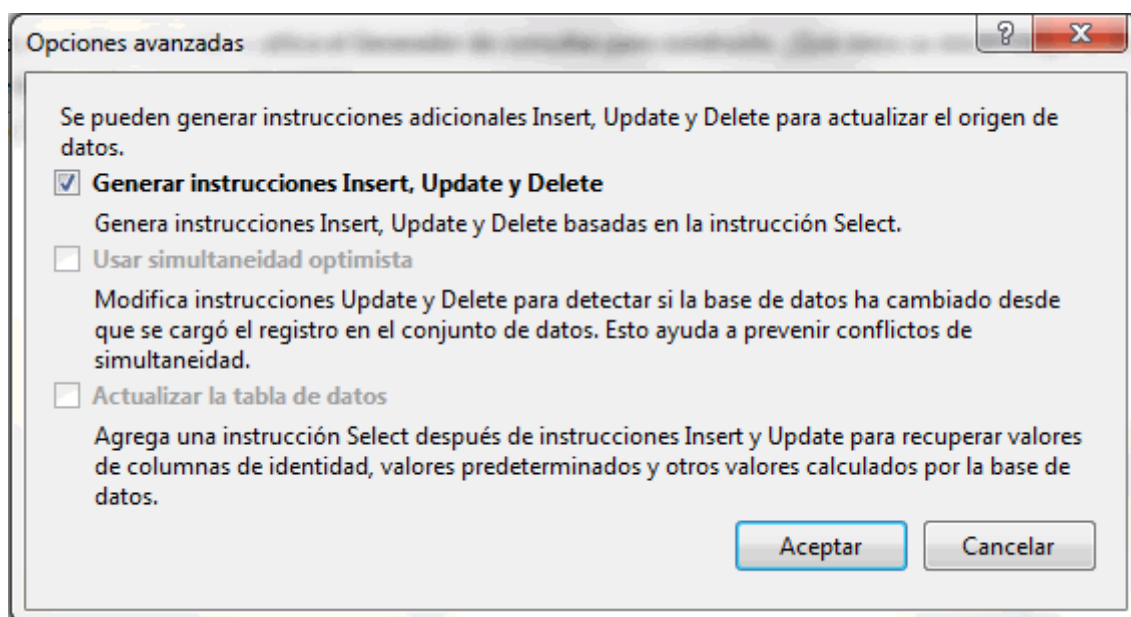
Si vemos los detalles sabemos que nos ha añadido un Dataset:



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:

Asistente para la configuración de TableAdapter

Elija los métodos que se van a generar

Los métodos de TableAdapter cargan y guardan los datos entre su aplicación y la base de datos.

¿Qué métodos desea agregar a TableAdapter?

☒ **Rellenar un DataTable**

Crea un método que toma un DataTable o DataSet como parámetro y ejecuta la instrucción SQL o el procedimiento almacenado SELECT introducido en la página anterior.

Nombre de método:

☒ **Devolver un DataTable**

Crea un método que devuelve una nueva DataTable con los resultados de la instrucción SQL o procedimiento almacenado SELECT introducido en la página anterior.

Nombre de método:

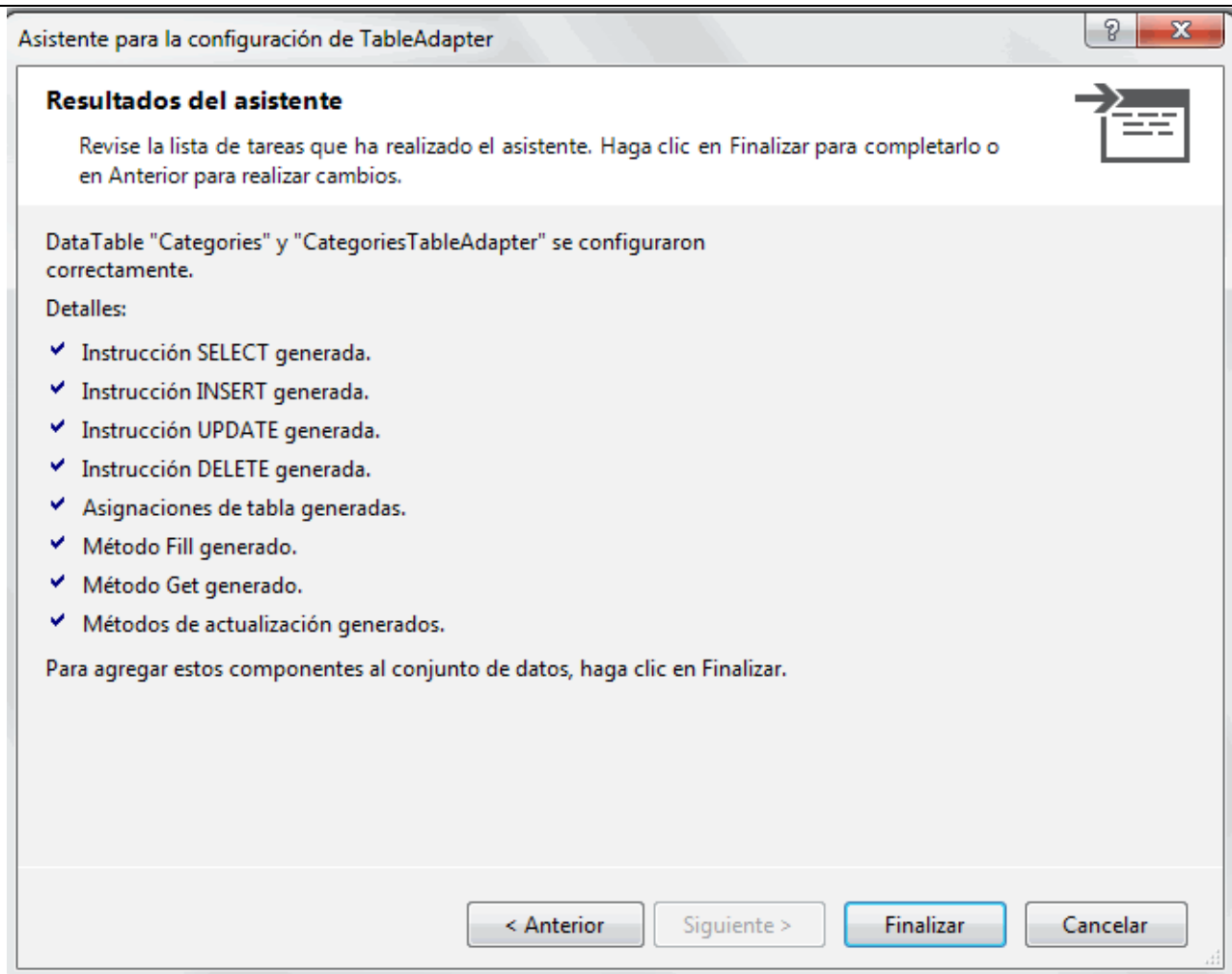
☒ **Crear métodos para enviar actualizaciones directamente a la base de datos (GenerateDBDirectMethods)**

Crea métodos Insert, Update y Delete que se pueden llamar para enviar cambios de filas individuales directamente a la base de datos.

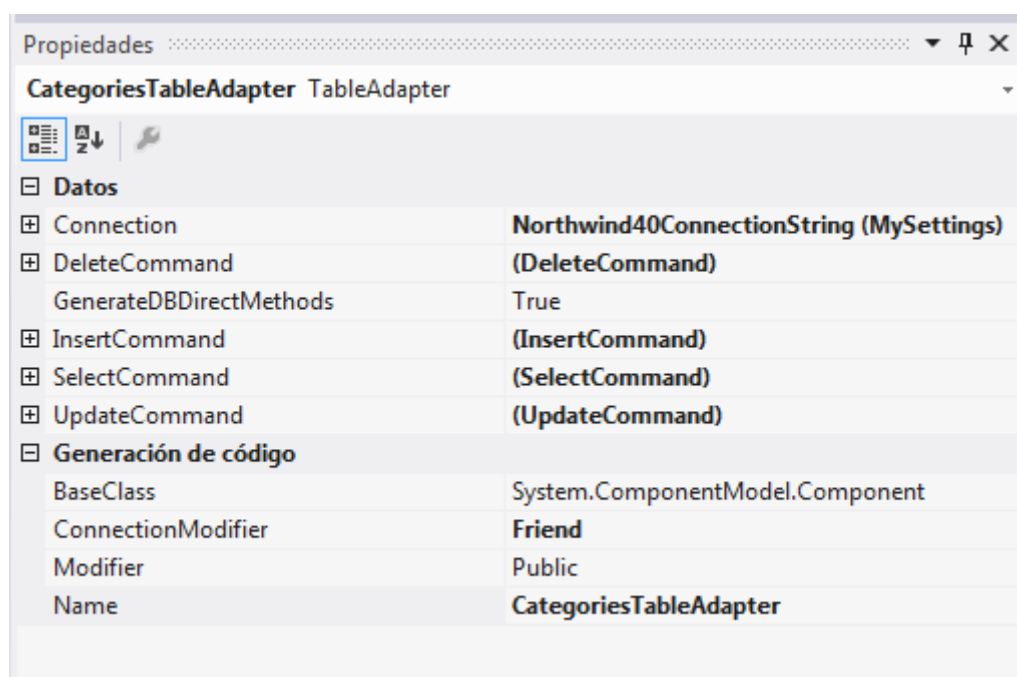
< Anterior Siguiete > **Finalizar** Cancelar

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:



Ahora veremos el equivalente a hacer todo esto por código, como hemos visto ya en los casos anteriores.

16.1. Generar los comandos

Tenemos que generar los comandos para que nuestro adaptador vuelque los datos según se hayan realizado las operaciones de inserción, modificación o borrado en nuestro Dataset. Pero resulta que como ADO.NET tiene de todo pues también nos provee de un generador de comandos, que no es ni más ni menos que un objeto que crea los comandos SQL automáticamente.

```
Dim MiCommandBuilder As SqlCommandBuilder = New SqlCommandBuilder(MiAdapter)
```

Esta simple línea crea los comandos: Insert, Update y Delete en MiAdapter...

```
Dim SqlString As String
'Parametros:
' data source = Nombre del Servidor (puede ser dirección IP)
' initial catalog = Nombre de la base de datos
' integrated security = Indica que usa Seguridad integrada
SqlString = ".....

'Se crea la conexión
Dim MiConexion As SqlConnection = New SqlConnection(SqlString)

'Se declara el Adaptador que utilizará el Select como SelectCommand
'y MiConexión como conexión
Dim MiAdapter As SqlDataAdapter = New SqlDataAdapter("Select * from Customers", MiConexion)
'El commandBuilder agrega los comandos de Insert, Update y Delete
Dim MiCommandBuilder As SqlCommandBuilder = New SqlCommandBuilder(MiAdapter)

Dim MiDataSet As DataSet
'Y por último Fill llena MiDataSet
MiAdapter.Fill(MiDataSet, "MiTabla")
```

Al estar ya los comandos implementados, podemos realizar todos los cambios que queramos. Cuando activemos la opción de sincronizar se volcarán gracias a esos comandos. Sólo como curiosidad veamos lo que tendríamos que haber escrito para crear un solo comando a mano:

```
Dim SqlString As String
    SqlString = "..."
    'Se crea la conexión
    Dim MiConexion As SqlConnection = New SqlConnection(SqlString)

    'Se declara el Adaptador
    Dim MiAdapter As SqlDataAdapter = New SqlDataAdapter
    'Declare 4 Comandos 1 para cada transacción
    Dim SqlSelectCommand1 As SqlCommand = New SqlCommand
    Dim SqlInsertCommand1 As SqlCommand = New SqlCommand
    Dim SqlUpdateCommand1 As SqlCommand = New SqlCommand
    Dim SqlDeleteCommand1 As SqlCommand = New SqlCommand

    'Pego cada comando a la transaccion que hará en el Adaptador
    MiAdapter.DeleteCommand = SqlDeleteCommand1
    MiAdapter.InsertCommand = SqlInsertCommand1
    MiAdapter.SelectCommand = SqlSelectCommand1
    MiAdapter.UpdateCommand = SqlUpdateCommand1

    'Configuro el SqlSelectCommand1 para que haga el SELECT
    SqlSelectCommand1.CommandText = "SELECT CustomerID, CompanyName, ContactName " & _
        "FROM Customers"
    'Le pego la conexión
    SqlSelectCommand1.Connection = MiConexion
    'Configuro el SqlInsertCommand1 para que inserte los nuevos campos
    SqlInsertCommand1.CommandText = "INSERT INTO Customers(CustomerID, CompanyName,
ContactName) " & _
        "VALUES (@CustomerID, @CompanyNa, @ContactName); " & _
        "SELECT CustomerID, CompanyName, ContactName, ContactTitle " & _
        "FROM Customers WHERE (CustomerID = @CustomerID)"
    'Le pego la conexión
    SqlInsertCommand1.Connection = MiConexion
    'Declare 3 parametros uno por cada valor que le voy a pasar al comando
    'Por supuesto si uso el metodo Adapter.Update este los pasa automáticamente
    SqlInsertCommand1.Parameters.Add(New SqlParameter("@CustomerID", SqlDbType.NVarChar, 5,
"CustomerID"))
    SqlInsertCommand1.Parameters.Add(New SqlParameter("@CompanyName", SqlDbType.NVarChar,
40, "CompanyName"))
    SqlInsertCommand1.Parameters.Add(New SqlParameter("@ContactName", SqlDbType.NVarChar,
30, "ContactName"))
```

Estos son los comandos creados de forma manual, pero solo están puestos los de seleccionar, Select, y el de Insertar, Insert. Faltarían todavía el de Borrar y actualizar, así que fíjate como ese "commandbuilder" nos ayuda.

Todo lo que nos quedaría sería la actualización de los datos:

```
'Y para actualizar  
MiAdapter.Update(MiDataSet)
```

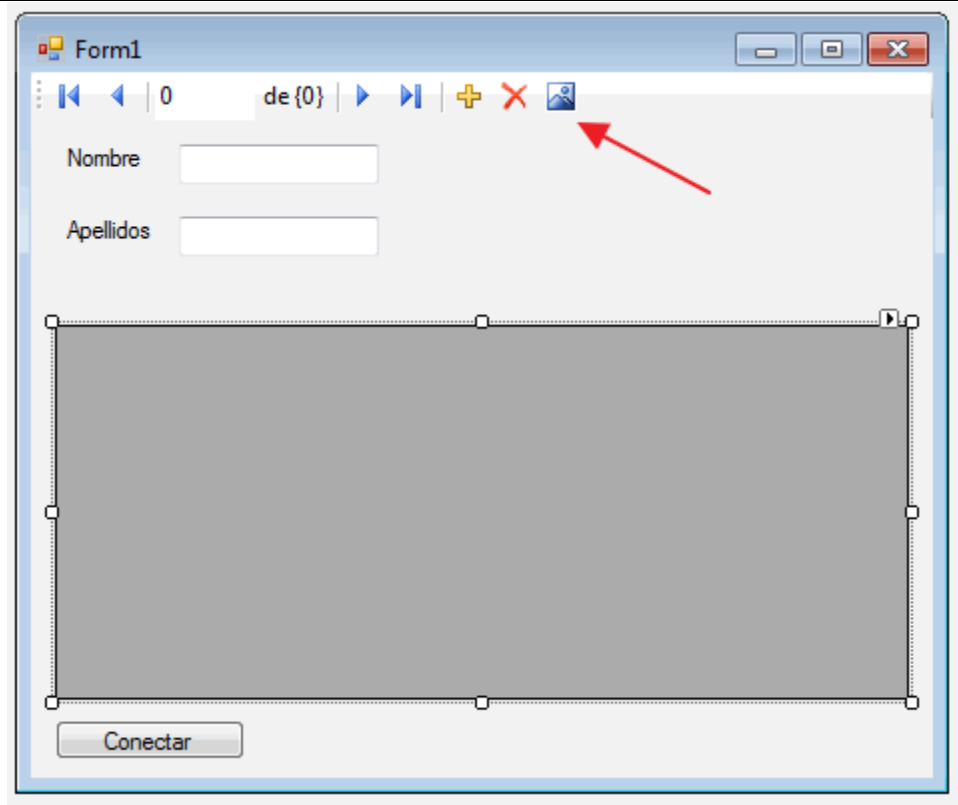
Que provocaría la sincronización volcando los datos al servidor de base de datos. Vamos a recordar lo que ponía el asistente que genera los formularios de datos cuando se pulsa el botón de "grabar":

```
Private Sub CategoriesBindingNavigationSaveItem_Click (....  
    Me.Validate()  
    Me.CategoriesBindingSource.EndEdit()  
    Me.TableAdapterManager.UpdateAll (Me.categoriasDataset)  
End Sub
```

Que es una de las formas de hacerlo:

- Me.Validate. Comprobamos el valor del último elemento que ha perdido el enfoque. Si está activada la validación se disparan los eventos de validación por si el usuario quiere hacer una validación de datos antes de ejecutar la siguiente instrucción.
- Me.CategoriesBindingSource.EndEdit. Este método confirma cualquier cambio que esté en proceso en cualquier control con enlace a datos que se esté editando actualmente. Por consiguiente, si un control con enlace a datos todavía tiene el foco y se hace clic en el botón Guardar, todas las ediciones pendientes en ese control se confirmarán antes de que se guarde (método TableAdapterManager.UpdateAll).
- Me.TableAdapterManager.UpdateAll (Me.NorthwindDataSet). Envía las actualizaciones de todas las tablas de NorthwindDataSet a la base de datos del servidor.

Si modificamos nuestro ejemplo anterior para añadirle un botón que grabe los datos:



Basta con que pongamos lo siguiente, por un lado el generador de comandos para que el adaptador los ejecute al sincronizar:

```
'Enlazamos los cuadros de texto:
txt_nombre.DataBindings.Add(New Binding("Text", enlace, "au_fname", True))
txt_apellidos.DataBindings.Add(New Binding("Text", enlace, "au_lname", True))

'Generamos los comandos para que se puedan actualizar los datos en el servidor de base de
Dim comandos As SqlCommandBuilder = New SqlCommandBuilder(adaptador)
```

Como ves, he puesto al final que me genere los comandos automáticamente con SqlCommandBuilder que ahora comentaremos un poco más. Por otro lado, en el botón que hemos añadido solo pondremos:

```
Private Sub ToolStripButton1_Click(ByVal sender As System.Object, ByVal e As
    adaptador.Update(resultado.Tables(0))
End Sub
```

Que mandaría todos los cambios de la tabla de datos al servidor con lo que tendremos terminada toda la explotación completa.

16.2. CommandBuilder

El requisito mínimo para que la generación automática de comandos funcione correctamente consiste en establecer la propiedad `SelectCommand`. El esquema de tabla que recupera `SelectCommand` determina la sintaxis de las instrucciones `INSERT`, `UPDATE` y `DELETE` generadas automáticamente.

`CommandBuilder` debe ejecutar `SelectCommand` con el objeto de poder devolver los metadatos necesarios para construir los comandos de inserción, actualización y eliminación. Por eso es necesario realizar un viaje adicional al origen de datos, con el consiguiente efecto adverso en el rendimiento. Para mejorar el rendimiento, debe especificar los comandos de forma explícita, en lugar de utilizar `CommandBuilder`.

`SelectCommand` debe además devolver al menos una clave principal o columna única. Si no hay ninguna, se inicia una excepción `InvalidOperationException` y no se genera ningún comando.

Cuando se asocia con un `DataAdapter`, `CommandBuilder` genera automáticamente las propiedades `InsertCommand`, `UpdateCommand` y `DeleteCommand` del `DataAdapter`, si son referencias nulas. Si ya existe algún comando `Command` para una propiedad, se usará ese.

En la tabla siguiente se muestran las reglas de la generación automática de comandos.

Comando	Regla
InsertCommand	Inserta una fila en el origen de datos para cada fila de la tabla cuyo estado <code>RowState</code> sea <code>DataRowState.Added</code> . Inserta valores en todas las columnas que se pueden actualizar (excepto en columnas como identidades, expresiones y marcas de hora).
UpdateCommand	Actualiza filas en el origen de datos para todas las filas de la tabla cuyo estado <code>RowState</code> sea <code>DataRowState.Modified</code> . Actualiza los valores de todas las columnas, excepto de las columnas que no son actualizables, como identidades o expresiones. Actualiza todas las filas en las que los valores de columna en el origen de datos coinciden con los valores de la columna de clave principal de la fila, siempre que las restantes columnas del origen de datos coincidan con los valores originales de la fila. Para obtener más información, vea la sección "Modelo de simultaneidad optimista para actualizaciones y eliminaciones" de este mismo tema.
DeleteCommand	Elimina las filas del origen de datos que corresponden a todas las filas de la tabla con un estado <code>RowState</code> de <code>DataRowState.Deleted</code> . Elimina todas las filas en las que los valores de columna en el origen de datos coinciden con los valores de la columna de clave principal de la fila, siempre que las restantes columnas del origen de datos coincidan con los valores originales de la fila. Para obtener más información, vea la sección "Modelo de simultaneidad optimista para actualizaciones y eliminaciones" de este mismo tema.

La generación automática de comandos tiene las siguientes limitaciones.

Sólo tablas no relacionadas

La lógica de generación automática de comandos crea instrucciones INSERT, UPDATE o DELETE para tablas independientes sin tener en cuenta las relaciones que éstas puedan tener con otras tablas en el origen de datos. Por eso se puede producir un error al llamar a Update para realizar cambios en una columna que participa en una restricción de clave externa en la base de datos. Para evitar esa excepción, no use CommandBuilder al actualizar las columnas que participan en una restricción de clave externa. En este caso debe especificar de forma explícita las instrucciones que se van a utilizar para llevar a cabo la operación.

Nombres de tabla y columna

La lógica de generación automática de comandos ocasiona un error cuando los nombres de las tablas o columnas incluyen algún carácter especial, como espacios, puntos, signos de exclamación y otros caracteres no alfanuméricos, aún en el caso de que se incluyan entre corchetes.

Si no nos es suficiente con este generador de comandos tendremos que hacerlos de forma manual. Esto entra ya dentro del mundo avanzado, bueno, bastante de lo que hemos visto aquí pero así tienes un punto de partida para avanzar en bbdd. Imagina que este curso hubiera terminado en el capítulo anterior, no sabrías hacer prácticamente nada por código, y eso es algo impensable al personalizar los programas.

Si con la generación automática de comandos no tenemos suficiente recuerda que podemos ejecutar instrucciones SQL directamente mediante el objeto de conexión. Por lo que podríamos lanzar una instrucción de "insert", "update" o "delete". Y si lo que queremos es poner los comandos personalizados en el adaptador tendremos que utilizar el método que hemos visto hace un par de apartados.

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:

Form1

2 de 8

Categoría: Condiments

Descripción: Sweet and savory s

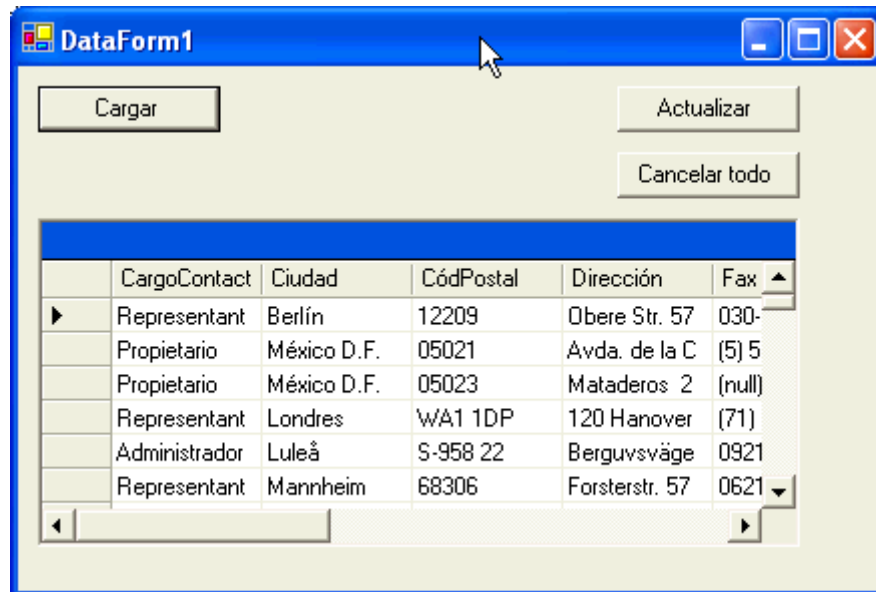
Imagen: 

	Categoría	Descripción
	Beverages	Soft drinks, coffees, teas, beer, and ale
▶	Condiments	Sweet and savory sauces, relishes, spreads, and s
	Confections	Desserts, candies, sweetbreads
	Dairy Products	Cheeses
	Grains/Cereals	Breads, crackers, pasta, and cereal
	Meat/Poultry	Prepared meats
	Produce	Dried fruit and bean curd
	Seafood	Seaweed and fish
*		

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 Neptuno.



	CargoContact	Ciudad	CódPostal	Dirección	Fax
►	Representant	Berlín	12209	Obere Str. 57	030-
	Propietario	México D.F.	05021	Avda. de la C	(5) 5
	Propietario	México D.F.	05023	Mataderos 2	(null)
	Representant	Londres	WA1 1DP	120 Hanover	(71)
	Administrador	Luleå	S-958 22	Berguvsväge	0921
	Representant	Mannheim	68306	Forsterstr. 57	0621

Realiza el código manualmente.

Ejercicio 3

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



	EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
►	1	Davolio	Nancy	Sales Represent...	Ms.	08/12/1948
	2	Fuller	Andrew	Vice President, S...	Dr.	19/02/1952
	3	Leverling	Janet	Sales Represent...	Ms.	30/08/1963
	4	Peacock	Margaret	Sales Represent...	Mrs.	19/09/1937
	5	Buchanan	Steven	Sales Manager	Mr.	04/03/1955
	6	Suyama	Michael	Sales Represent...	Mr.	02/07/1963
	7	King	Robert	Sales Represent...	Mr.	29/05/1960
	8	Callahan	Laura	Inside Sales Coor...	Ms.	09/01/1958
	9	Dodsworth	Anne	Sales Represent...	Ms.	27/01/1966

Actualizar datos Eliminar seleccionado Buscar LastName

EmployeeID: Last Name: Añadir empleado First Name:

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.

PEDIDOS REALIZADOS SEGÚN CLIENTE

Company Name:

Contact Name: City:

Contact Title: Country:

Address: Phone:

	OrderID	CustomerID	OrderDate	RequiredDate	ShippedDate	ShipName	ShipCity
▶	10643	ALFKI	25/09/1995	23/10/1995	03/10/1996	Alfreds Futterkiste	Berlin
	10692	ALFKI	03/11/1995	01/12/1995	13/11/1995	Alfred's Futterkiste	Berlin
	10702	ALFKI	13/11/1995	25/12/1995	21/11/1995	Alfred's Futterkiste	Berlin
	10835	ALFKI	15/02/1996	14/03/1996	21/02/1996	Alfred's Futterkiste	Berlin
	10952	ALFKI	15/04/1996	27/05/1996	23/04/1996	Alfred's Futterkiste	Berlin
	11011	ALFKI	09/05/1996	06/06/1996	13/05/1996	Alfred's Futterkiste	Berlin
*							