Tema Gestión BD – ADOnet

Contenido

1. Introducción 2
2. Generamos la BD 2
3. Conexión con la BD 2
4. Configurar el Origen de Datos 4
3. Crear nuevas Consultas 7
   3.1. Asistente de Consulta 8
   3.2. Agregar una nueva consulta 10
4. Acceder a los datos consultados desde el código 12
5. Generar un formulario en vista Detalle de una tabla 13
6. Generar un formulario con Vista DataGridView de una tabla 16
7. Formulario de las Películas 18
   7.1. Cargar los ComboBox de Estilos y Categorías 19
   7.2. Filtro o búsqueda de películas por título 19
   7.3. Actualizar los Registros 20
   7.4. Eliminar los Registros 21
   7.5. Mostramos las Carátulas 22
8. Panel Principal 23
9. Formulario de Alquileres 24
   9.1. Alquilar y Devolver Películas 25

1. Introducción

En este tema se verá cómo crear una aplicación gestora de datos. Para ello seguiremos la realización de una práctica a lo largo del tema.

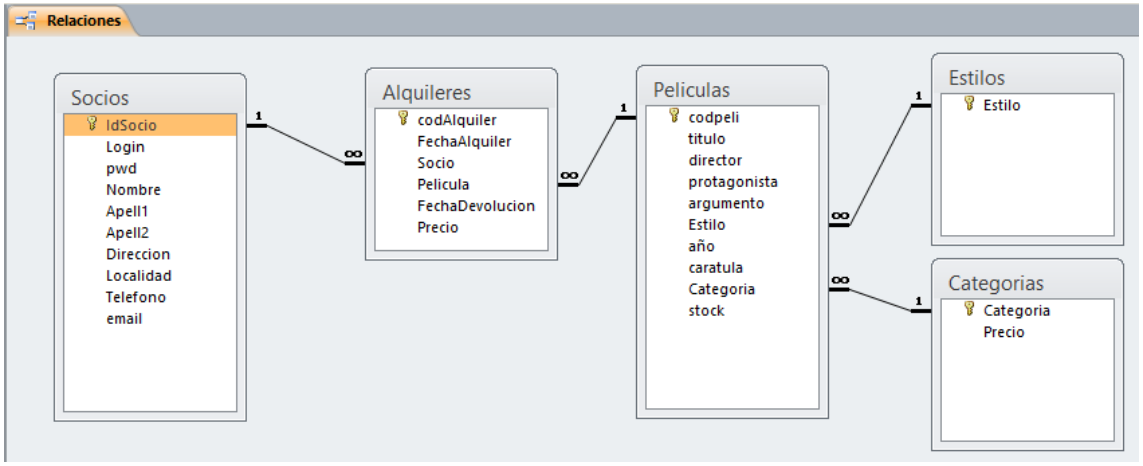
Antes de nada se deben instalar en el equipo las aplicaciones necesarias para gestionar bases de datos MySQL, así como su conector con Visual Studio, de manera que nos permita trabajar con este tipo de base de datos desde el entorno de desarrollo.

2. Generamos la BD

Vamos a crear la siguiente estructura de la BD para llevar la gestión sencilla de un video club. Por lo que gestionaremos los datos de los socios, las películas y sus préstamos.

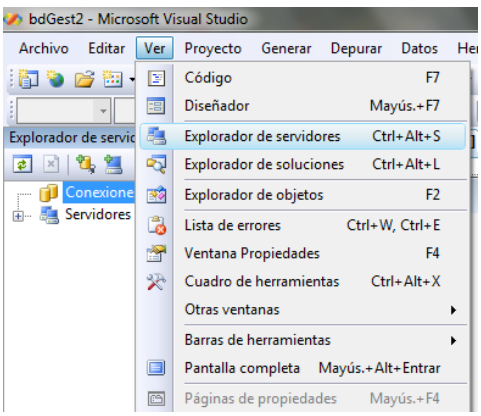
Tendremos dos tablas auxiliares para facilitarnos la clasificación de las películas, por su "Estilo" y por su "Categoría" marcar el precio de la misma.

El campo Caratula de la tabla Peliculas será de tipo texto, porque almacenará el nombre del fichero (nombreCaratula.jpg). La carpeta donde estarán almacenadas las imágenes estará local al proyecto.



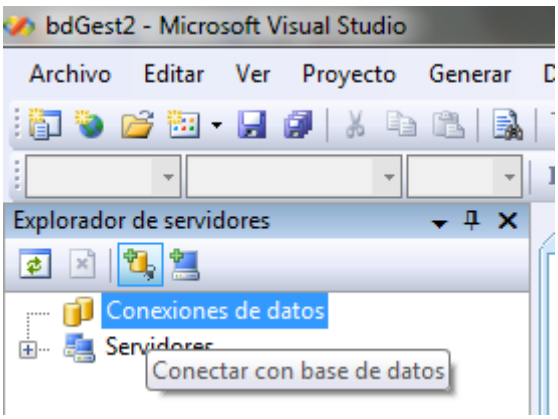
Es recomendable que antes de programar la aplicación introduzcamos algunos datos en nuestras tablas para comprobar que está bien diseñada, sus campos, el tamaño de estos, las relaciones entre las distintas tablas, que no me faltan ni me sobran tablas, etc.

3. Conexión con la BD

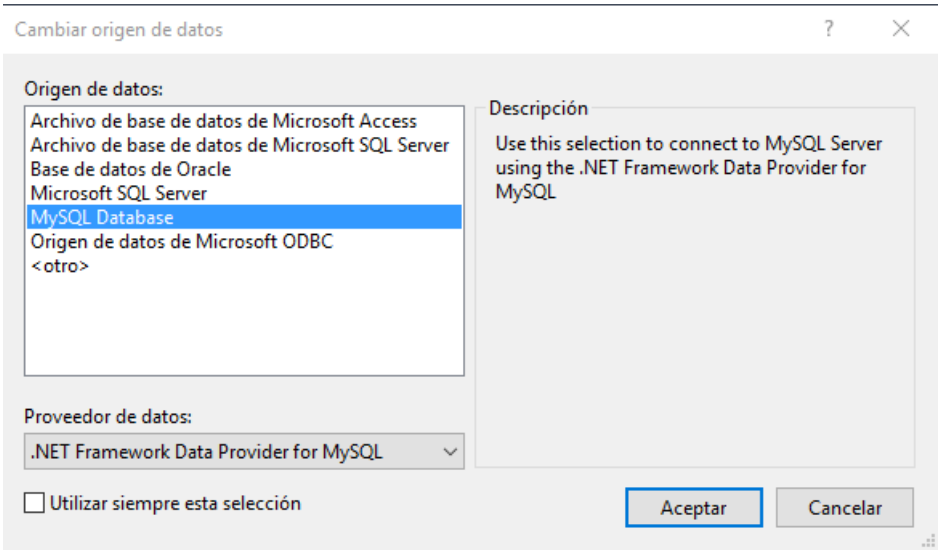


Una vez que tenemos creada la BD en MySQL, por ejemplo, lo primero es ver si podemos conectar bien con ella, para eso seguimos los siguientes pasos:

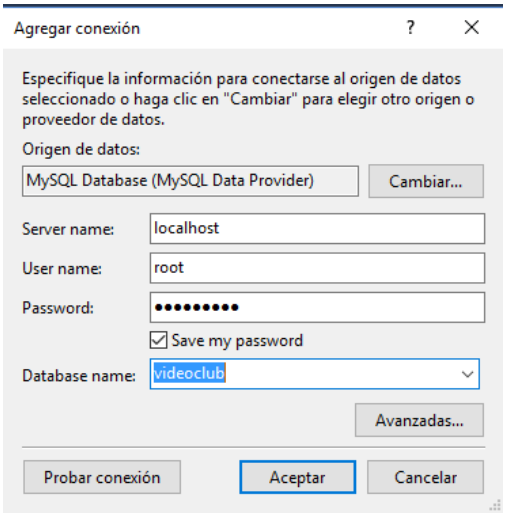
1. Visualizamos el Explorador de Servidores
2. En el Explorador de Servidores → Pinchamos en Conectar BD



3. Lo primero es seleccionar el Origen de Datos, por defecto saldrá el último que se haya elegido, pulsando en el botón Cambiar, vemos la siguiente ventana:

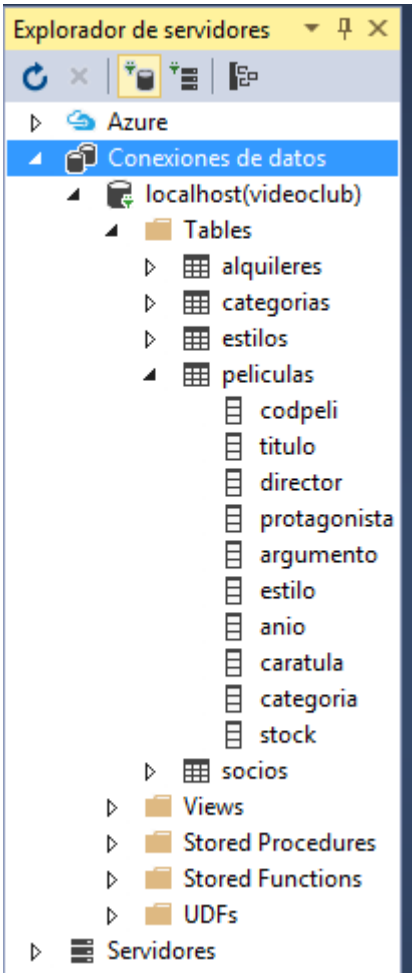


4. A continuación, enlazamos con nuestro servidor de BD MySQL, en este caso en nuestro propio equipo:

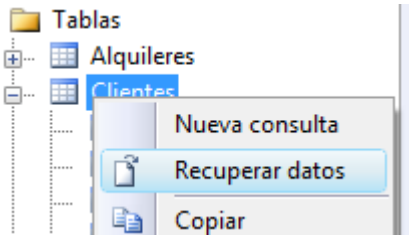


También conviene que antes de pulsar Aceptar, hagamos clic en **Probar Conexión,** para ver que todo ha ido bien.

5. Ya tenemos en nuestro Explorador de Servidores el enlace a nuestra BD.



Desde aquí ya podemos ver las tablas que tenemos, sus campos e incluso modificar sus datos:



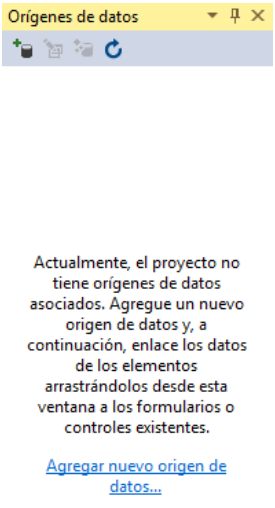
Situándonos sobre alguna de sus tablas, y en su menú contextual, pulsando en **Recuperar Datos.**

Hasta aquí lo único que hemos hecho, es incorporar a mi plataforma de Visual Studio, un enlace a una BD, para poder gestionarla también dese dentro.

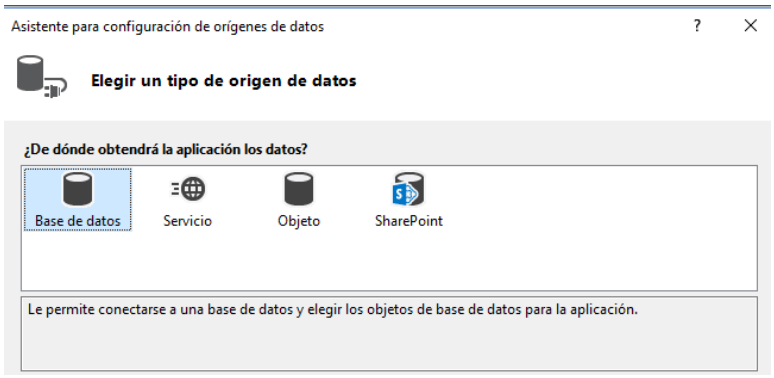
4. Configurar el Origen de Datos

Ahora para poder gestionar dicha BD desde el código, yo tengo que configurar mi origen de datos… para eso, una vez dentro de nuestro nuevo proyecto de Visual Studio:

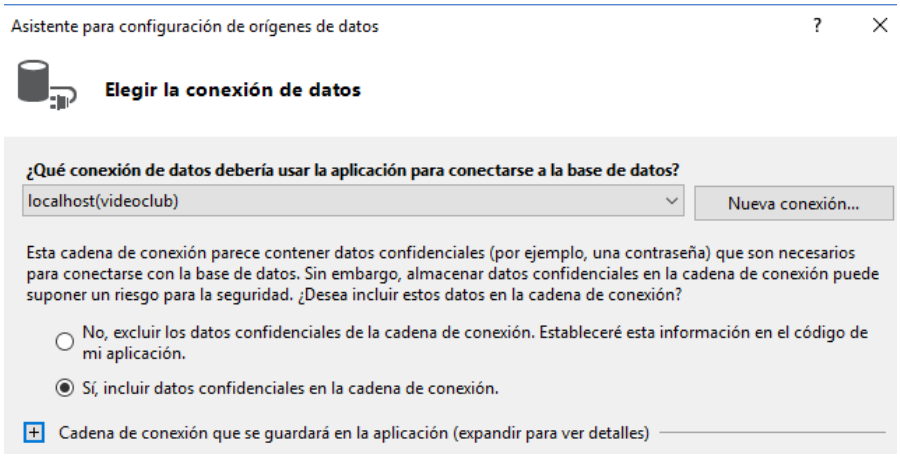
1. Lo primero es mostrar dicha ventana:



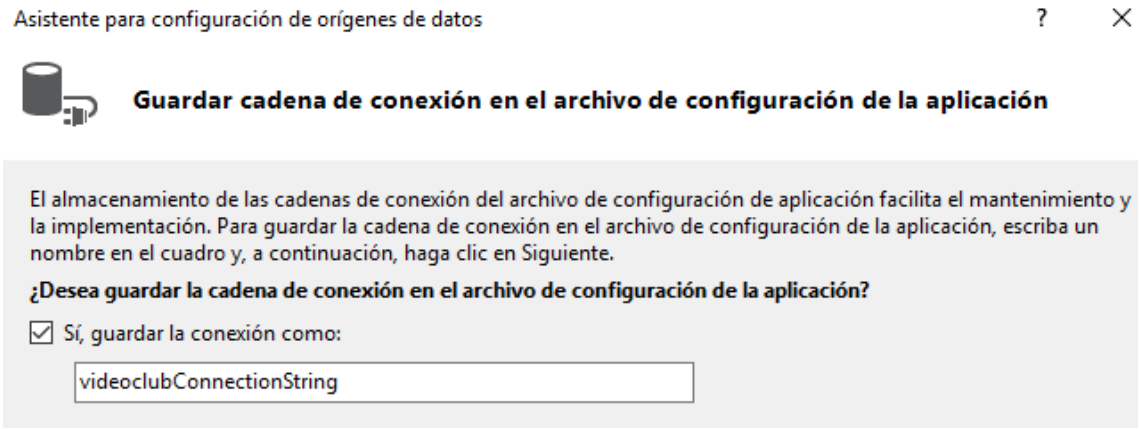
2. Pinchamos en **Agregar Nuevo Origen de Datos…** y abrimos su Asistente



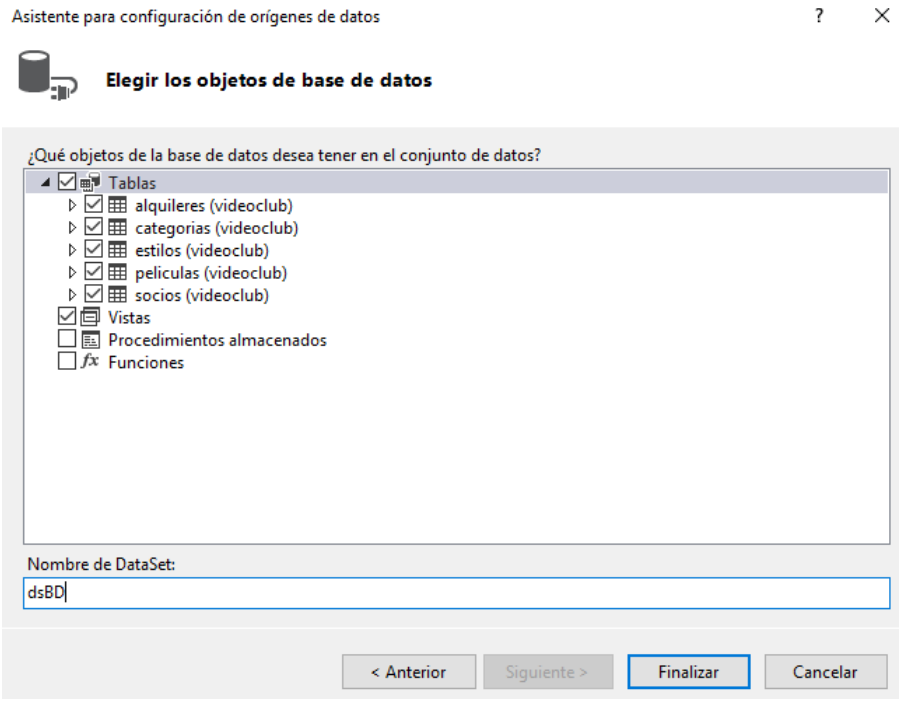
Por defecto, me saldrá ya mi BD, puesto que la recoge del Explorador de Servidores, si no fuese así, pinchamos en Nueva Conexión y elegimos la correcta:



Guardamos la cadena de conexión con los datos confidenciales (usuario y contraseña) dentro del propio proyecto (porque estamos en un proyecto de desarrollo en clase), aunque en un proyecto real que haya que entregar al cliente, se debería parametrizar estos datos en un fichero aparte.

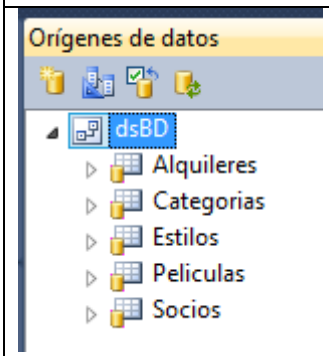
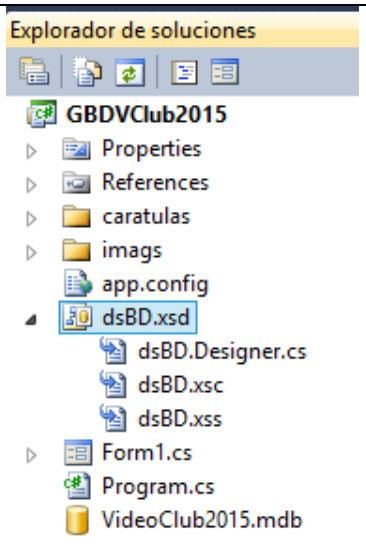


Generamos el DataSet: Eligiendo las tablas y/o vistas de mi BD, no tiene porqué ser todas, y poniéndole nombre a mi dataSet

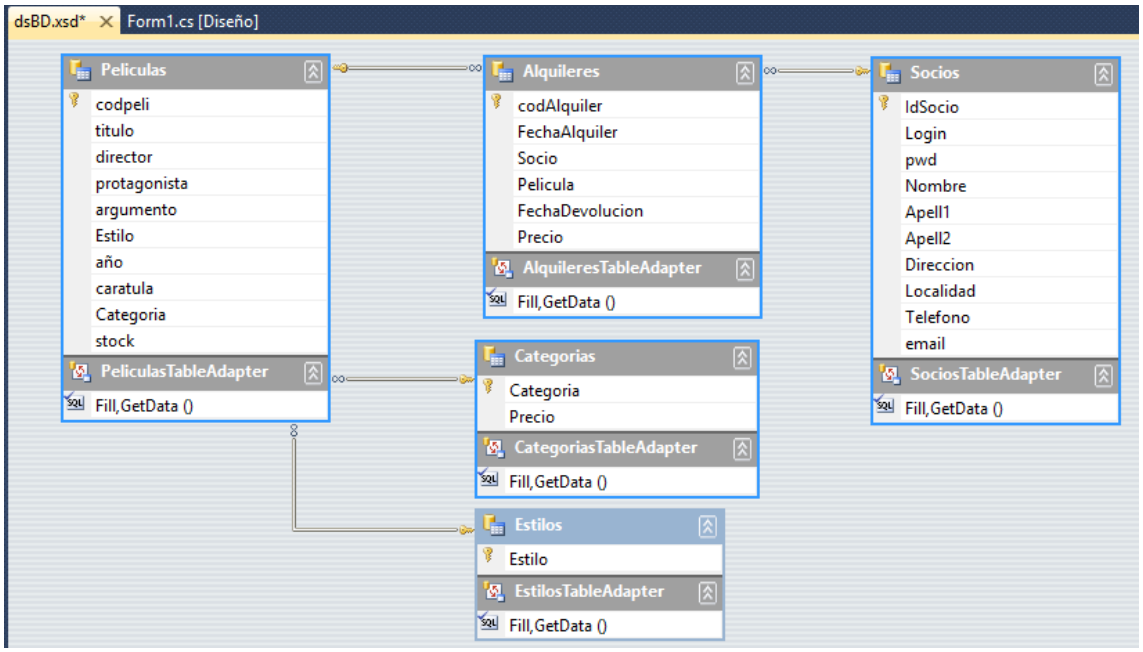


.. y Finalizamos.

3. Una vez que hemos finalizado nos encontramos dos detalles:

a. En la Ventana Orígenes de datos:	b. En el Explorador de Soluciones
	

El que nos interesa es el del Explorador de Soluciones, ya que es ahí donde se encuentra todo el código para la gestión de mi BD. Si hacemos doble clic sobre **dsBD.xsd** veré la siguiente pantalla:



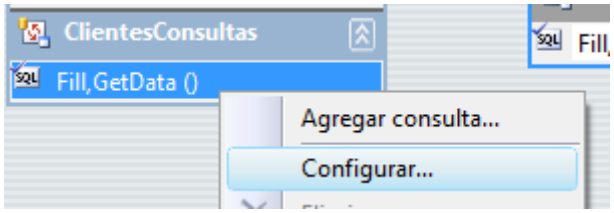
A partir de ahora, esta ventana la visitaremos mucho, ya que es donde deberemos añadir todas y cada una de las consultas que queramos utilizar sobre estas tablas.

Al origen de datos regresaremos más adelante.

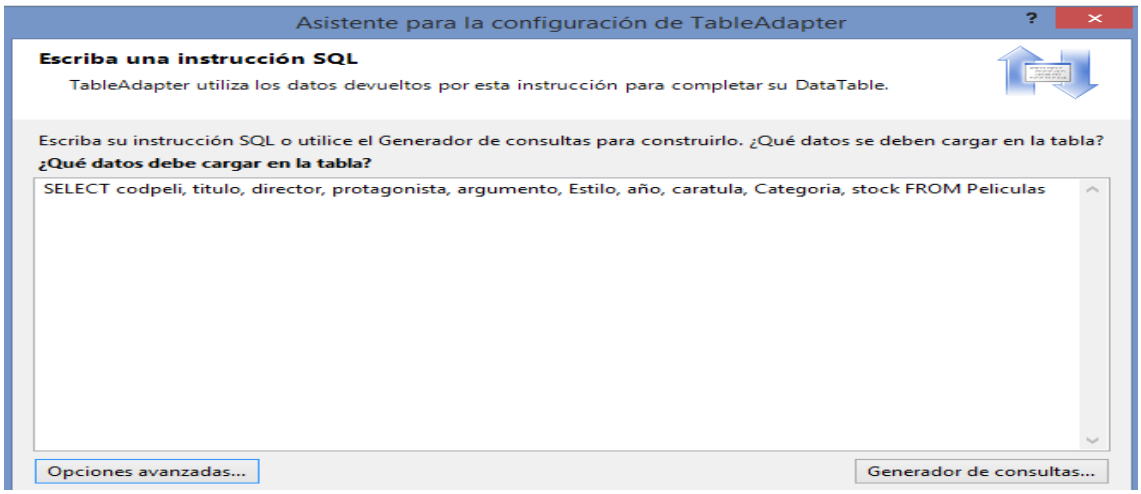
3. Crear nuevas Consultas

Podemos cambiar el nombre al objeto que va a recoger las consultas, poniendo otro más identificativo, pero tampoco es necesario y generalmente en los libros se refieren a las consultas como los “Adaptadores” de ahí el nombre.

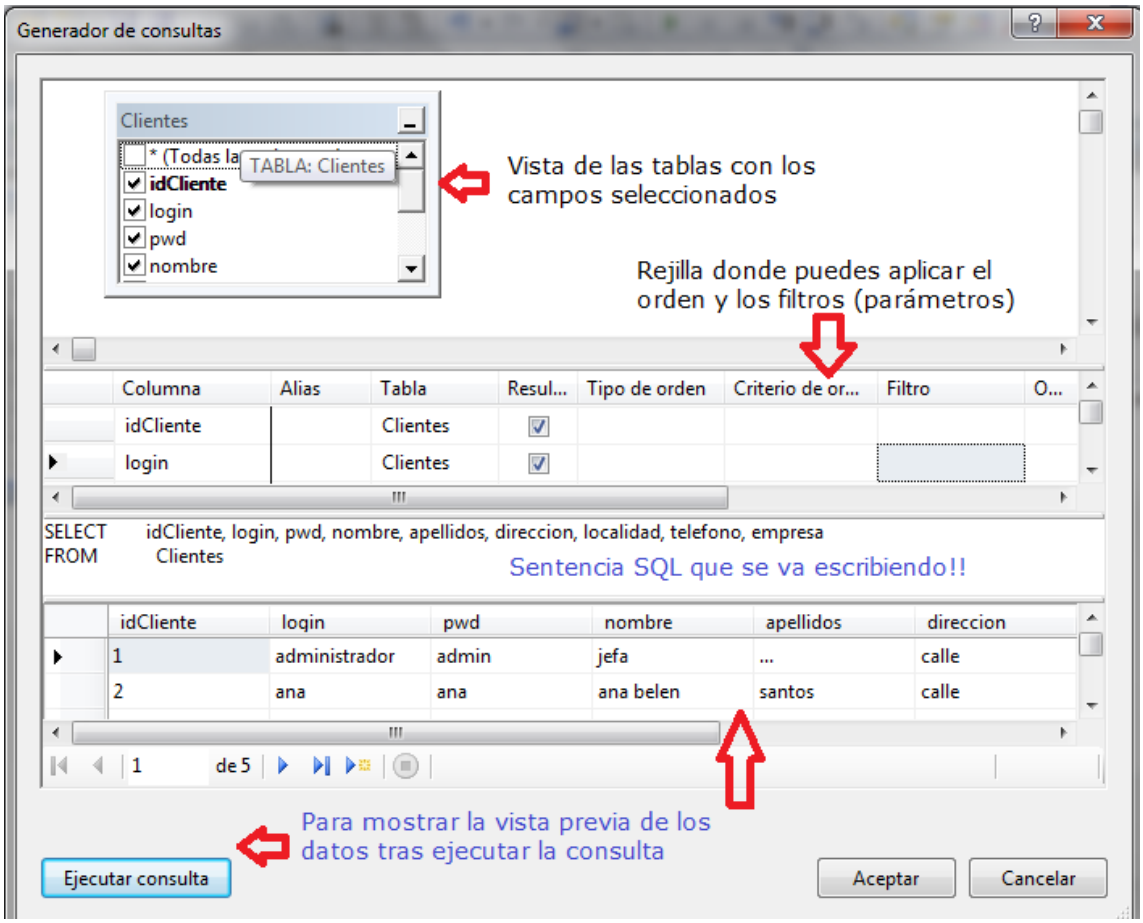
Toda tabla trae por defecto su consulta **Fill.** Para verla, y conocer el asistente para consultas, hacemos clic sobre ella con el botón derecho y seleccionamos **Configurar…**



3.1. Asistente de Consulta

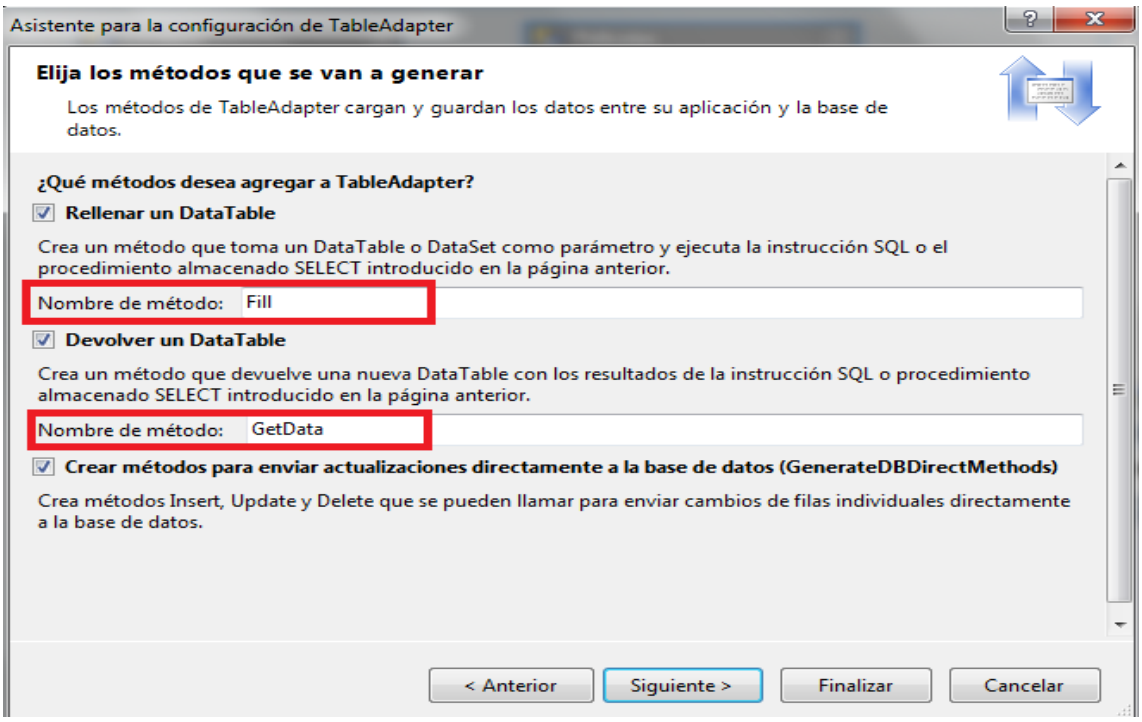


Esta es la ventana principal, y donde debe aparecer la sentencia SQL de nuestra consulta. Pero como se ve en la parte inferior hay dos botones, el de la derecha, el **Generador de Consultas…**, nos facilitará la labor de escribir la sentencia SQL, llevándonos a la siguiente ventana:



Podemos escribir la sentencia SQL como queramos, o ayudados por el panel de diseño, o directamente escribiendo el código.

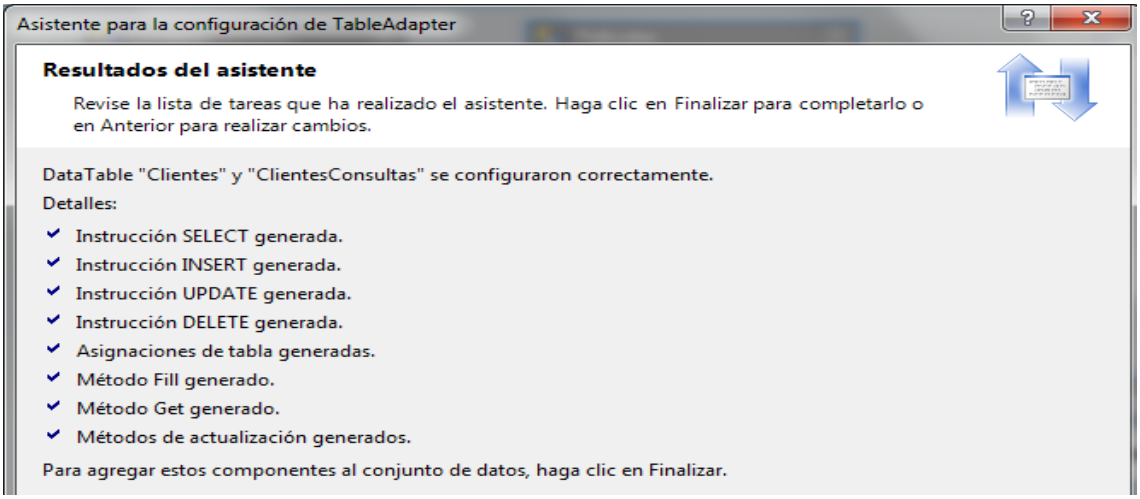
Una vez que ya tenemos nuestra sentencia SQL, y damos a Siguiente… nos encontramos con lo siguiente:



Importante de esta ventana: poner el nombre a los métodos Fill y GetData. Ambos los terminaremos con el mismo nombre identificativo de qué hace esa consulta:

- Fill y GetData → son las consultas por defecto, como ya hemos dicho, que por lo general es una sentencia Select de todos los registros y todos sus campos, sin ningún filtro.
- FillBy**Campo** y GetDataBy**Campo** → Cuando realicemos consultas paramétricas, o filtradas o con la cláusula Where, el nombre deberá ser este, sustituyendo "campo", por el nombre del campo por el que realizamos el filtro.

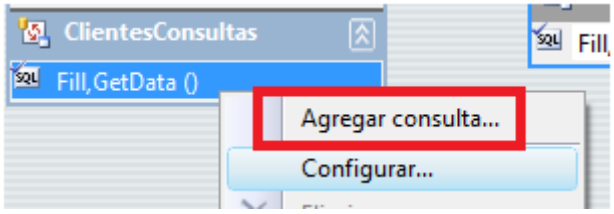
Y ya el último paso es el de comprobación de lo que hemos hecho:



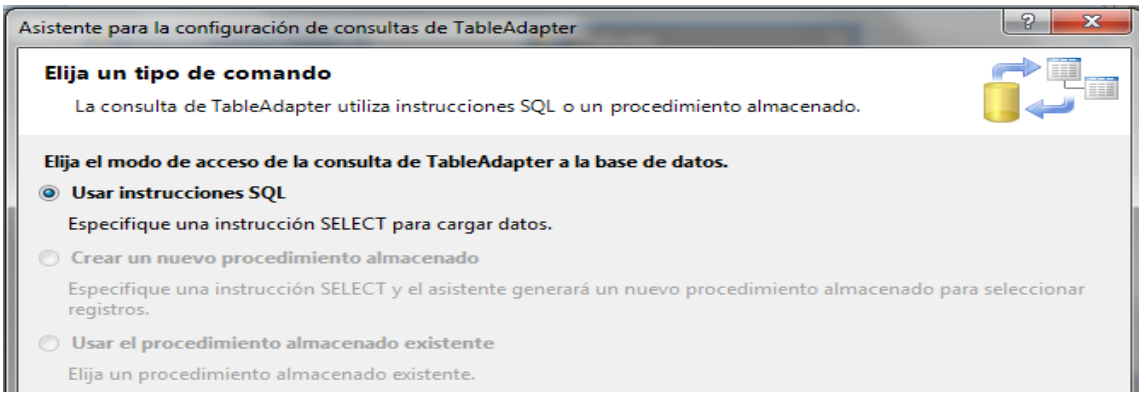
3.2. Agregar una nueva consulta

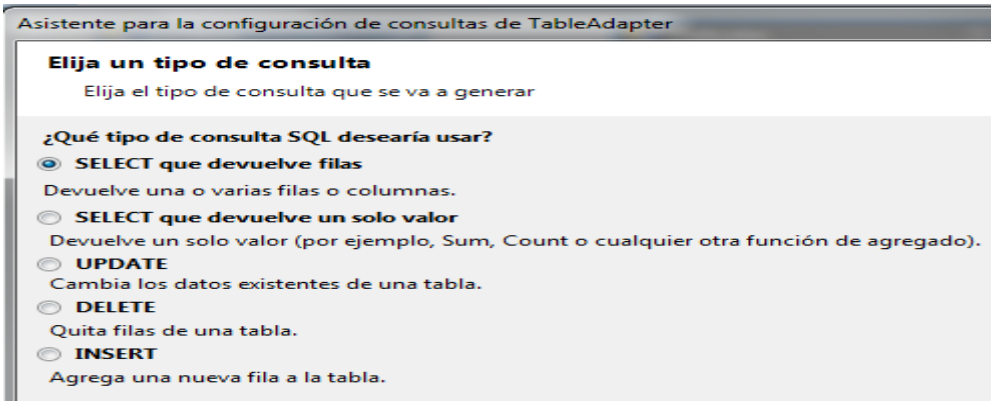
Obviamente con una consulta no hacemos nada…. ya que para nuestro código tendremos que recurrir a estas consultas para obtener los datos que precisamos y modificarlos….

Como ya hemos visto, si queremos Agregar una Consulta:



Nos lleva al mismo Asistente, salvo las dos primeras ventanas:





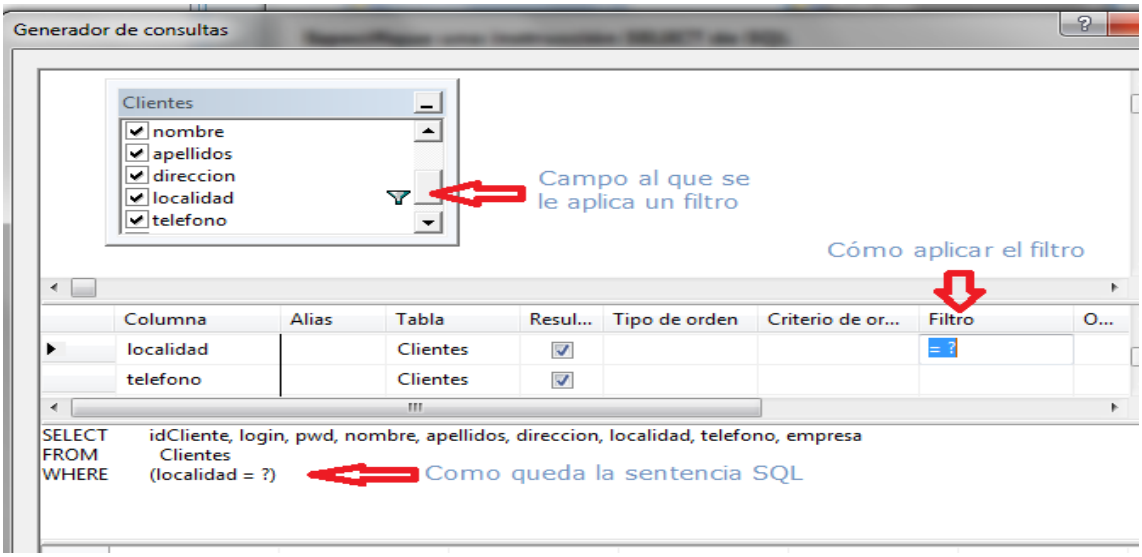
Para las consultas paramétricas… fácil!!!!

Son consultas paramétricas todas aquellas que requieren de que se les introduzcan datos o parámetros o argumentos, para obtener el resultado esperado

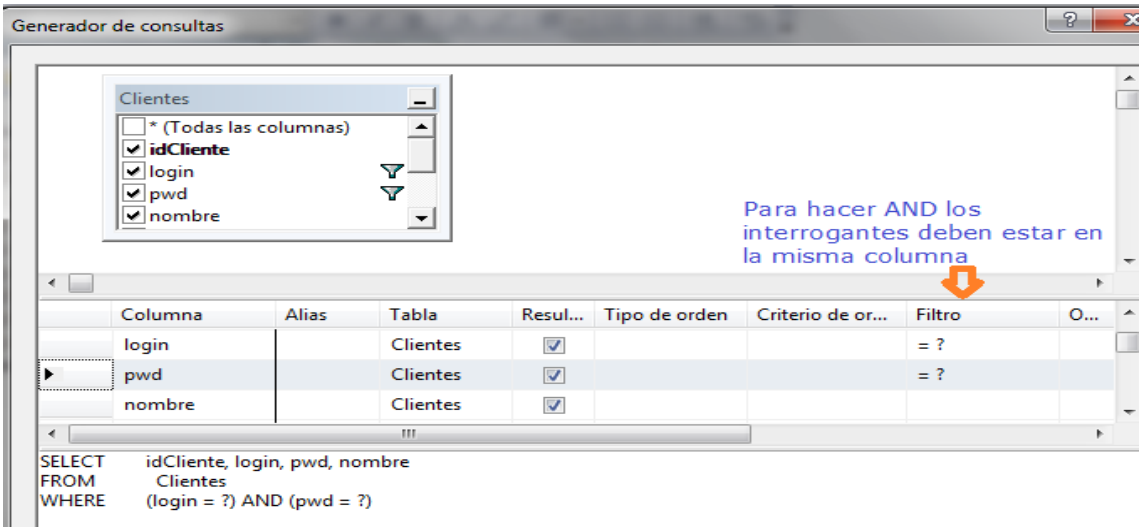
Y por tanto, utilizaremos la sentencia WHERE.

A continuación presento varios ejemplos:

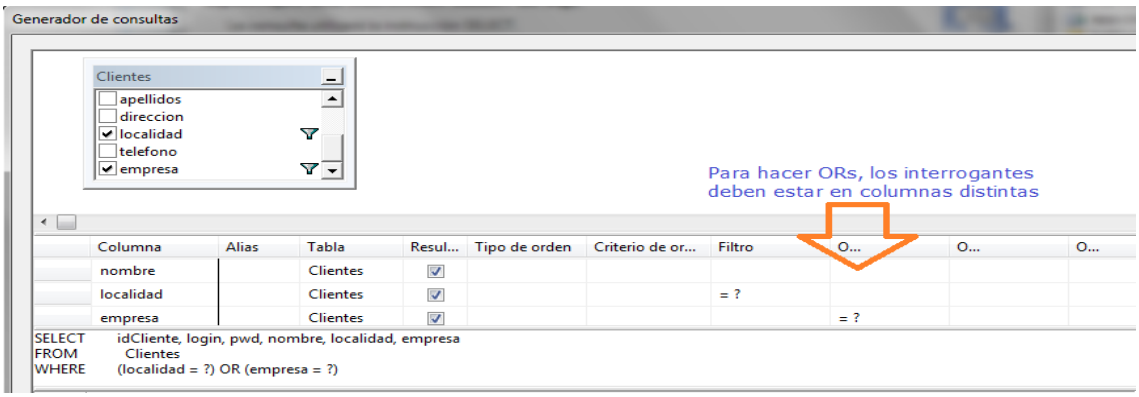
1º: Cuántos clientes son de una determinada localidad:



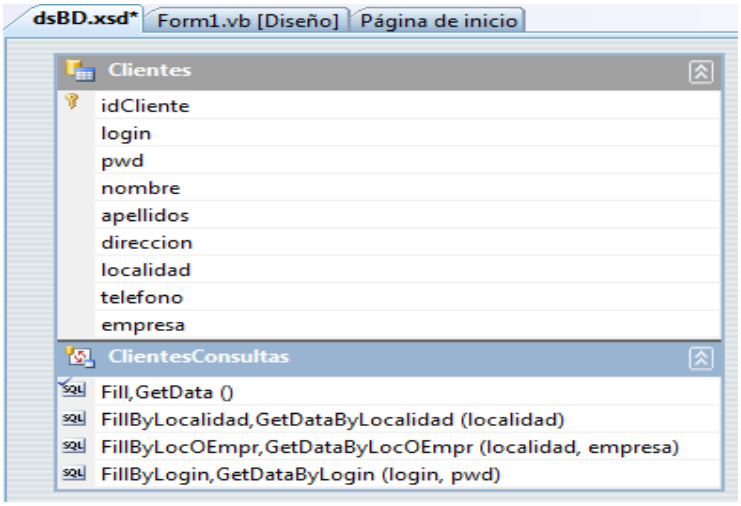
2º: Existe algún cliente cuyo login y contraseña coincida con la introducida por el usuario



3º: Ver qué clientes son de una determinada Empresa o de una determinada Localidad



Quedándonos el conjunto de consultas de la siguiente forma:



En nuestro Conjunto de Datos, estará la tabla en cuestión, y en su parte de Consultas, iremos viendo las que vamos creando, y los parámetros que requieren, si es que requieren.

4. Acceder a los datos consultados desde el código

Imagina que queremos que nuestros clientes se validen, introduciendo su login y su clave, para poder utilizar la aplicación.

En este caso, previamente deberíamos tener creada la consulta "**FillByLogin**" que hemos hecho en el punto anterior, para introduciéndole el login y la contraseña, nos devuelva si existe algún registro con estos datos.

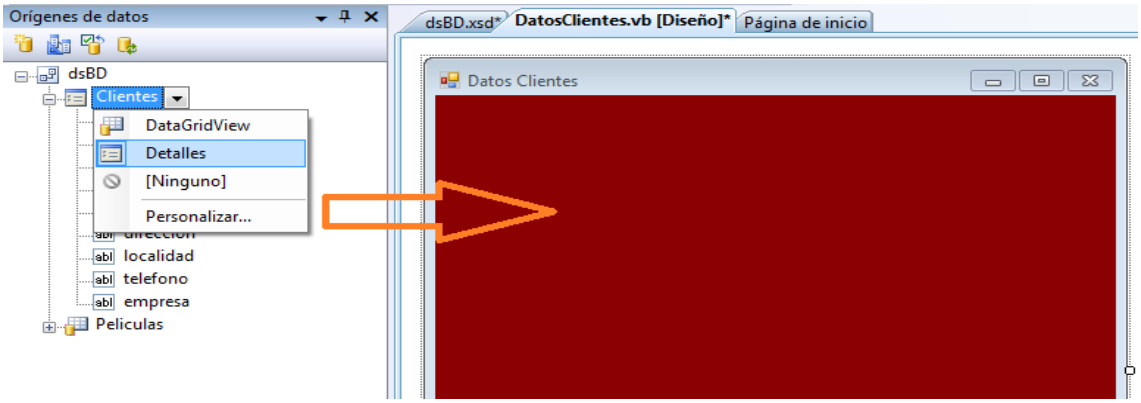
En código cómo se haría:

```
'
 public int IDSocio;

 private void btnOK_Click(object sender, EventArgs e)
 {
     //Creamos las variables necesarias para gestionar los datos:
     // Nuestro conjunto de datos
     dsBD ds = new dsBD();

     // Las consultasd de la Tabla Socios
     dsBDTableAdapters.SociosTableAdapter tbSocios = new dsBDTableAdapters.SociosTableAdapter();

     // Ejecutamos la consulta para ver si los datos introducidos son correctos
     tbSocios.FillByLoginPwd(ds.Socios,txtLogin.Text,txtPWD.Text);

     // Si tenemos algún socio con el Login y la Contraseña introducidas...
     if (ds.Socios.Count > 0)
     {
         MessageBox.Show("Buenas " + ds.Socios[0].Nombre, "Saludo", MessageBoxButtons.OK);
         IDSocio = ds.Socios[0].IdSocio;
     }
     else MessageBox.Show("Te has tenido que equivocar...");
 }
```

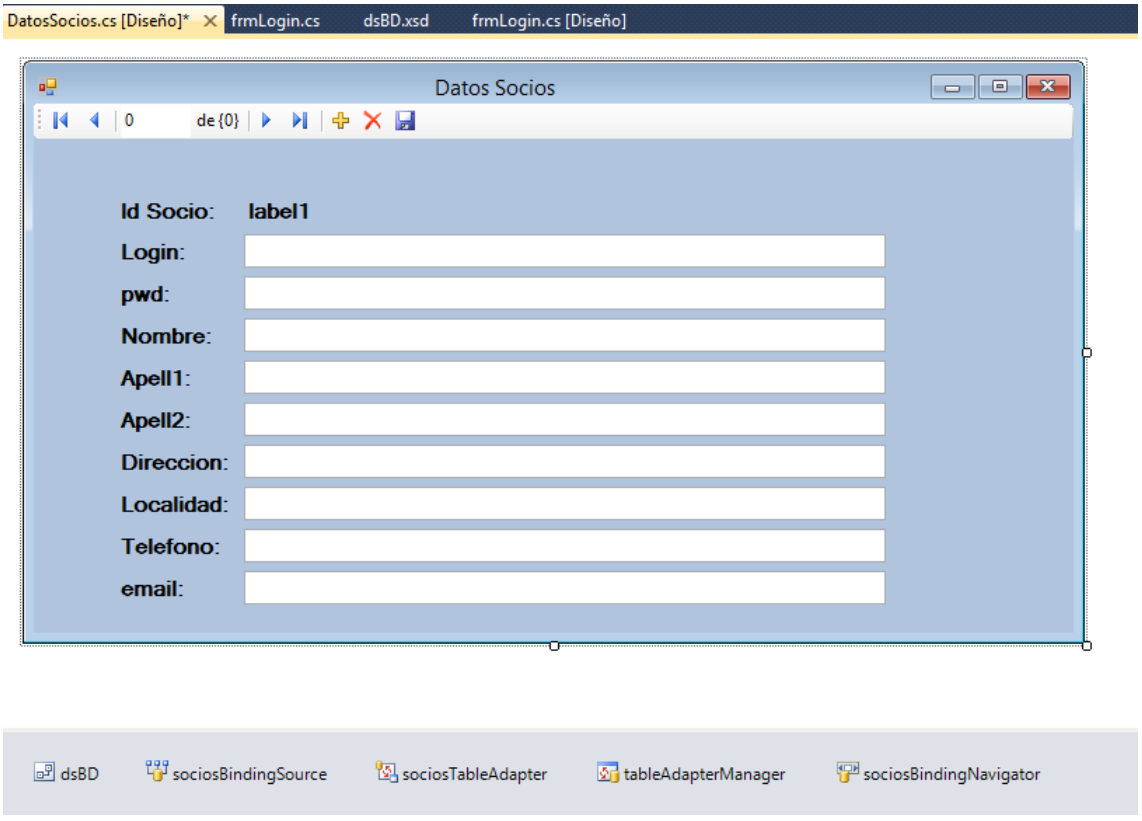
5. Generar un formulario en vista Detalle de una tabla

Vamos a generar un formulario donde podamos gestionar todos los registros de una tabla, de una forma rápida y sencilla.

Para eso, en nuestro Origen de Datos, elegimos la opción Detalles, y podemos modificar el control que queremos para cada campo, una vez hecho, solo queda pinchar la tabla y arrastrarla hasta nuestro formulario:



Quedando lo siguiente:



Como podemos apreciar, no solo se me han incorporado todos los controles vinculados a los campos de mi tabla, sino que en la parte no visible tenemos:

- **dsBD**: Mi conjunto de datos, ya no tendría que declararlo en código

- **sociosTableAdapter**: Las consultas de mi tabla.
- y los otros 3 controles que son los que me facilitan la barra Navegadora de los registros, y que además de que me permita moverme de uno a otro, me facilita las tareas de añadir, eliminar y modificar los registros.

Y el código escrito es:

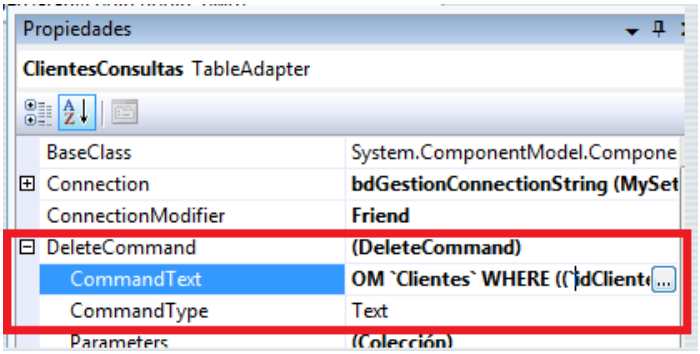
```
    public partial class DatosSocios : Form
    {
        public DatosSocios()
        {
            InitializeComponent();
        }

        private void sociosBindingNavigatorSaveItem_Click(object sender, EventArgs e)
        {
            this.Validate();
            this.sociosBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.dsBD);

        }

        private void DatosSocios_Load(object sender, EventArgs e)
        {
            // TODO: esta línea de código carga datos en la tabla 'dsBD.Socios' Puede
moverla o quitarla según sea necesario.
            this.sociosTableAdapter.Fill(this.dsBD.Socios);

        }
    }
```

Vamos que ya lo tendría todo hecho!!!!

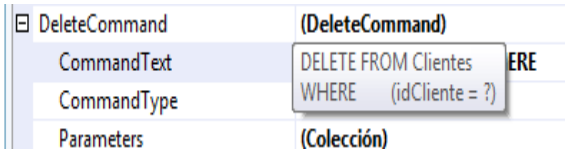
Vamos a modificar lo siguiente:

1º **No quiero que elimine un registro sin pedir la confirmación previamente.** Para ello:

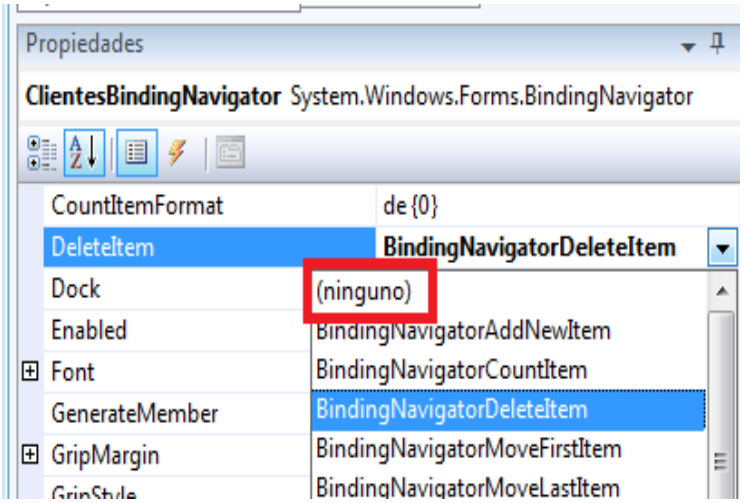
1. Modificaremos la sentencia Delete
   Nos vamos al Conjunto de Datos, seleccionamos **sociosTableAdapter**, y en sus propiedades editamos DeleteComand



2. Hacemos clic en el botoncito de los puntos suspensivos y en el editor de la consulta dejamos lo siguiente:



3. Una vez que ya tenemos la sentencia **Delete** como a nosotros nos gusta… pasamos a quitar el control de eliminar a la Barra Navegadora



En las propiedades del **SociosBindingNavigator**, buscamos su propiedad **DeleteItem**, y sustituimos lo que tiene por **(ninguno)**

Así cuando el usuario pulse en el botón eliminar, automáticamente no hará nada, salvo lo que nosotros le hayamos asignado por código.

4. Escribimos el código para el botón eliminar:

```
        private void bindingNavigatorDeleteItem_Click(object sender, EventArgs e)
        {
            int regs;
            DialogResult resp = new DialogResult();
            resp = MessageBox.Show("Seguro que quieres eliminar este registro???", "Eliminar Registro",
MessageBoxButtons.YesNo,MessageBoxIcon.Exclamation);

            if (resp== System.Windows.Forms.DialogResult.Yes)
            {
                regs= this.sociosTableAdapter.Delete(int.Parse(this.idSocioLabel1.Text));

                if (regs>0)
                    MessageBox.Show("Registro Eliminado","Ok",MessageBoxButtons.OK,MessageBoxIcon.Information);
                dsBD.Clear();
                sociosTableAdapter.Fill(dsBD.Socios);
            }

        }
```

De una forma más sencilla, pero también controlamos que al usuario se le indique cuando un registro ha sido “guardado” o “actualizado” que es lo que realmente hacemos:

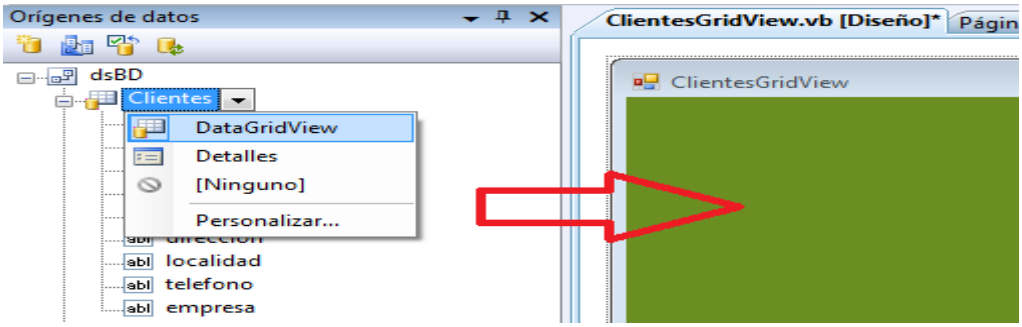
Para ello:

En el método: `sociosBindingNavigatorSaveItem_Click` le añadimos la sentencia MessageBox.Show con el mensaje que indica que ya se ha modificado correctamente un registro.

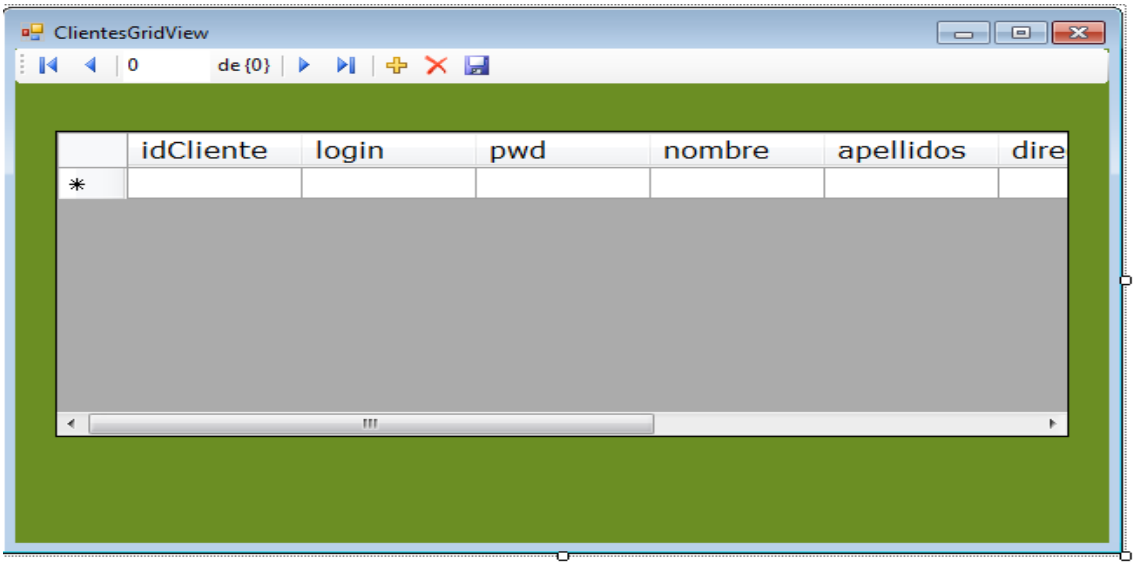
6. Generar un formulario con Vista DataGridView de una tabla

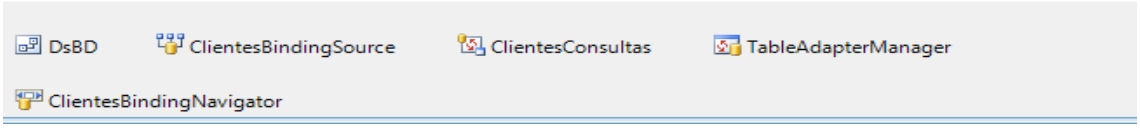
De una forma similar a lo hecho en el punto anterior, pero ahora eligiendo en vez de Detalle, DataGridView.

De igual forma podemos cambiar los controles asociados a los campos, y una vez que ya los tengamos, pinchamos sobre la tabla y la arrastramos al formulario

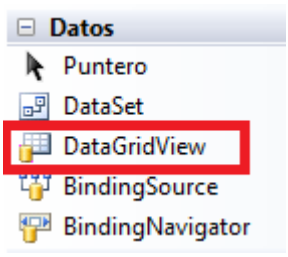


Quedándome lo siguiente:



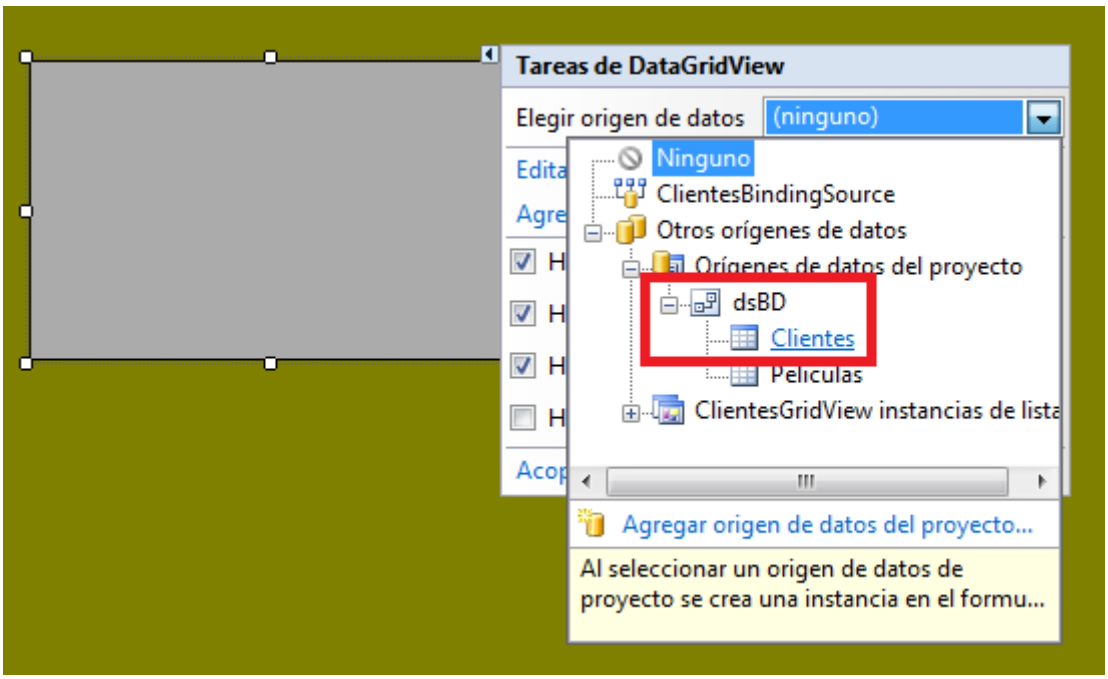


Solo que aquí no tendría sentido tanta historia… por lo que nos olvidamos de esta forma, y regresamos a nuestro formulario vacío…

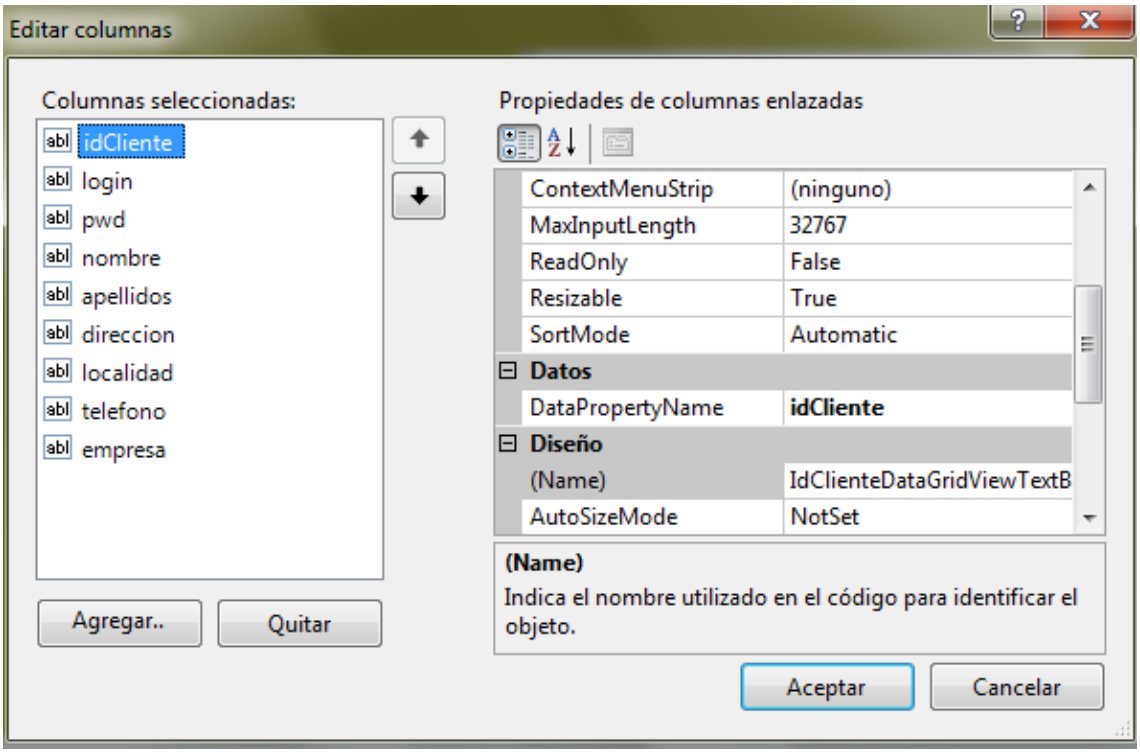


Añadimos de nuestra ventana de Herramientas, en su sección Datos, el control DataGridView

Al añadir el control, nos sale la siguiente ventana de Tareas del DataGridView, donde elegimos a qué tabla lo vinculamos:



Una vez vinculada, podemos editar las columnas, ordenando, quitando, modificando sus propiedades, etc.



Y de forma similar, conviene modificar las propiedades del DataGridView, para hacerlo más atractivo.

Por último nos quedaría añadir a nuestro formulario un botón de **Actualizar Cambios** con el siguiente código:

```
        private void btnActualizar_Click(object sender, EventArgs e)
        {
            int regs;
            regs = this.sociosTableAdapter.Update(this.dsBD.Socios);
            MessageBox.Show("Ha sido actualizados " + regs + " registros", "ok",
MessageBoxButtons.OK, MessageBoxIcon.Information);

            // Limpiamos el conjunto de datos para volverlo a llenar
            dsBD.Clear();
            sociosTableAdapter.Fill(dsBD.Socios);
        }
    }
```

7. Formulario de las Películas

Sería un formulario a simple vista sencillo, donde pudiéramos recorrer todos los registros de la tabla películas y poder insertar nuevas películas, modificar las que ya tenemos y eliminar las que queramos, siempre y cuando no estén alquiladas por ningún usuario.

Cosillas a tener en cuenta…

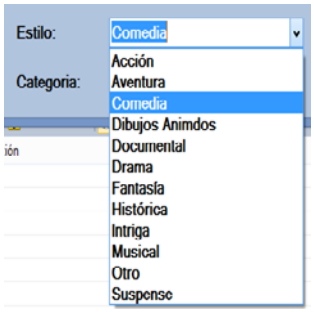
- Antes de arrastrar la tabla películas, desde el Origen de Datos al formulario, modifica aquellos campos que no sean de tipo texto (campo codPeli, Estilo, Stock, etc…)

- El campo foto tipo Objeto OLE, si lo tienes… lo puedes quitar… porque no nos va a servir… Para mostrar la carátula, utilizaremos el nombre del fichero contenido en el campo carátula, y un control PictureBox que mostrará dicho fichero.
- Y codifica las restricciones a la hora de eliminar una película. De Actualizar los registros y de añadirlas… informando al usuario en cada momento.

7.1. Cargar los ComboBox de Estilos y Categorías

Además de que me muestre el estilo de la película que estoy viendo… quiero que en caso de que quiera cambiar a otro estilo, yo pueda elegirlo de la lista de estilos de dicha tabla.

Para eso en el **Form_Load**, tendremos que llenar ese comboBox tanto para el campo de **Estilos** como el de **Categorías**.




```
//Cargar el combo de  Estilos

dsBDTableAdapters.EstilosTableAdapter tbEstilos = new dsBDTableAdapters.EstilosTableAdapter();
tbEstilos.Fill(dsBD.Estilos);
for (i=0; i<dsBD.Estilos.Count; i++)
    estiloComboBox.Items.Add(dsBD.Estilos[i].Estilo);
```

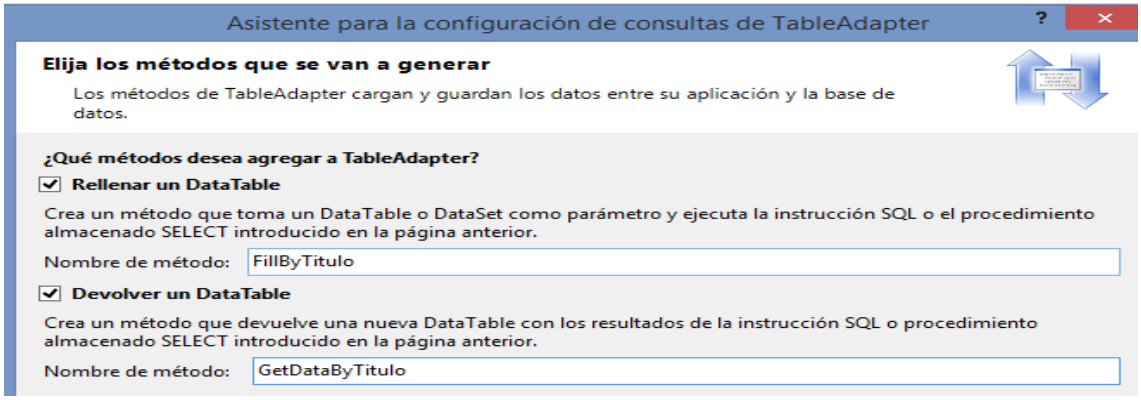
… de forma similar haríamos con el ComboBox de Categorías

7.2. Filtro o búsqueda de películas por título

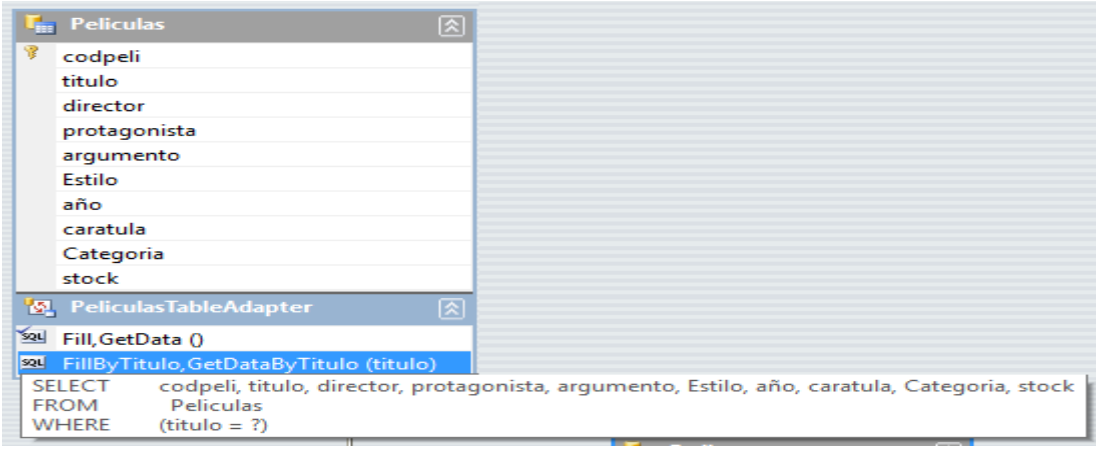
En la barra navegadora podemos además añadir nuevos botones y otros controles… en este caso, pondremos un **ComboBox** con los títulos de todas las películas, de forma que cuando seleccione un título, me muestre los datos de esa película. Este listado haza que salga ordenado alfabéticamente de forma ascendente.

Y también un botón de comando, "**Mostrar Todas**", para poder quitar el filtro de una película en concreto, y poder volver a mostrarlas todas…

Para ello debemos crear la consulta que me seleccione los datos de una película al introducir su Título. Te acuerdas dónde era???



En el Conjunto de Datos, Agregamos Nueva Consulta y la generamos…



Una vez que ya tengo la consulta solo tengo programar el procedimiento de cuando cambie el texto del comboBox de Titulos para que el conjunto de datos se seleccione solo con los datos de la película seleccionada:

```
private void TitulosComboBox_TextChanged(object sender, EventArgs e)
{
    dsBD.Peliculas.Clear();
    peliculasTableAdapter.FillByTitulo(dsBD.Peliculas, TitulosComboBox.Text);

}
```

De forma similar, cuando pulse el Botón “Mostrar Todas” debo cargar nuevamente el conjunto de datos con todas las películas.

7.3. Actualizar los Registros

Este evento se produce cuando hacemos clic sobre el disquete, al Guardar, tanto si estamos introduciendo un nuevo registro (primero pulsamos en el icono del +, introducimos los datos, y debemos finalizar la acción “guardando”), como si queremos guardar alguna modificación hecha.

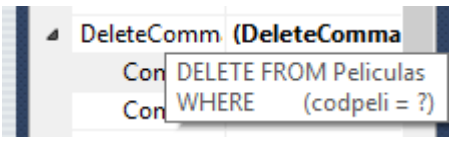
En este caso, como ya hicimos con los socios, mostraremos un mensaje al usuario indicándole que el registro se ha “Actualizado”

Y por si la actualización consistía en la modificación del Título de una película, o la inserción de una película nueva… actualizaremos también el comboBox de búsqueda por Títulos. Primero “limpiando” y luego “llenando” nuevamente.

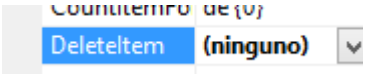
```
//Cargar el combo de Títulos de Películas
int i;
dsBD.Peliculas.Clear();
TitulosComboBox.Items.Clear();
peliculasTableAdapter.Fill(dsBD.Peliculas);
for (i = 0; i < dsBD.Peliculas.Count; i++)
    TitulosComboBox.Items.Add(dsBD.Peliculas[i].titulo);
```

7.4. Eliminar los Registros

Como ya hicimos con los socios… modificaremos este elemento para poder pedir al usuario que me confirme la acción antes de realizarla, y no podré eliminar una película que esté prestada a algún socio. Es requisito imprescindible para poder eliminar una película que no esté prestada a nadie. (Ha podido ser prestada, pero ya ha sido devuelta).

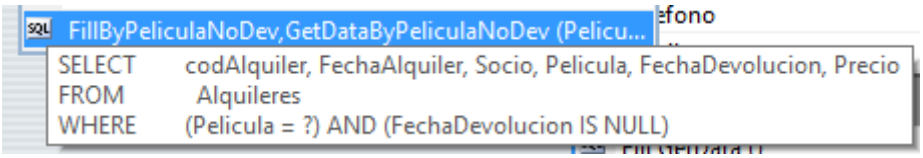


Lo primero… Simplificamos la consulta de Eliminar (Delete) a la mínima expresión.



Quitamos el control de este botón al PeliculasBindingNavigator

Generamos la consulta para ver si una película ha sido prestada y no devuelta.



Y ya lo tenemos todo para escribir el código…

```
        private void bindingNavigatorDeleteItem_Click(object sender, EventArgs e)
        {
            DialogResult resp = new DialogResult();
            resp = MessageBox.Show("Estás seguro de querer eliminar esta película??", "BORRAR",
MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

            if (resp==DialogResult.Yes)
            {
                //Miramos si esa peli está prestada
                dsBDTableAdapters.AlquileresTableAdapter tbAlqs = new dsBDTableAdapters.AlquileresTableAdapter();
                tbAlqs.FillByPeliculaNoDev(dsBD.Alquileres, int.Parse(codpeliLabel1.Text));
                if (dsBD.Alquileres.Count > 0)
                    MessageBox.Show("No puede eliminarse porque está alquilada!!!", "Error!!",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                else    // Eliminamos
                {
                    peliculasTableAdapter.Delete(int.Parse(codpeliLabel1.Text));
                    MessageBox.Show("Película eliminada correctamente!", "BORRADA", MessageBoxButtons.OK,
MessageBoxIcon.Information);

                    // Actualizamos la lista de Titulos
```

7.5. Mostramos las Carátulas

Quizás sea lo más complicado porque esto sí que se distingue de hacerlo en uno u otro lenguaje de programación… cuando realmente la idea es la misma siempre.

Queremos hacer lo siguiente…

En el campo Carátula solo tendremos el **nombre del fichero.extensión** y sabemos que todas las carátulas están en la carpeta **Carátulas** que se encuentra en mi carpeta principal, o local al fichero ejecutable. Es decir, la he colocado en el directorio **\bin** para así poder acceder a ella de modo local.

Con lo cual el procedimiento para mostrar las carátulas quedaría de la siguiente forma:

```
private void CargarCaratula()
{
    //MOSTRAMOS LA IMAGEN DE LA CARÁTULA!!

    string fichCarat = "caratulas\\" + caratulaTextBox.Text;
    string rutaCaratula = System.IO.Path.Combine(Application.StartupPath, fichCarat);

    this.picCaratula.ImageLocation = rutaCaratula;
}
```

De forma que con **Application.StartupPath** nos situamos en el directorio actual lo que en Visual Basic era con CurDir();

Y al directorio actual (\bin) le añadimos la ruta local de la imagen que queremos mostrar.

Al estar en C# tenemos que poner un doble \ como elemento de escape (salvamos la barra)

También queremos tener todas las carátulas en esta carpeta… con lo que si el usuario cargase la imagen de una carátula de otra carpeta… no pasaría nada, solo que de forma transparente yo copiaría el fichero elegido a mi carpeta de carátulas… y para eso en el procedimiento del botón de comando "Nueva Carátula":

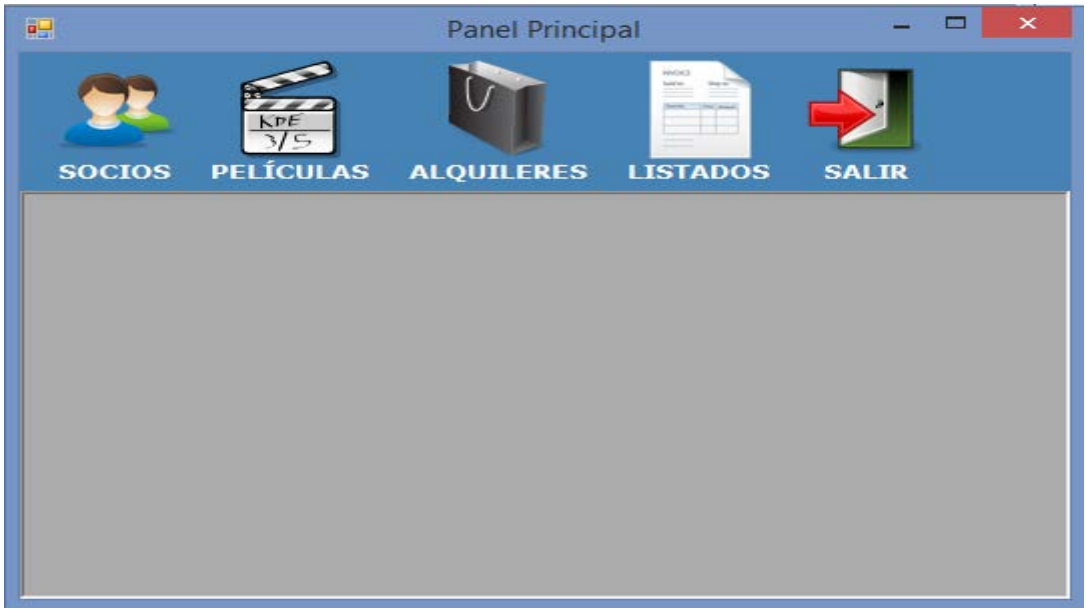
…. Además de abrir el OpenFileDialog y cargar la imagen que haya elegido el usuario, tanto en foto como en el campo CaratulaTextBox, también realizaríamos la copia del fichero con la instrucción: **System.IO.File.Move(rutaFichOrigen, rutaFichFinal)** (para mover el fichero, o **Copy** si lo que queremos es copiarlo)

```
if (!carpetaFinal.Equals(carpetaOrigen))    // Hay que copiar el nuevo fichero a la carpeta Carátulas
{
    string sourceFile = OpenFiles.FileName;
    string destinationFile = carpetaOrigen + caratulaTextBox.Text ;

    // To move a file or folder to a new location:
    System.IO.File.Move(sourceFile, destinationFile);
}
```

8. Panel Principal

Vamos a ir poniendo bonita la aplicación… Tendremos el siguiente panel principal, al cual accederemos si somos socios… se presupone que todos somos super-administradores, pero que si no lo quisiéramos así, ya sabríamos cómo hacerlo para que solo pudiera acceder a este panel si es el administrador por ejemplo.

Nuestro panel principal va a ser el siguiente:



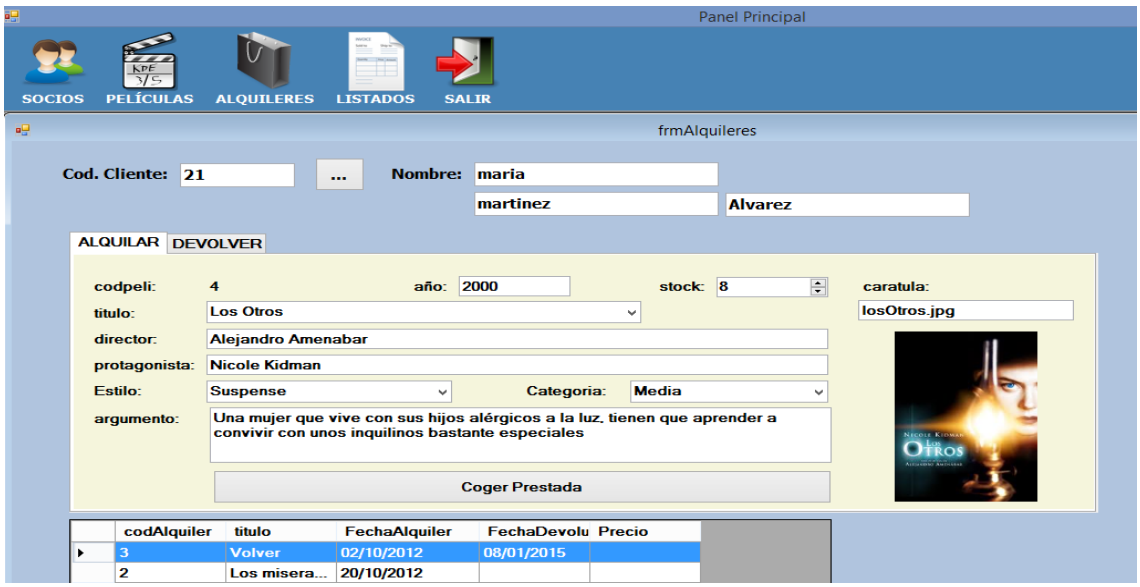
Un botón para cada formulario principal… y todos ellos se abrirán **CONTENIDOS** en el formulario padre principal.

Y no podrá haber más de un formulario hijo abierto a la vez. Si se pulsa en un botón para abrir un formulario hijo, cuando ya hay otro abierto, se cerrará el primero y se abrirá el que el usuario ha seleccionado.

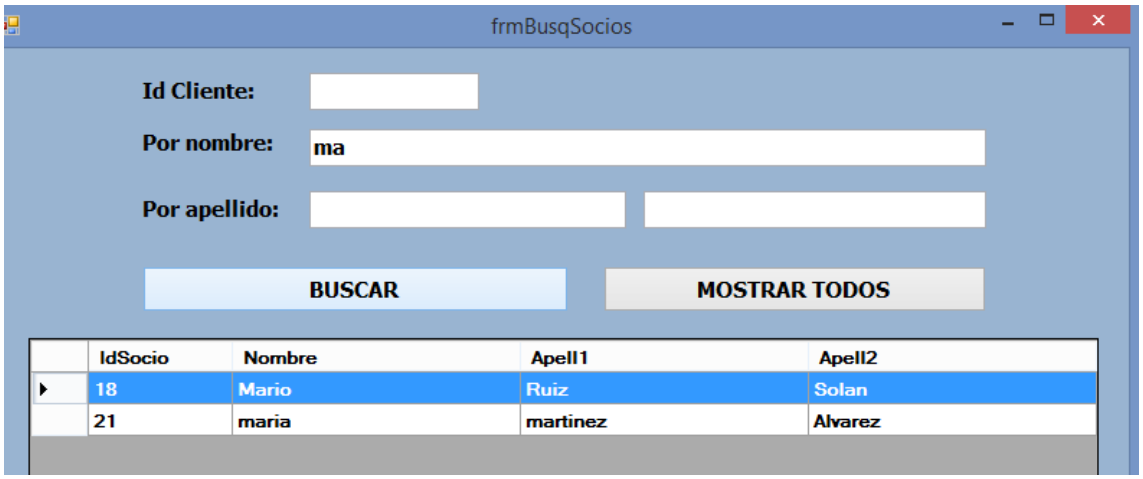
Ya tenemos hechos los formularios: Datos Socios y Películas… a continuación haremos lo que nos falta.

9. Formulario de Alquileres

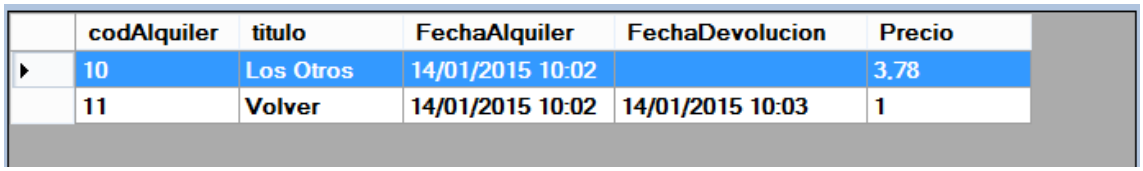
Este formulario sería el más completo porque es el que realiza la acción de prestar y devolver películas a los distintos socios.



Como podéis observar, la idea es que indiquemos el socio por su nº de código, o bien porque nos lo sabemos, o bien porque pincho en el botón de puntos suspensivos… y me lleva a la siguiente ventana desde la cual yo veo todos los socios, y haciendo doble clic en la el DataGridView sobre el que busco me regresa a esta ventana ya con los datos del socio. También podemos acotar el listado de socios introduciendo una aproximación del nombre y/o los apellidos, o indicando el código. Esta ventana se cerrará cuando haga doble clic sobre el socio que busco.



Al tener al socio, en la ventana de Alquileres, ya me muestra en la tabla inferior qué películas tiene alquiladas y si han sido devueltas o no…



9.1. Alquilar y Devolver Películas

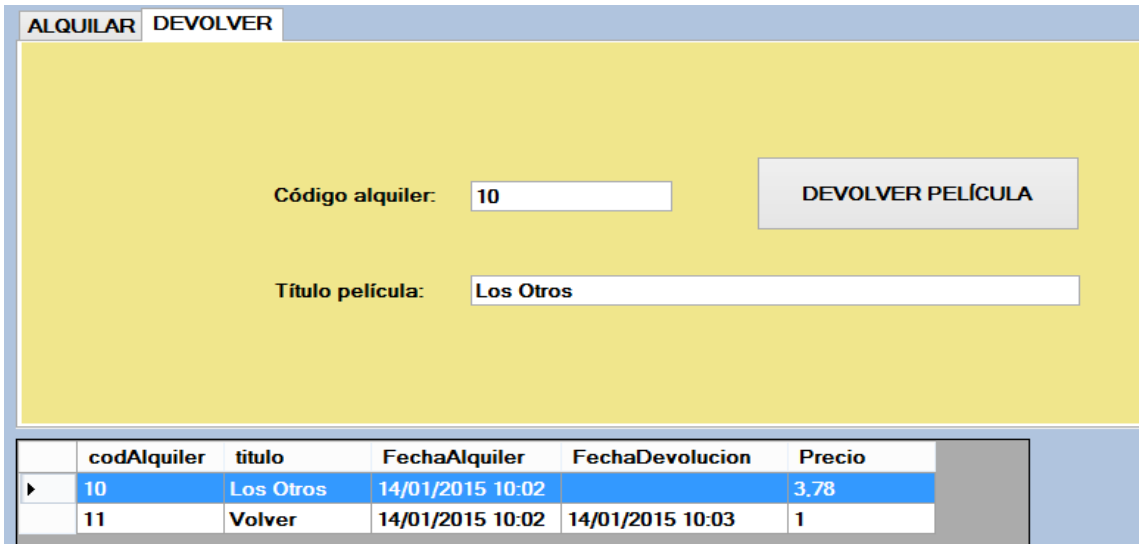
En el TabControl tenemos lo necesario para alquilar y para devolver las películas.

Tendremos un ComboBox para seleccionar por el Título la película que desearíamos Coger Prestada.

Cuando la carguemos, si el valor del campo Stock es superior a 0, entonces el botón Coger Prestada estará activo, si no estará deshabilitado.

Si la Cogemos Prestada actualizaremos su Stock restándole uno, y la tabla alquileres añadiendo esta película al socio.

Y para Devolver…



Simplemente haciendo dobleClick sobre la película prestada que queremos devolver, se introducirán los datos (Código Alquiler y Título Película) en los cuadros de texto, para verificar que se ha seleccionado bien… y al pulsar en Devolver Película… se actualizará la tabla de Alquileres introduciendo en el campo Fecha de Devolución la fecha del día y se actualizará el stock de la película incrementándose en uno.