



Cuadernillo de entregas teóricas de la materia:

PROGRAMACIÓN ORIENTADA A OBJETOS

Lenguaje: Visual C#

Profesor: Ing. Leandro A. Pini

Ciclo Lectivo: 2025

Carrera: Tecnicatura Superior en Análisis, Desarrollo y Programación de Aplicaciones

Institución: ISFT N° 93 de San Vicente

Tabla de contenidos / ÍNDICE

<u>1.</u>	<u>METODOLOGÍA DE TRABAJO Y APROBACIÓN DE LA MATERIA</u>	4
<u>2.</u>	<u>Repaso completo de conocimientos previos en C#</u>	7
<u>3.</u>	<u>ACTIVIDADES PARA REPASO (Obligatorias)</u>	21
<u>4.</u>	<u>INTRODUCCIÓN A “PROGRAMACIÓN ORIENTADA A OBJETOS”</u>	22
<u>5.</u>	<u>EJEMPLO PASO A PASO DEL PRIMER PROGRAMA EN VISUAL C#</u>	26
<u>6.</u>	<u>Cosas útiles en Visual C#</u>	31
<u>7.</u>	<u>ANEXO 1 - Usando un listBox</u>	33
<u>8.</u>	<u>Clases y Objetos en Visual C#</u>	34
<u>9.</u>	<u>CLASES ABSTRACTAS</u>	35
<u>10.</u>	<u>CREANDO Y UTILIZANDO CLASES GLOBALES</u>	36
<u>11.</u>	<u>ANEXO 2 - CARGA DE UN ComboBox CON C# PASO a PASO</u>	50
<u>12.</u>	<u>ANEXO 3 - CARGA DE UNA GRILLA (DataGridView) CON C# PASO a PASO</u>	52
<u>13.</u>	<u>ANEXO 4 - TIPS USANDO DATAGRIDVIEW Y COMBOBOX</u>	54
<u>14.</u>	<u>PANEL, GET-SET Y OTROS</u>	57
<u>15.</u>	<u>EVENTOS VALIDATING Y VALIDATED</u>	67
<u>16.</u>	<u>STRUCT Y RECORDS</u>	69
<u>17.</u>	<u>ACCESO A BASE DE DATOS USANDO OLEDB</u>	75
<u>18.</u>	<u>Ejemplo simple de conexión a un archivo de DB Access</u>	76
<u>19.</u>	<u>CONECTANDO A UNA DB SQL</u>	80
<u>20.</u>	<u>ANEXO 4 - Carga de un combobox usando DataSource</u>	87
<u>21.</u>	<u>ANEXO 5 - Carga de una grilla (DataGridView) con una consulta a una DB</u>	88
<u>22.</u>	<u>ANEXO 6 - Jugando con el DataGridView</u>	89
<u>23.</u>	<u>APLICANDO FILTROS SOBRE UN DATAGRIDVIEW</u>	92
<u>24.</u>	<u>CARGA TXT A DB SQL Y APLICANDO FILTROS</u>	96
<u>25.</u>	<u>CONECTANDO A UNA DB DE PostgreSQL</u>	103
<u>26.</u>	<u>VALIDACIÓN DE DATOS INGRESADOS</u>	116
<u>27.</u>	<u>UTILIZANDO HILOS (Threads) CON OBJETOS</u>	120
<u>28.</u>	<u>MANEJO DE EXCEPCIONES EN C#</u>	123
<u>29.</u>	<u>PROGRAMACIÓN EN 3 CAPAS CON VISUAL C#</u>	124
<u>30.</u>	<u>ANEXO 7 - Tutoriales de ayuda para 3 capas</u>	125
<u>31.</u>	<u>ANEXO 8 - QUE ES EL e.Handled</u>	126
<u>32.</u>	<u>ANEXO 9 - EJERCICIOS SIMPLES DE PROGRAMACIÓN EN “Visual C#”</u>	127

<u>33.</u>	<u>ANEXO 10 - Programa en C++ que genera Zudokus</u>	131
<u>34.</u>	<u>Ejemplos Básicos de programación en windows form</u>	133
<u>35.</u>	<u>Ejemplo completo ABMC de una Base de Datos Acces</u>	144
<u>36.</u>	<u>Cambio de Formulario inicial</u>	155
<u>37.</u>	<u>Generación de reportes de DB con Visual C#</u>	159
<u>38.</u>	<u>Especial Funciones</u>	171
<u>39.</u>	<u>Manejo de objetos de un formulario a otro</u>	174
<u>40.</u>	<u>Pasaje de valores (datos en variables) de un formulario a otro</u>	176
<u>41.</u>	<u>EXTRA 1 - Apuntes de SQL</u>	179
<u>42.</u>	<u>EXTRA 2 - PRACTICA PRIMER PARCIAL</u>	195
<u>43.</u>	<u>EXTRA 3 – PORTAFOLIO EN GIT-HUB</u>	196
<u>44.</u>	<u>EXTRA 4 - PROYECTOS CON EJEMPLOS ÚTILES</u>	197

1. METODOLOGÍA DE TRABAJO Y APROBACIÓN DE LA MATERIA

- Por parte del profesor y del ayudante: Exposición oral en el aula y/o laboratorio, de forma escrita por medio de “entregas” y prácticas en laboratorio de los diversos temas de la materia.
- Durante los primeros días de cursada se realizará un repaso de conocimientos adquiridos en Programación I. Luego se comenzará con la migración al lenguaje Visual C#.
- La asistencia será tomada de la siguiente manera: además de cumplir con el porcentaje mínimo de asistencia, el alumno deberá presenciar la clase completa, o sea, se tomará asistencia al comienzo de la clase y antes de finalizar la misma, si un alumno no está presente en alguno de los dos momentos entonces se le pasará como ausente. Esto se debe a una cuestión de respeto hacia el profesor y hacia sus compañeros.

Aclaración: Si el alumno por algún motivo llegara más tarde a las clases o deberá retirarse antes de que la misma finalice se requerirá un certificado ya sea laboral y/o médico.

FECHAS de Parciales y TPs:

- Entrega a estudiantes de la lista de programas a desarrollar, divididos en 2 Trabajos Prácticos (uno individual -TP1- y uno grupal -TP2-): primera semana de abril.
- Entrega del TP1: fecha límite viernes 11 de julio.
- Defensa del TP1: semana del 14 al 18 de Julio.
- Primer Parcial: semana del 25 al 29 de agosto.
- Entrega del TP2: fecha límite viernes 24 de octubre.
- Defensa del TP2: semana del 27 al 31 de octubre.
- Segundo Parcial: semana del 03 al 07 de noviembre.
- Recuperatorio de uno de los dos parciales (el que tengan desaprobado claro está): semana del 10 al 14 de noviembre.
- Integrador (última oportunidad de aprobación de cursada): semana del 17 al 21 de noviembre.

Aclaración: si el alumno no cumple con algunos puntos anteriores mencionados no aprobará la materia ya que, de ese modo, no cumple con los requisitos y pautas para la aprobación de la misma.

Los trabajos serán evaluados de la siguiente manera:

- Diseño y funcionalidad del trabajo/programa/aplicación/sistema.
- Calidad tanto a nivel programacional como a nivel de redacción para el caso del manual de usuario de dicho sistema.
- Nivel y calidad de la Defensa individual.
- **No se aceptará el trabajo si es entregado fuera del plazo establecido.**
- **Cada parte a evaluar tendrá su nota y la nota final del trabajo será el promedio de todas ellas.**
- Antes del receso vacacional se tomará el primer parcial de la materia.
- No dejar trabajos en las PC del instituto, siempre tener consigo un bkp de la última versión de los trabajos por ejemplo en un pen-drive (se recomienda una copia por cada integrante de equipo).
- **No se re-explicarán temas completos de clases pasadas.** Si alguno no entiende, pregunte el día que se da el tema, sino, utilice libros y/o Internet. Si no presenció la clase, pida que alguien le explique el tema.
- La clase anterior al parcial será para despejar dudas (esto no significa que se re-explicarán temas, sino, que solo se aclararán dudas de alguno de ellos).

EVALUACIÓN Y ACREDITACIÓN

Según el encuadre normativo resultante de la Resolución 4043 de Educación Superior y la Resolución 4196/24 RAM de Nivel Superior, se estipula el siguiente orden o esquema de evaluaciones:

Trabajos prácticos:

- Trabajos prácticos individuales y grupales (uno individual y uno grupal), de manera tal que los mismos sirvan no solo para incentivar el interés y la investigación sino también para que el estudiante se prepare de forma más segura y apropiada para las evaluaciones parciales y finales de la materia conociendo el grado de madurez de los conocimientos adquiridos.

Los trabajos prácticos serán tomados en cuenta como notas de afectación directa para la aprobación de la cursada de materia, aunque la idea principal de los mismos es estimular la investigación, el trabajo grupal y la utilización de los conceptos adquiridos para la resolución de problemas planteados en los mismos.

Parciales:

- Dos exámenes parciales escritos teórico-práctico, multiple-choice y desarrollo de código en hoja, con un recuperatorio (solo un recuperatorio a ser utilizado para recuperar el parcial que sea desaprobado por el estudiante). Cada parcial incluirá los temas vistos hasta el momento/fecha del mismo. Para la aprobación de éstos se les exigirá que la resolución correcta sea mayor o igual al 65% del mismo (nota requerida para la aprobación de cada parcial 4(cuatro)).

Integrador:

- Examen teórico-práctico, multiple-choice y desarrollo de código en hoja. A ser utilizado por quienes, en base a las notas obtenidas en las instancias de evaluación ya mencionadas (TPs y Parciales), no logren aprobar la cursada y necesiten una última oportunidad para lograrlo. Podrán acceder quienes hayan cumplido con la entrega y defensa de ambos TPs en tiempo y forma y tengan aprobado al menos un parcial o su recuperatorio. Incluye los temas vistos en todo el año o cursada. Para la aprobación del mismo se les exigirá que la resolución correcta sea mayor o igual al 70% (nota requerida para la aprobación del final 4(cuatro)).

Final:

- Examen teórico-práctico, multiple-choice. Para la aprobación del mismo se les exigirá que la resolución correcta sea mayor o igual al 65% (nota requerida para la aprobación del final 4(cuatro)).

Acreditación por Promoción Directa:

- Estudiantes que obtengan nota mayor o igual a 7 (siete) en todas y cada una de las instancias de evaluación (TPs y Parciales) y además cumplan con el porcentaje de asistencia requerido (mínimo de 60% de asistencia) tendrán Promoción Directa (sin instancia de evaluación final)

Criterios de Evaluación:

- Planteamiento de problemas: en el desarrollo de aplicaciones, los problemas son necesidades de los usuarios en un 90% de los casos, en el resto son fallas funcionales o cambios que se detectan dentro del ciclo de vida de estos.
- Producción de textos escritos: en los TP.
- Manejo de herramientas: uso de diversas herramientas de diagramación lógica, compiladores, etc. para el análisis y desarrollo de aplicaciones. Si solo se quedan con lo que el profesor dice o por su cuenta buscan opciones o posibles mejoras.
- Formulación de hipótesis en los TP.

La evaluación como concepto en si es tomada como un instrumento de medición tanto para el docente como para estudiante a modo de volcar de alguna forma a papel lo que se lleva aprendido y comprendido hasta el momento, a modo de estadísticas de avances y desarrollo de los estudiantes en cuanto al avance de la materia.

Devolución de los resultados: como costumbre, una vez corregidos los TP y finales son revisados de forma conjunta con los estudiantes en la siguiente clase a modo autocorrección y evacuación de sus dudas, porqué se equivocaron e incluso porqué algo es verdadero y/o porque es falso. Esta es una práctica que clarifica y da transparencia a los estudiantes del porqué de la nota obtenida y

además sirve para que no se queden con que sus respuestas estaban o todas bien o todas mal, es una forma de reafirmar los conceptos y trabajar con ellos para resolver los problemas planteados de forma correcta.

- Instrumentos:

- Para evaluar conocimientos: Pruebas escritas
- Para evaluar capacidades: Guías de práctica (con una serie de programas a desarrollar) y los TP.

- Indicadores:

Todo indicador debe tener 3 componentes:

ACCIÓN	CONTENIDO	CONDICIÓN
Definir la habilidad, destreza, actitud que se espera. Se expresa en tercera persona.	Se indica el contenido que debe aprender. Puede ser: conceptos, procedimientos o actitudes.	Aquí precisamos la cantidad o calidad que debe tener el resultado de una acción.
Ejemplo: Resuelve	Ejemplo: Problemas de geometría	Ejemplo: Utilizando teoremas y postulados.
Formula	Hipótesis sobre el origen de la vida	Utilizando sustento teórico.
Escribe	Un cuento	Respetando los signos de puntuación.

2. Repaso completo de conocimientos previos en C#

Este capítulo no pretende ser un estudio exhaustivo de los tipos de datos, estructuras de control y librerías de C#, sino una muestra de un conjunto de ejemplos de código escritos de dicho lenguaje.

Todo lo que aquí se muestra, se muestra de forma muy simple, procurando huir de elementos muy diferenciales. La forma de proceder utilizada en C# para estos ejemplos no es la más adecuada, sino la más simple.

Se parte de la base de que el alumno "sabe C#", y este capítulo es un repaso de los conocimientos que el alumno adquirió en su primer años de estudios. Por lo tanto, si el alumno no comprende los ejemplos propuestos, debería repasar "un poco" los contenidos de las asignaturas previas.

Para simplificar el código al máximo y quedarnos con lo esencial en lo que a sintaxis se refiere, todos los ejemplos están realizados en modo consola.

Ejemplo 1:

Hola mundo en C#
<pre>using System; namespace Simple { class CSimple { [STAThread] static void Main(string[] args) { Console.Write ("Hola mundo\n"); } } }</pre>

El ejemplo de la derecha muestra un típico programa 'Hola mundo' escrito en C#. Es necesaria la línea `using System` para acceder a la clase `Console` y al método `Write`. La función `Main` en este caso recibe directamente un array de `strings` (cadenas de caracteres en C#), como en el caso anterior. El número de argumentos está implícito en el array de C#.

En C#, además, es necesario un espacio de nombres (`namespace Simple`) y una clase (`class CSimple`), para poder contener la función `Main`, que será el punto de entrada del programa al ejecutarse.

Ejemplo 2:

Declaración de variables elementales e impresión de información por pantalla.
<pre>using System; namespace Simple { class CSimple { [STAThread] static void Main(string[] args) { int i = 4; double k = 3.74; string C = "Hola que tal"; Console.Write ("i = {0}, k = {1}, C = {2}\n", i, k, C); Console.WriteLine ("i = " + i + ", k = " + k + ", C = " + C); } } }</pre>

Se ha declarado un array de caracteres, con el tipo `string` de C#.

Se ha hecho la impresión, usando el método `Write` de `Console`, de dos formas diferentes: La primera usa `{0}`, `{1}`, etc. para indicar los argumentos a imprimir, mostrando su semejanza con los `%d` y `%f`, de C. La segunda usa el operador de concatenación de strings (+), para unir los fragmentos que se desean imprimir.

Ejemplo 3:

Entrada de datos por la consola (teclado). Conversión de tipos.
<pre>using System; namespace Simple { class CSimple { [STAThread] static void Main(string[] args) { string Cadena; int DatoEntero; double DatoReal; Console.Write ("Introduce tu nombre: "); Cadena = Console.ReadLine(); Console.Write ("Introduce tu edad: "); DatoEntero = Convert.ToInt32 (Console.ReadLine()); Console.Write ("Introduce el precio de una barra de pan: "); DatoReal = Convert.ToDouble (Console.ReadLine ()); Console.WriteLine ("Tu nombre: " + Cadena); Console.WriteLine ("Tu edad: " + DatoEntero.ToString()); Console.WriteLine ("Barra pan: " + DatoReal.ToString("0.00")); } } }</pre>

Vemos que toda la entrada por la consola se realiza a través de la función `ReadLine` de `Console`. `ReadLine` lee hasta que llega un <CR> (ENTER) del teclado, y retorna siempre un `string`.

Las operaciones de la clase `Convert` permiten convertir los `strings` leídos por el teclado al tipo de datos deseado.

Operadores:

Categorías	Operadores
Aritméticos	+ - * / %
Lógicos (booleanos y bit a bit)	& ^ ! ~ && false true
Concatenación de cadenas	+
Incremento y decremento	++ --
Desplazamiento	<< >>
Relacionales	== != < > <= >=
Asignación	= += -= *= /= %= &= = ^= <<= >>=
Acceso a miembros	.
Indización	[]
Conversión de tipos explícita	()
Condicional	?:
Concatenación y eliminación de delegados	+ -
Creación de objetos	new
Información de tipos	is sizeof typeof
Control de excepciones de desbordamiento	checked unchecked
Direcccionamiento indirecto y dirección	* -> [] &

Sentencias de control y palabras reservadas:

La lista de palabras reservadas de C# es la siguiente:

<u>abstract</u>	<u>event</u>	<u>new</u>	<u>struct</u>
<u>as</u>	<u>explicit</u>	<u>null</u>	<u>switch</u>
<u>base</u>	<u>extern</u>	<u>object</u>	<u>this</u>
<u>bool</u>	<u>false</u>	<u>operator</u>	<u>throw</u>
<u>break</u>	<u>finally</u>	<u>out</u>	<u>true</u>
<u>byte</u>	<u>fixed</u>	<u>override</u>	<u>try</u>
<u>case</u>	<u>float</u>	<u>params</u>	<u>typeof</u>
<u>catch</u>	<u>for</u>	<u>private</u>	<u>uint</u>
<u>char</u>	<u>foreach</u>	<u>protected</u>	<u>ulong</u>
<u>checked</u>	<u>goto</u>	<u>public</u>	<u>unchecked</u>
<u>class</u>	<u>if</u>	<u>readonly</u>	<u>unsafe</u>
<u>const</u>	<u>implicit</u>	<u>ref</u>	<u>ushort</u>
<u>continue</u>	<u>in</u>	<u>return</u>	<u>using</u>
<u>decimal</u>	<u>int</u>	<u>sbyte</u>	<u>virtual</u>
<u>default</u>	<u>interface</u>	<u>sealed</u>	<u>volatile</u>
<u>delegate</u>	<u>internal</u>	<u>short</u>	<u>void</u>
<u>do</u>	<u>is</u>	<u>sizeof</u>	
<u>double</u>	<u>lock</u>	<u>stackalloc</u>	
<u>else</u>	<u>long</u>	<u>static</u>	
<u>enum</u>	<u>namespace</u>	<u>string</u>	

Ejemplo 4:

Declaración y recorrido básico de arrays.

```
using System;
namespace Simple
{
    class CSimple
    {
        [STAThread]
        static void Main(string[] args)
        {
            int[] Numeros = new int[5];
            int i;
            for (i=0 ; i<5 ; i++) Numeros[i] = i;
            for (i=0 ; i<5 ; i++) Console.Write ("{0} ",Numeros[i]);
        }
    }
}
```

En el caso de C#, un array ha de “construirse”, con el operador `new`. Por ahora no vamos a ahondar en el tema de la “construcción”, basta con “creer” que es así, y utilizarlo, y dentro de poco tiempo lo entenderemos mejor.

```
using System;
namespace Simple
{
    class CSimple
    {
        [STAThread]
        static void Main(string[] args)
        {
            int[] Numeros = new int[] {1,2,3,4,5};
            for (int i=0 ; i<5 ; i++) Console.Write ("{0} ",Numeros[i]);
        }
    }
}
```

En el ejemplo anterior se muestra la declaración estática de un array. Vemos también la declaración de la variable de control dentro del `for`, algo muy habitual en el estilo de programación de C#.

Ejemplo 5:

Declaración y llamadas a funciones
<pre>using System; namespace Simple { class CSimple { static int UnaFuncion (int p, int q) { return (p+q); } [STAThread] static void Main(string[] args) { int Resultado = UnaFuncion (7,2); Console.WriteLine (Resultado); } } }</pre>

En C# se puede inicializar una variable con el resultado de una expresión ejecutable.

Ejemplo 6:

Paso de parámetros por referencia
<pre>using System; namespace Simple { class CSimple { static void UnaFuncion (ref int p) { p = p + 2; } [STAThread] static void Main(string[] args) { int Dato = 7; UnaFuncion (ref Dato); Console.WriteLine (Dato); } } }</pre>

En C# los punteros son considerados como “algo no seguro”, y hay otras maneras de efectuar esta operación. Vemos en el ejemplo que el parámetro se marca con el modificador ‘`ref`’, y que también se indica al invocar la función.

Para enviar parámetros por referencia, ya no son necesarios los punteros, y no han de utilizarse para ello.

En C# es preferible utilizar objetos (elementos de clases) en lugar de estructuras, y en este caso, el paso de parámetros es siempre por referencia, aunque no se indique. Esto lo veremos en un ejemplo posterior.

Ejemplo 7:

Ya se ha comentado anteriormente que no es necesario utilizar punteros en C#. Si aún así queremos seguir utilizándolos (no en esta asignatura), es necesario marcar las funciones que los usen como “no seguras”. En resumen, evitaremos usar punteros.

Uso de punteros
<pre>using System; namespace Simple { class CSimple { unsafe static void UnaFuncion (int* p) { *p = *p + 2; } [STAThread] unsafe static void Main(string[] args) { int Dato = 7; UnaFuncion (&Dato); } } }</pre>

```
        Console.WriteLine (Dato);  
    }  
}
```

Ejemplo 8:

Definición de estructuras

```
using System;
namespace Simple
{
    class CSimple
    {
        class Persona
        {
            public string Nombre;
            public int Edad;
        }
        [STAThread]
        static void Main(string[] args)
        {
            Persona P = new Persona();
            P.Edad = 39;
            P.Nombre = "Ruben Hidalgo";
            Console.WriteLine ("{0} - {1}", P.Edad, P.Nombre);
        }
    }
}
```

En C# son preferibles las clases a las estructuras. En el ejemplo anterior vemos la declaración de la clase `Persona`, semejante a la estructura equivalente de C. Los miembros de la clase deben ser públicos (`public`) para que la clase sea “equivalente” a la estructura.

Salvo en la declaración de la variable de tipo `Persona` (revisar el ejemplo anterior que mostraba la declaración de un array), el uso de una clase definida de este modo y una estructura es prácticamente el mismo.

Ejemplo 9:

Paso de estructuras/objetos por parámetro

```
using System;
namespace Simple
{
    class CSimple
    {
        class Persona
        {
            public int Edad;
            public string Nombre;
        }
        static void TratarPersona (Persona X)
        {
            X.Edad++;
        }
        [STAThread]
        static void Main(string[] args)
        {
            Persona P = new Persona();
            P.Edad = 39;
            P.Nombre = "Ruben Hidalgo";
            Console.WriteLine ("{0} - {1}", P.Edad, P.Nombre);
            TratarPersona (P);
            Console.WriteLine ("{0} - {1}", P.Edad, P.Nombre);
        }
    }
}
```

Tal como se comentaba en un punto anterior, el paso de objetos es siempre por referencia, y vemos en este ejemplo que la edad de la persona pasada por parámetro queda modificada.

Ejemplo 10:

Arrays dinâmicos

```

using System;
using System.Collections;
namespace Simple
{
    class CSimple
    {
        class Persona
        {
            public int Edad;
            public string Nombre;
        }
        static void ImprimirPersonas (ArrayList K)
        {
            Console.WriteLine ("Listado de personas:");
            for (int i=0 ; i<K.Count ; i++)
            {
                Persona P = (Persona) K[i];
                Console.WriteLine ("{0} - {1}", P.Edad, P.Nombre);
            }
            Console.WriteLine ();
        }
        [STAThread]
        static void Main(string[] args)
        {
            ArrayList K = new ArrayList ();
            Persona P = new Persona();
            P.Edad = 15;
            P.Nombre = "Manolo";
            K.Add (P);
            P = new Persona();
            P.Edad = 25;
            P.Nombre = "Filomeno";
            K.Add (P);
            P = new Persona();
            P.Edad = 48;
            P.Nombre = "Martirio";
            K.Add (P);
            ImprimirPersonas (K);
            K.RemoveAt (1);
            ImprimirPersonas (K);
        }
    }
}

```

Vemos que un objeto `ArrayList` tiene las operaciones necesarias para agregar y eliminar elementos, igual que el tratamiento implementado por `malloc`, `realloc` y `free`. Un objeto `ArrayList` “se inicializa solo” y “se destruye solo”. Pero esto es un tema sobre el que incidiremos profundamente cuando veamos la programación orientada al objeto, ahora no es relevante.

Ejemplo 11:

Acceso a ficheros de texto en modo lectura
<pre> using System; using System.IO; namespace Simple { class CSimple { [STAThread] static void Main(string[] args) { FileStream F = new FileStream ("C:\\\\Texto.txt", FileMode.Open, FileAccess.Read); while (F.Position < F.Length) { char c = (char) F.ReadByte (); Console.Write (c); } F.Close (); } } } </pre>

El ejemplo anterior muestra cómo leer carácter a carácter un fichero de texto. Un objeto `FileStream` sólo permite leer caracteres o bloques de ellos, no es capaz de ir leyendo de separador en separador.

```

static void Main(string[] args)
{
    StreamReader F = new StreamReader ("C:\\\\Texto.txt");
    for (;;)
    {
        string s = F.ReadLine ();
        if (s == null) break;
        Console.WriteLine ("Linea: " + s);
    }
    F.Close ();
}

```

El ejemplo anterior usa un objeto `StreamReader` para leer el fichero de texto, línea a línea.

```

using System;
using System.IO;
namespace Simple
{
    class CSimple
    {
        [STAThread]
        static void Main(string[] args)
        {
            StreamReader F = new StreamReader ("C:\\\\Texto.txt");
            for (;;)
            {
                string s = F.ReadLine ();
                if (s == null) break;
                Console.WriteLine ("Linea: " + s);
                string[] Palabras = s.Split (' ');
                for (int i=0 ; i<Palabras.Length ; i++)
                    Console.WriteLine ("      " + Palabras[i]);
            }
            F.Close ();
        }
    }
}

```

El ejemplo anterior va leyendo del fichero línea a línea, y utiliza el método `Split` del objeto `string` para separar las palabras en un nuevo array de cadenas de caracteres. La salida de este programa muestra línea a línea el fichero, y para cada línea muestra las palabras, separadas unas de otras.

Ejemplo 12:

Acceso a ficheros de texto en modo escritura
<pre> using System; using System.IO; namespace Simple { class CSimple { [STAThread] static void Main(string[] args) { StreamWriter F = new StreamWriter ("C:\\\\TextoSalida.txt"); F.Write ("Esto es una cosa a escribir\r\n"); F.WriteLine ("Esto es otra cosa, que tambien salta de linea"); int i = 4; double j = 45.67; string k = "Hola"; F.WriteLine ("Impresion de datos: {0} {1} {2}",i,j,k); F.Close (); } } } </pre>

`StreamWriter` es el tipo de datos (objeto) de C# que permite imprimir en un fichero de texto

MÉTODOS Y FUNCIONES - TIPOS DE FUNCIONES

El método Main es el punto de entrada principal en un programa en C#. Es el primer método que se ejecuta cuando el programa inicia, y su tarea principal es iniciar la ejecución del programa.

¿Cómo se ve el método Main?

En C#, el método Main se define como sigue:

```
static void Main(string[] args)
{
    // Código que se ejecuta al inicio
}
```

static: Significa que el método pertenece a la clase, no a una instancia de la clase.

void: Significa que el método no retorna ningún valor.

string[] args: Es un parámetro que recibe los **argumentos de la línea de comandos** (si los hay). Esto permite pasar información al programa desde la consola al momento de ejecutar el programa.

Funciones:

- Las funciones son algoritmos separados del cuerpo principal del programa que realizan tareas específicas.
- Las funciones hacen que el programa sea más modular.
- Las funciones ayudan a los programadores a ser ordenados en su código y durante todo el desarrollo.
- Las funciones solo deben ser depuradas una vez ya que una vez funcionales no es necesarios volver sobre ellas sino que simplemente se las usa.
- Las funciones ayudan al seguimiento y corrección de errores.

Las mismas pueden ser de los siguientes tipos:

- a- Función que ejecuta ciertas acciones pero que lo hace sin parámetros de entrada ni de salida (no recibe ni devuelve valores).

```
public static void maquillaje()
{
    Console.WriteLine("Buen día, espere por favor... el programa se está inciando...");
    Thread.Sleep(2500);
}
```

- b- Función que recibe parámetro de entrada, pero no retorna un resultado.

```
public static void SUMA(int valor1, int valor2)
{
    int resultado = valor1 + valor2;
    Console.WriteLine("El resultado es: " + resultado);
}
```

- c- Función que retorna un resultado sin recibir valores (parámetros) de entrada.

```
public static int MULTIPLICA()
{
    int resultado = 45 * 96;
    return resultado;
}
```

- d- Función que recibe valores de entrada y devuelve (retorna) valores resultantes.

```
public static int RESTA(int valor1, int valor2)
{
    int resultado = valor1 - valor2;
    return resultado;
}
```

Como se las utiliza dentro de un programa:

```

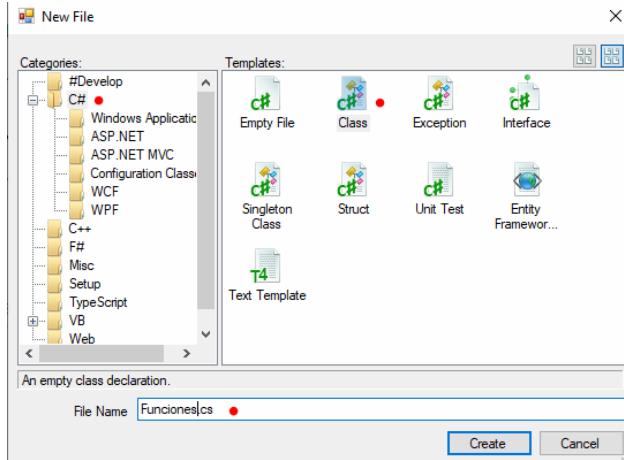
Variables.cs  Program.cs
Program
1  using System;
2  using System.Threading;
3  using Variables;
4
5  namespace EspecialFunciones
6  {
7      class Program
8      {
9          public static void Main(string[] args)
10         {
11             maquillaje(); // Llamada a función sin parámetros.
12
13             Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
14             Global.nombre = Console.ReadLine();
15             Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");
16
17             SUMA(25, 46); // Llamada a función con parámetros de entrada.
18
19             int result_multi = MULTIPLICA(); // Llamada a función con parámetros de salida
20             Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);
21
22             int result_resta = RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
23             Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);
24
25
26             Console.ReadKey();
27         }
28
29         public static void maquillaje()
30         {
31             Console.WriteLine("Buen día, espere por favor... el programa se está inciando...");
32             Thread.Sleep(2500);
33         }
34
35         public static void SUMA(int valor1, int valor2)
36         {
37             int resultado = valor1 + valor2;
38             Console.WriteLine("\n\nEl resultado de sumar es: " + resultado);
39         }
40
41         public static int MULTIPLICA()
42         {
43             int resultado = 45 * 96;
44             return resultado;
45         }
46
47         public static int RESTA(int valor1, int valor2)
48         {
49             int resultado = valor1 - valor2;
50             return resultado;
51         }
52     }
53 }
54

```

FUNCIONES EN CLASES GLOBALES

Ahora vamos a unir dos conceptos, Clases globales y Funciones.

Tomando el ejemplo de Clases Globales, crearemos una segunda clase que la llamaremos “Funciones” y la utilizaremos desde nuestro programa principal:



Código a colocar en esa clase:

```

using System;
using System.Threading;

namespace Funciones
{
    public static class Funcion
    {
        public static void maquillaje()
        {
            Console.WriteLine("Buen día, espere por favor... el programa se está inciendo...");
            Thread.Sleep(2500);
        }

        public static void SUMA(int valor1, int valor2)
        {
            int resultado = valor1 + valor2;
            Console.WriteLine("\n\nEl resultado de sumar es: " + resultado);
        }

        public static int MULTIPLICA()
        {
            int resultado = 45 * 96;
            return resultado;
        }

        public static int RESTA(int valor1, int valor2)
        {
            int resultado = valor1 - valor2;
            return resultado;
        }
    }
}

```

Como las usamos desde nuestro programa principal:

Con el mismo concepto y metodología que usamos con la clase de las Variables...

```

using System;
using System.Threading;
using Variables;
using Funciones; ←

namespace EspecialFunciones
{
    class Program
    {
        public static void Main(string[] args)
        {
            Funcion.maquillaje(); // Llamada a función sin parámetros.

            Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
            Global.nombre = Console.ReadLine();
            Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");

            Funcion.SUMA(25, 46); // Llamada a función con parámetros de entrada.

            int result_multi = Funcion.MULTIPLICA(); // Llamada a función con parámetros de salida
            Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);

            int result_resta = Funcion.RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
            Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);

            Console.ReadKey();
        }
    }
}

```

Algunas consideraciones que deben tener en cuenta al momento de usar funciones:

- 1- Verán que en algunos libros o incluso algunos profesores las llaman “métodos” en lugar de funciones. Yo prefiero separar entre “métodos” que hacen referencia a un objeto en particular con “funciones” que solo hacen referencia a una clase que puede no representar un objeto en si mismo.
- 2- Cuando usen funciones con parámetros de salida recuerden que deben respetar:
 - a. El tipo de dato de salida.
 - b. El tipo de dato con el cual es declarada la función (que debe ser el mismo tipo de dato que devuelve la misma).

Ejemplo:

Llamada a función que devuelve un valor de tipo entero, a una variable de tipo entero le asigno el resultado de invocar a dicha función:

```
int result_multi = Funcion.MULTIPLICA();
```

Esa función esta declara como “int” lo que indica que retorna un dato de tipo entero, y en el return de la función devolvemos una variable que ese de tipo entera:

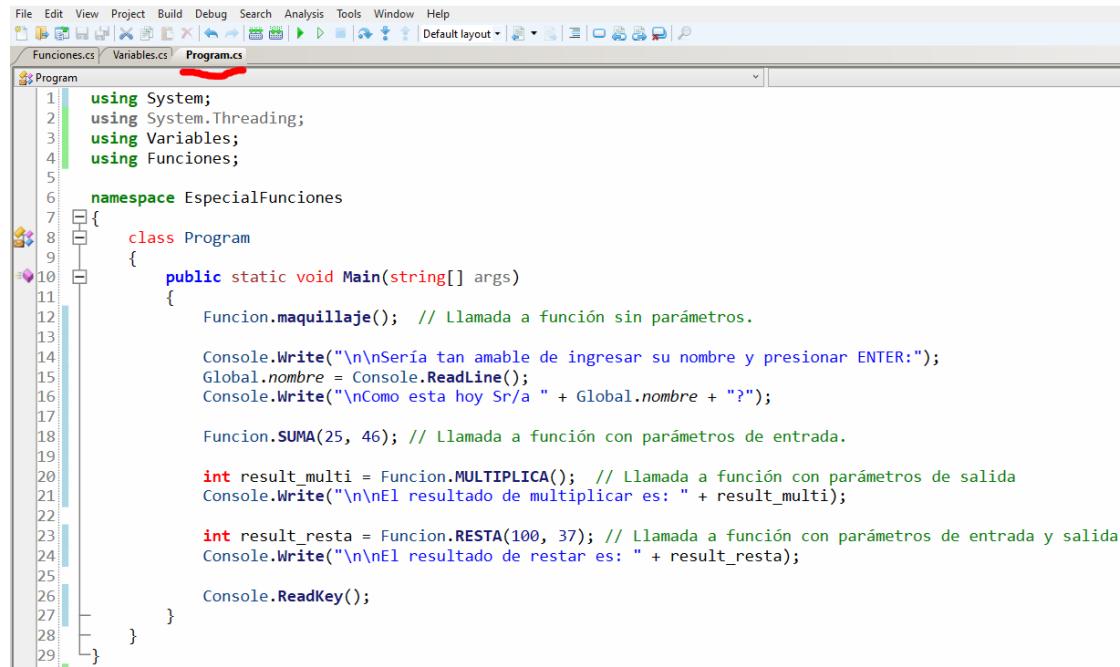
```

public static int MULTIPLICA()
{
    int resultado = 45 * 96;
    return resultado;
}

```

Diferentes vistas del programa terminado...

Vista el programa principal:

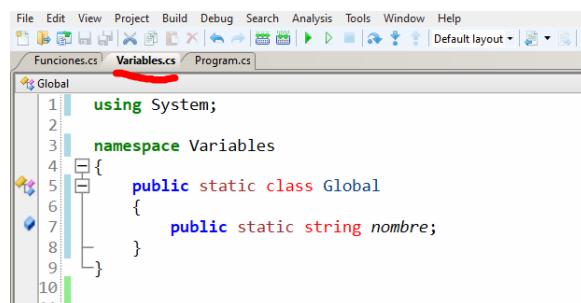


```

1  using System;
2  using System.Threading;
3  using Variables;
4  using Funciones;
5
6  namespace EspecialFunciones
7  {
8      class Program
9      {
10         public static void Main(string[] args)
11         {
12             Funcion.maquillaje(); // Llamada a función sin parámetros.
13
14             Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
15             Global.nombre = Console.ReadLine();
16             Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");
17
18             Funcion.SUMA(25, 46); // Llamada a función con parámetros de entrada.
19
20             int result_multi = Funcion.MULTIPLICA(); // Llamada a función con parámetros de salida
21             Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);
22
23             int result_resta = Funcion.RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
24             Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);
25
26             Console.ReadKey();
27         }
28     }
29 }

```

Vista de la Clase Variables:

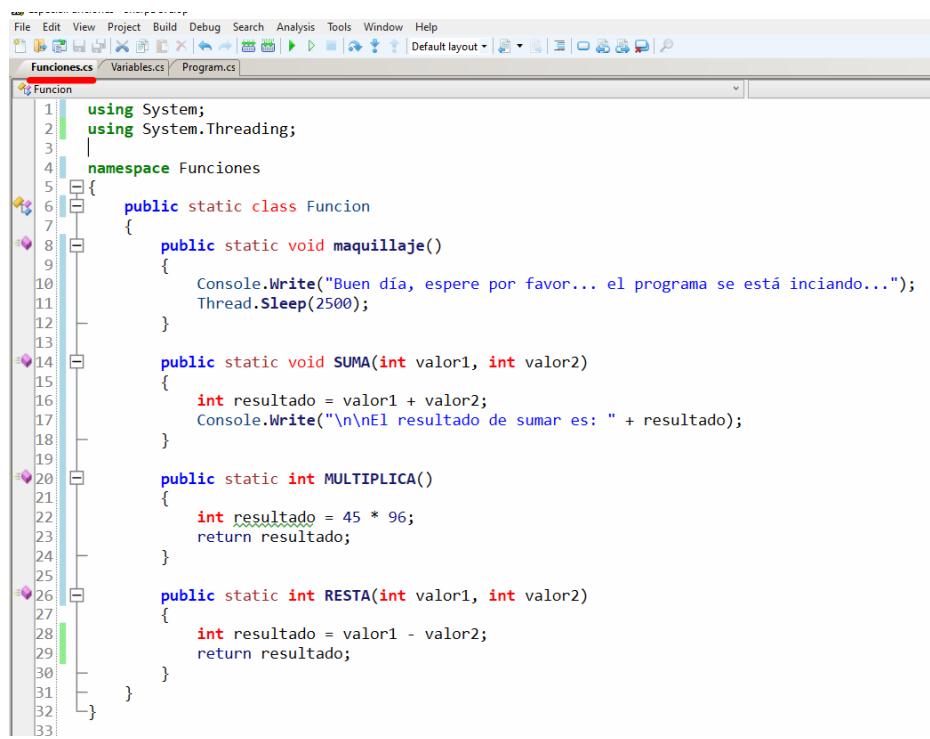


```

1  using System;
2
3  namespace Variables
4  {
5      public static class Global
6      {
7          public static string nombre;
8      }
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

```

Vista de la clase Funciones:



```

1  using System;
2  using System.Threading;
3
4  namespace Funciones
5  {
6      public static class Funcion
7      {
8          public static void maquillaje()
9          {
10             Console.WriteLine("Buen día, espere por favor... el programa se está inciando...");
11             Thread.Sleep(2500);
12         }
13
14          public static void SUMA(int valor1, int valor2)
15          {
16              int resultado = valor1 + valor2;
17              Console.WriteLine("\n\nEl resultado de sumar es: " + resultado);
18          }
19
20          public static int MULTIPLICA()
21          {
22              int resultado = 45 * 96;
23              return resultado;
24          }
25
26          public static int RESTA(int valor1, int valor2)
27          {
28              int resultado = valor1 - valor2;
29              return resultado;
30          }
31      }
32  }
33

```

FLUJOS DE DATOS

1. Flujos (Streams) en C#:

Un flujo (o stream) es una secuencia de datos que puede ser leída o escrita de forma continua. En C#, un flujo se usa para manejar la lectura y escritura de datos de una manera organizada y eficiente, y esto aplica tanto para archivos como para otros recursos (como la entrada o salida estándar).

En términos sencillos:

- Un flujo de entrada es cuando leemos datos desde algún lugar (como un archivo).
- Un flujo de salida es cuando escribimos datos a algún lugar (como un archivo o la consola).

En C#, la clase base para trabajar con flujos es Stream. Sin embargo, hay clases más específicas que permiten trabajar con flujos de manera más concreta, como FileStream, MemoryStream, etc.

2. Flujos de Caracteres (StreamReader y StreamWriter):

Ahora, si trabajamos específicamente con caracteres (y no solo bytes), C# proporciona clases más especializadas:

- StreamReader: Sirve para leer datos de un flujo (como un archivo) en formato de caracteres. Es útil cuando trabajas con texto y no con datos binarios.
- StreamWriter: Sirve para escribir datos de caracteres a un flujo, como cuando escribes en un archivo de texto.

Ejemplo básico de StreamReader y StreamWriter:

```
// Leer un archivo de texto
using (StreamReader sr = new StreamReader("archivo.txt"))
{
    string linea;
    while ((linea = sr.ReadLine()) != null)
    {
        Console.WriteLine(linea);
    }
}

// Escribir un archivo de texto
using (StreamWriter sw = new StreamWriter("archivo_salida.txt"))
{
    sw.WriteLine("Hola, mundo!");
}
```

3. Clases File, Directory y Path:

En C#, las clases File, Directory y Path proporcionan herramientas útiles para trabajar con archivos y carpetas en el sistema de archivos.

- Clase File: Se usa para trabajar con archivos de forma sencilla. Permite leer, escribir, copiar, eliminar archivos, etc.

Ejemplo:

```
// Comprobar si un archivo existe
if (File.Exists("archivo.txt"))
{
    Console.WriteLine("El archivo existe.");
}

// Copiar un archivo
File.Copy("archivo.txt", "copia_archivo.txt");

// Eliminar un archivo
File.Delete("archivo_a_borrar.txt");
```

4. Clase Directory

Se usa para trabajar con carpetas (directorios). Permite crear, eliminar o listar directorios.

Ejemplo:

```
// Crear un directorio
Directory.CreateDirectory("mi_carpeta");

// Listar los archivos en un directorio
string[] archivos = Directory.GetFiles("mi_carpeta");
foreach (var archivo in archivos)
{
    Console.WriteLine(archivo);
}
```

5. Clase Path

Proporciona métodos para trabajar con las rutas de los archivos y directorios (como obtener el nombre de un archivo de una ruta, combinar rutas, etc.).

Ejemplo:

```
// Obtener el nombre de un archivo desde una ruta
string archivo = Path.GetFileName("C:\\mis_archivos\\documento.txt");
Console.WriteLine(archivo); // Salida: documento.txt

// Combinar rutas
string rutaCompleta = Path.Combine("C:\\mis_archivos", "archivo.txt");
Console.WriteLine(rutaCompleta); // Salida: C:\\mis_archivos\\archivo.txt
```

Resumiendo los conceptos:

- 1- Flujos (Streams): Son secuencias de datos que pueden ser leídas o escritas.
- 2- StreamReader y StreamWriter: Trabajan con flujos de caracteres para leer o escribir texto en archivos.
- 3- File: Proporciona métodos para manejar archivos (crear, leer, escribir, eliminar).
- 4- Directory: Permite trabajar con directorios (crear, listar, eliminar).
- 5- Path: Facilita trabajar con rutas de archivos y directorios.

Actividad sugerida:

Desarrollar un programa que realice lo siguiente:

- 1- Crear un directorio.
- 2- Crear un archivo de texto dentro de ese directorio.
- 3- Escribir algunas líneas de texto en el archivo.
- 4- Leer el archivo y mostrar su contenido en la consola.

3. ACTIVIDADES PARA REPASO (Obligatorias)

Un cliente nos pide una entrevista para que le desarrollemos un programa simple, en este caso solamente de control de inicio de sesión.

Para tal fin nos comparte su idea a modo diagrama de flujo y nos dice:

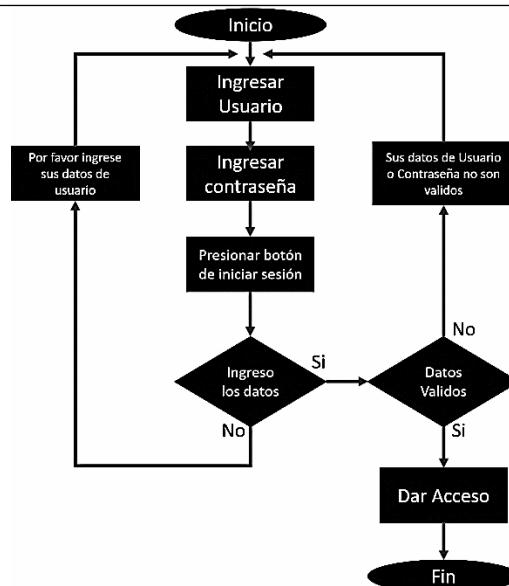
- 1- Deben respetar esta lógica.
- 2- Debe ser informativo e intuitivo (claridad en los mensajes que se emitan al usuario).

La tarea consiste en que:

- 1- Desarrollen/generen un diseño basado en una o varias pantallas, entrelazadas entre sí, conservando las premisas y conocimientos adquiridos en UX-UI en primer año.
- 2- Transformen este diagrama en un programa funcional desarrollado en C#, modo consola, utilizando los conocimientos de primer año.

Pautas para el desarrollo estipuladas por la empresa para la que trabajan:

- Deberán utilizar:
 - Funciones con parámetro de entrada y salida.
 - Bucles.
 - Condicionales.
- Nomenclatura de nombres al momento de declarar estructuras, arreglos, funciones, variables etc.



Un cliente nos pide una entrevista para que le desarrollemos un programa simple, en este caso solamente de control de inicio de sesión.

Para tal fin nos comparte su idea a modo diagrama de flujo y nos dice:

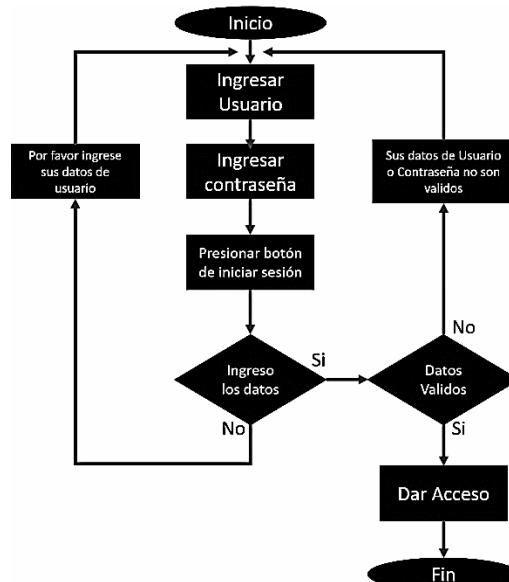
- 4- Deben respetar esta lógica.
- 5- Debe ser informativo e intuitivo (claridad en los mensajes que se emitan al usuario).

La tarea consiste en que:

- 3- Desarrollen/generen un diseño basado en una o varias pantallas, entrelazadas entre sí, conservando las premisas y conocimientos adquiridos en UX-UI en primer año.
- 4- Transformen este diagrama en un programa funcional desarrollado en C#, modo consola, utilizando los conocimientos de primer año.

Pautas para el desarrollo estipuladas por la empresa para la que trabajan:

- Deberán utilizar:
 - Funciones con parámetro de entrada y salida.
 - Bucles.
 - Condicionales.
- Nomenclatura de nombres al momento de declarar estructuras, arreglos, funciones, variables etc.



Un cliente nos pide una entrevista para que le desarrollemos un programa simple, en este caso solamente de control de inicio de sesión.

Para tal fin nos comparte su idea a modo diagrama de flujo y nos dice:

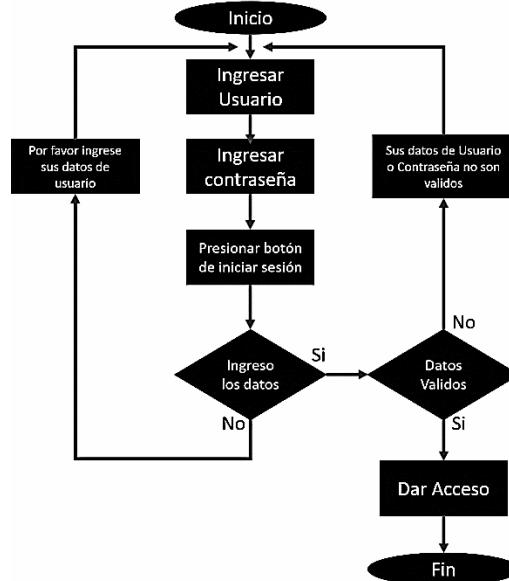
- 5- Deben respetar esta lógica.
- 6- Debe ser informativo e intuitivo (claridad en los mensajes que se emitan al usuario).

La tarea consiste en que:

- 5- Desarrollen/generen un diseño basado en una o varias pantallas, entrelazadas entre sí, conservando las premisas y conocimientos adquiridos en UX-UI en primer año.
- 6- Transformen este diagrama en un programa funcional desarrollado en C#, modo consola, utilizando los conocimientos de primer año.

Pautas para el desarrollo estipuladas por la empresa para la que trabajan:

- Deberán utilizar:
 - Funciones con parámetro de entrada y salida.
 - Bucles.
 - Condicionales.
- Nomenclatura de nombres al momento de declarar estructuras, arreglos, funciones, variables etc.



4. INTRODUCCIÓN A “PROGRAMACIÓN ORIENTADA A OBJETOS”

Seguramente pienses al principio que todas las excelencias de la programación orientada a objetos vienen a ser no más que una mentira, dado que al final sigues programando todo lo que el programa tiene que hacer. Sin embargo te aconsejo que tengas un poco de paciencia: cuando empecemos a desarrollar aplicaciones para Windows verás que no te mentía, al desarrollar programas para Windows es cuando se ve que casi todo está hecho (las ventanas, los botones, las cajas de texto, cuadros de diálogo, etc) y solamente hay que saber usarlos.

Preferí dejar el desarrollo de aplicaciones para Windows al final, puesto que de lo contrario, con tantos objetos, propiedades y eventos hubiera sido mucho más complicado hacer que comprendieras este lenguaje. Por este motivo, empezaremos desarrollando pequeños programas de consola (ejecuciones en una línea de comando como las aplicaciones vistas en C) para que puedas irte familiarizando cómodamente con la sintaxis, sin otras distracciones.

La POO es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

- **Clase:** Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.
- **Herencia:** Por ejemplo, herencia de la clase C a la clase D, es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de POO.
- **Objeto:** Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa).
- **Método:** Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
- **Evento:** Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.
- **Atributos:** Características que tiene la clase.
- **Mensaje:** Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
- **Propiedad o atributo:** Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
- **Estado interno:** Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.
- **Componentes de un objeto:** Atributos, identidad, relaciones y métodos.
- **Identificación de un objeto:** Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

Aclaraciones: En comparación con un lenguaje imperativo, una "variable" no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

Que es la programación orientada a objetos?:

La programación orientada a objetos es algo más que “el último grito en programación”. No se trata de una moda, sino de un modo de trabajo más natural, que te permite centrarte en solucionar el problema que tienes que resolver en lugar de tener que andar pensando en cómo le digo al ordenador que haga esto o lo otro. La programación orientada a objetos (POO en adelante) te abstrae de muchas de estas preocupaciones para que puedas dedicarte a escribir realmente el código útil, es decir, resolver el problema y ya está. Veamos un ejemplo muy claro de lo que quiero decir:

Imagina hacer un programa que mantenga una base de datos de personas. Simple y llanamente. ¿Cómo era esto antes? ¡JA! ¡JAJA! Recoge los datos, abre el archivo, define la longitud del registro, define la longitud y el tipo de cada campo, pon cada campo en su sitio, guarda el registro en el lugar del archivo donde le corresponde y cierra el archivo. Después, para una búsqueda, recoge los datos a buscar, abre el archivo, busca los datos, cierra el archivo, presenta los resultados. Si además permites modificaciones, recoge los nuevos datos, vuelve a abrir el archivo, guarda los datos modificados en el registro que le corresponde, cierra el archivo... Pesado, ¿eh? Ciertamente. La mayor parte del tiempo la dedicábamos a comunicarnos con el ordenador. ¿Cómo sería esto con un lenguaje orientado a objetos, como C#? Mucho más sencillo. Tenemos un objeto Persona. Para agregar un registro, sencillamente habría que dar los valores a dicho objeto y decirle que los guarde. Ya está. Nos da igual cómo el objeto Persona es guardado con los datos. Veámoslo:

```
Persona.Nombre = Pepe
Persona.Apellido = Pepe
Persona.Dirección = la dirección que sea
Persona.Guardar
```

¿Y para buscar? por ejemplo:

```
Persona.Buscar(Pepe)
```

Si lo encuentra, las propiedades Nombre, Apellido y Dirección se habrán rellenado con los datos de Pepe. ¿Cómo lo ha hecho el objeto Persona? Solo por ahora, qué importa! Esto es lo verdaderamente útil de la POO, ya que no tienes que preocuparte de cómo el objeto hace su trabajo. Si está bien construido y funciona no tienes que preocuparte de nada más, sino simplemente de usarlo según tus necesidades.

Si lo piensas un poco, no se trata de un sistema arbitrario, o de una invención particular de algún iluminado. Pongamos por ejemplo que, en lugar de diseñar un programa, estás conduciendo un coche. ¿Qué esperas que suceda cuando pisas el acelerador? Simplemente que el coche acelere, claro. Ahora bien, cómo hace el coche para decirle al motor que aumente de revoluciones no importa. En realidad, da igual que haya un mecanismo mecánico mediante un cable, o un mecanismo electrónico, o si debajo del capó hay un burro y al pisar el acelerador se lo pincha con un punzón en el sitio que más le duela al pobre animal. Además, esto nos lleva a otra gran ventaja: Por mucho que avance la tecnología, el modo de conducir un coche siempre es el mismo, ya que lo único que cambia es el mecanismo interno, no la interfaz que te ofrece. Esto mismo es aplicable a los objetos en programación: por mucho que cambien las versiones de los objetos para hacerlos más eficientes, estos siempre ofrecerán la misma interfaz, de modo que podrás seguir utilizándolos sin necesidad de hacer modificación alguna cuando aparezca una nueva versión del objeto.

Clases y objetos

Ya hemos visto algunas de las principales ventajas de la POO. Vamos a entrar ahora en más detalles: qué son las clases, qué son los objetos y en qué se diferencian.

A menudo es fácil confundir ambos términos. ¿Ambas cosas son iguales? No, para nada, aunque están íntimamente relacionados. Para que pueda haber un objeto debe existir previamente una clase, pero no al revés. Me explico: la clase es la “plantilla” en la que nos basamos para crear el objeto. Volvamos al ejemplo del coche: todos ellos tienen una serie de características comunes: todos tienen un motor, ruedas, un volante, pedales, chasis, carrocería...; todos funcionan de un modo parecido para acelerar, frenar, meter los cambios, encender las luces...; sin embargo, cada uno de ellos es diferente de los demás, dado que cada uno es de su marca, modelo, color, diseño,..., propiedades que lo diferencian de los demás, aunque una o varias de

ellas puedan coincidir en varios coches. Diríamos entonces que todos los coches están basados en una plantilla, o un tipo de objeto, es decir, pertenecen todos a la misma clase: la clase coche. Sin embargo, cada uno de los coches es un objeto de esa clase: todos

comparten la "interfaz", pero no tienen por qué compartir los datos (marca, modelo, color, etc). Se dice entonces que cada uno de los objetos es una **instancia** de la clase a la que pertenece, es decir, un objeto. En resumen, la clase es algo genérico (la idea que todos tenemos sobre lo que es un coche) y el objeto es algo mucho más concreto (el coche del vecino, el nuestro, el papamóvil...). Veamos cómo sería esto en C#. El diseño de la clase Coche sería algo parecido a esto (aunque más ampliado):

```
class Coche
{
    public Coche(string marca, string modelo, string color, string nummotor)
    {
        this.Marca=marca;
        this.Modelo=modelo;
        this.Color=color;
        this.Nummotor=nummotor;
    }
}

class EjemploCoche
{
    static void Main()
    {
        Coche MiCoche=new Coche("Peugeot", "306", "Azul","1546876");

        Console.WriteLine("Los datos de mi coche son:");
        Console.WriteLine("Marca: {0}", MiCoche.Marca);
        Console.WriteLine("Modelo: {0}", MiCoche.Modelo);
        Console.WriteLine("Color: {0}", MiCoche.Color);
        Console.WriteLine("Número de motor: {0}", MiCoche.Nummotor);

    }
}
```

El resultado que aparecería en la consola al ejecutar este programa sería este:

```
Los datos de mi coche son los siguientes:
Marca: Peugeot
Modelo: 306
Color: Azul
Número de motor: 1546876
```

No te preocupes por no entender todo el código todavía, ya hablaremos largo y tendido de la sintaxis. Sólo quiero que te fijes en que en la clase es donde se definen todos los datos y se programan todas las acciones que manejan los objetos de esta clase. Sin embargo, el objeto, MiCoche (creado en la primera línea del método Main) no define absolutamente nada. Simplemente usa la interfaz diseñada en la clase (la interfaz de una clase es el conjunto de métodos y propiedades que esta ofrece para su manejo). Por lo tanto, Coche es la clase y MiCoche un objeto de esta clase.

El Programa completo de uso de la clase coche sería:

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

namespace ConsoleApplication3
{
    class Coche
    {
        string Marca;
        string Modelo;
        string Color;
        string Nummotor;

        public Coche(string marca, string modelo, string color, string nummotor)
        {
            this.Marca = marca;
            this.Modelo = modelo;
            this.Color = color;
            this.Nummotor = nummotor;
        }
    }
}
```

```

        this.Color = color;
        this.Nummotor = nummotor;
    }

    static void Main()
    {
        Coche MiCoche = new Coche("Peugeot", "306", "Azul", "1546876");
        Console.WriteLine("Los datos de mi coche son:");
        Console.WriteLine("Marca: {0}", MiCoche.Marca);
        Console.WriteLine("Modelo: {0}", MiCoche.Modelo);
        Console.WriteLine("Color: {0}", MiCoche.Color);
        Console.WriteLine("Número de motor: {0}", MiCoche.Nummotor);

        Thread.Sleep(3000);
    }
}
=====
```

El siguiente es un programa que muestra en pantalla operaciones simples entre dos numeros:

```

using System;
using System.Collections.Generic;
using System.Threading;

namespace aritmetica
{
    class MainClass
    {
        /*
         * Operaciones aritméticas
         */
        public static void Main(string[] args)
        {
            int dato1, dato2, resultado;

            dato1 = 20;
            dato2 = 10;

            // Suma
            resultado = dato1 + dato2;

            System.Console.WriteLine("{0} + {1} = {2}", dato1, dato2, resultado);

            // Resta
            resultado = dato1 - dato2;
            System.Console.WriteLine("{0} - {1} = {2}", dato1, dato2, resultado);

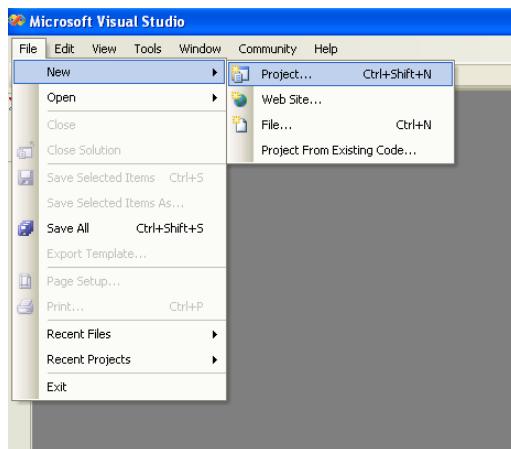
            // Producto
            resultado = dato1 * dato2;
            System.Console.WriteLine("{0} * {1} = {2}", dato1, dato2, resultado);

            // Cociente
            resultado = dato1 / dato2;
            System.Console.WriteLine("{0} / {1} = {2}", dato1, dato2, resultado);

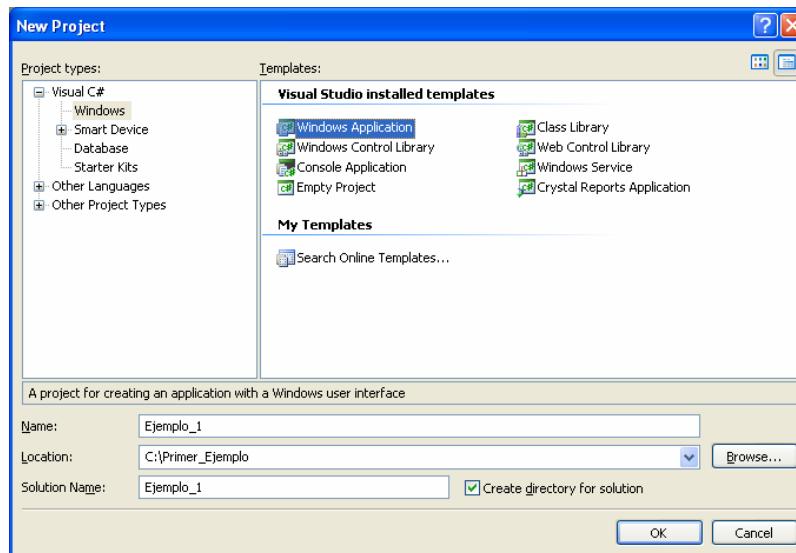
            Thread.Sleep(5000);
        }
    }
}
```

5. EJEMPLO PASO A PASO DEL PRIMER PROGRAMA EN VISUAL C#

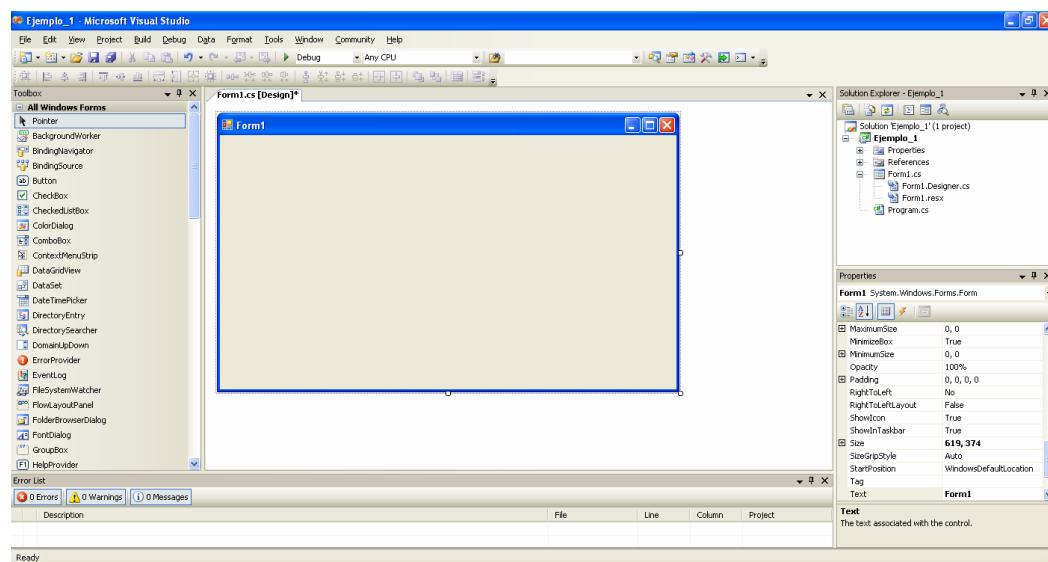
Lo primero que debemos hacer es abrir un nuevo proyecto en nuestro Visual Studio 2005/2008.



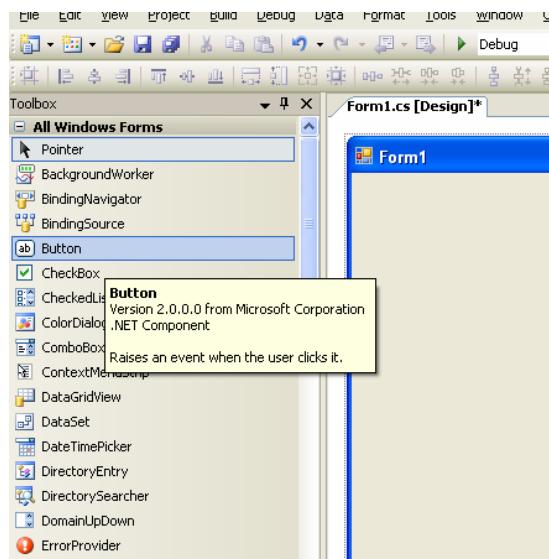
Dicho proyecto en tendrá que ser en el lenguaje "Visual C#". Será un proyecto "Windows" y del tipo "Windows Application".



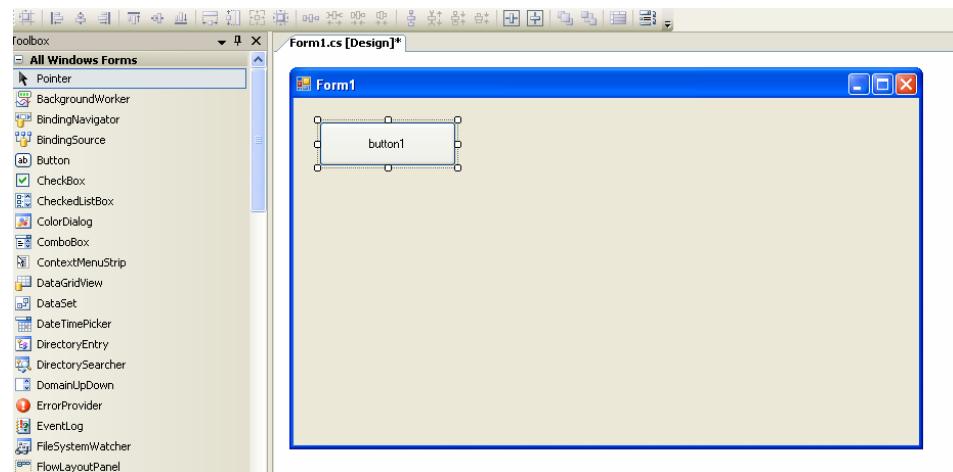
Una vez elegido este tipo de proyecto, indicado el lugar en donde se creara y colocado el nombre, se dará click al botón ok y se verá algo como lo siguiente:



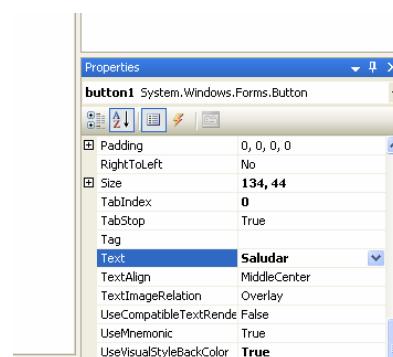
Lo primero que haremos será colocar un botón en nuestra ventana de la aplicación (de hoy en adelante dicha ventana se llamará "Form" o "Formulario"):



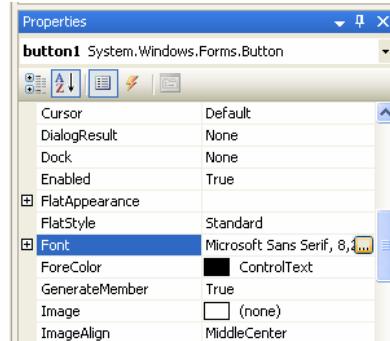
Para colocar dicho botón, una vez ubicado el objeto llamado "Button" en la lista de "toolbox", lo único que se hace es arrastrar como quien mueve un ícono o un archivo de un lado a otro y se lo coloca sobre el formulario en la ubicación deseada:



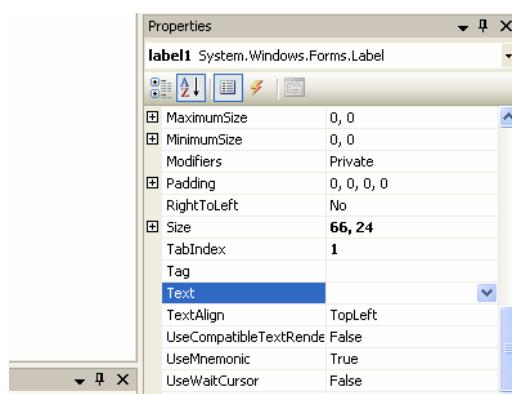
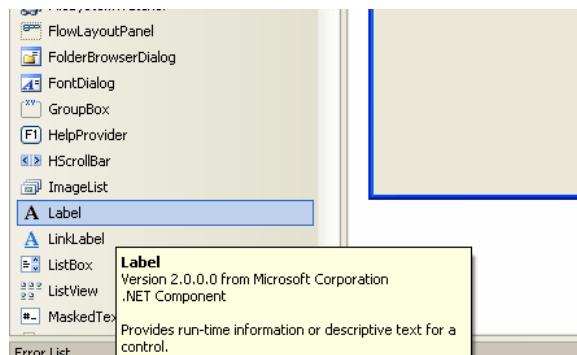
Se le cambiará el texto que aparece sobre el botón, para ello iremos a la ventana de propiedades de dicho objeto y en la propiedad text se colocará por ejemplo "Saludar" como se muestra en la siguiente imagen:



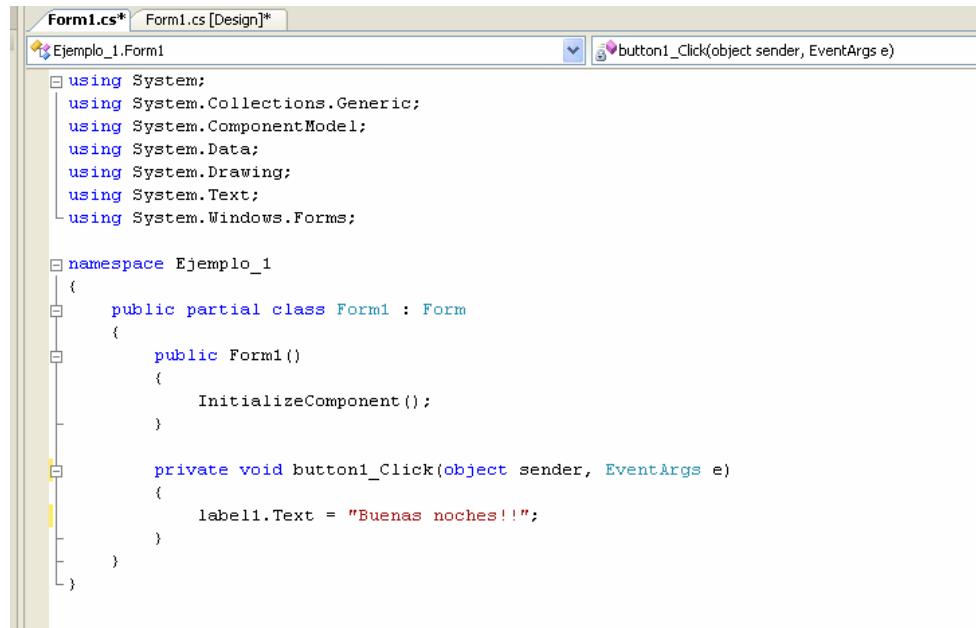
Le cambiaremos el tamaño de la letra de dicho texto (solo para que se vea mejor):



De la misma forma que agregamos un botón agregaremos un objeto llamado "Label". Dicho objeto sirve tanto como para colocar nombre a otros objetos con el fin de indicarle al usuario para que sirve o bien como en este caso para mostrar un texto en él.



Ahora, haciendo doble click sobre el botón agregado al formulario nos lleva al código fuente de nuestro programa. Verán que el compilador ya genera un código por defecto y que el cursos queda a la espera dentro del bloque llamado "private void button1_Click(object sender, EventArgs e)". Esto es en el evento Click del botón, osea, que debería suceder cuando el usuario hace Click sobre ese botón, en este caso deberá colocar un texto en el objeto label (se asigna un valor a la propiedad Text del label) como se muestra en el siguiente ejemplo:



```

Form1.cs*  Form1.cs [Design]*

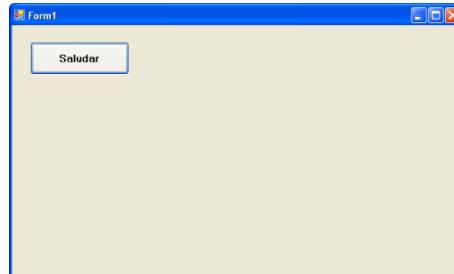
Ejemplo_1.Form1
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Ejemplo_1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

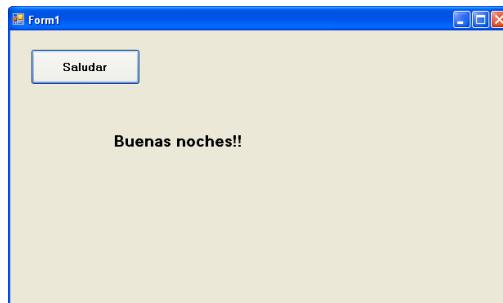
        private void button1_Click(object sender, EventArgs e)
        {
            label1.Text = "Buenas noches!!";
        }
    }
}

```

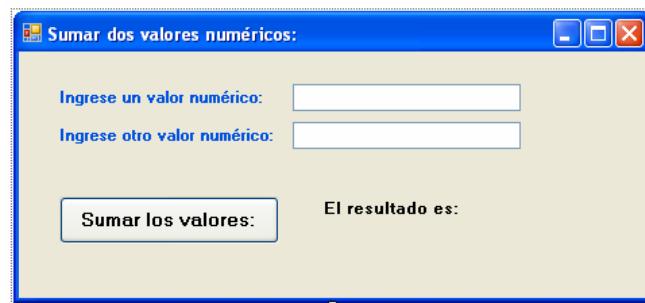
Al ejecutar la aplicación, se abrirá un programa de Windows, aparecerá la siguiente ventana:



Al hacer clic sobre el botón agregado aparecerá la frase que le asignaron al label:



Ahora, comenzemos a generar algo un poco más amigable y útil. Agreguemos algunos otros objetos como por ejemplo más labels y un par de "TextBox" (cajas de texto) de forma tal que quede algo como la imagen que se muestra a continuación:



No hace falta explicar mucho que es lo que hará el programa no? Simplemente, luego de colocar los valores en los textbox y hacer click en el botón, sumará ambos valores y los mostrará en otro objeto.

Para que esto suceda, al igual que en el ejemplo anteriores, en el evento Click del botón se deberá colocar el siguiente código:

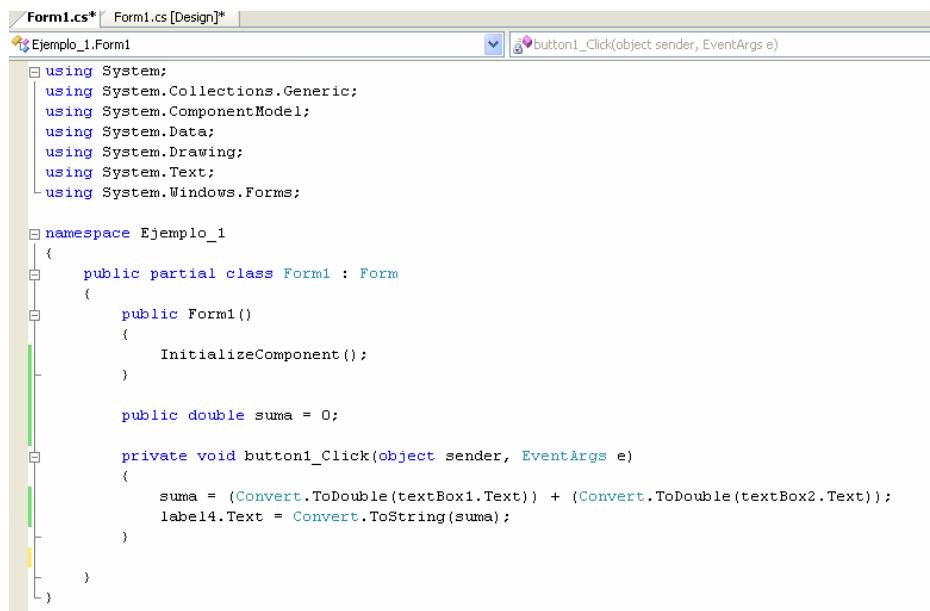
```
suma = (Convert.ToDouble(textBox1.Text)) + (Convert.ToDouble(textBox2.Text));
label4.Text = Convert.ToString(suma);
```

En la primera línea, lo que hace es asignarle un valor a una variable llamada "suma". Ese valor es el resultado de la suma de otros dos valores (los ingresados por el usuario).

Para poder realizar operaciones entre ellos, se los tiene que convertir a valores numéricos porque todo lo que se ingresa por teclado es texto. Utilizando la función "Convert" con la propiedad ".ToDouble" y eso aplicado sobre la propiedad "Text" del objeto (en este caso los textbox) será suficiente.

Luego, simplemente se deberá asignar el resultado al objeto en donde se mostrará (como se mostró en la segunda línea de código). Pero, antes de asignarlo se deberá convertir en texto dado que es un dato de tipo numérico.

Todo el código de nuestro programa será el siguiente:



```
Form1.cs* | Form1.cs [Design]* | Ejemplo_1.Form1 | button1_Click(object sender, EventArgs e)
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Ejemplo_1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        public double suma = 0;

        private void button1_Click(object sender, EventArgs e)
        {
            suma = (Convert.ToDouble(textBox1.Text)) + (Convert.ToDouble(textBox2.Text));
            label4.Text = Convert.ToString(suma);
        }
    }
}
```

6. Cosas útiles en Visual C#

Abrir otro formulario desde un botón:

```
private void button1_Click(object sender, EventArgs e)
{
    Formulario form = new Formulario();
    form.ShowDialog();           <- Formulario modal
    form.Show();                <- Formulario no modal
}
```

Reemplaza Formulario por el nombre del formulario que deseas mostrar.

modal = del tipo messagebox (se tiene que cerrar para trabajar con el resto de los formularios abiertos).
no modal = se puede cambiar de formulario sin cerrar el mismo.

Cerrar un formulario con un botón:

```
private void buttonSalir_Click(object sender, EventArgs e)
{
    This.Close();
}
```

Preguntar si el contenido cargado por ejemplo en un textbox es numérico:

Agregar uno de los siguientes bloques según corresponda dentro de la clase:

```
public static bool IsNumeric(object value)
{
    try
    {
        double d = System.Double.Parse(value.ToString(),
System.Globalization.NumberStyles.Any);
        return true;
    }
    catch (e = System.FormatException)
    {
        return false;
    }
} //IsNumeric

public static bool IsDecimal(string theValue
{
    try
    {
        Convert.ToDouble(theValue);
        return true;
    }
    catch
    {
        return false;
    }
} //IsDecimal

public static bool IsInteger(string theValue)
{
    try
    {
        Convert.ToInt32(theValue);
        return true;
    }
    catch
    {
        return false;
    }
}
```

```
    }  
} //IsInteger
```

Luego llamar al procedimiento de esta forma:

```
if (IsNumeric(txtbox.Text))  
{  
    Sentencias/acciones;  
}
```

Reemplazo de método SetFocus en C#

A diferencia de otros lenguajes, en C#, el SetFocos no existe con ese nombre, en lugar de llamar al evento SetFocus se debe usar Select().

Veamos un ejemplo:

```
public Form1()  
{  
    InitializeComponent();  
    txt6.Select();           << "SetFocus" = Select en el txt6 cuando inicia el programa.  
}
```

7. ANEXO 1 - Usando un listBox

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Listas
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Add_Click(object sender, EventArgs e)
        {
            if (textBox1.Text.Length == 0)
            {
                MessageBox.Show("Debe Ingresar un Alumno", "Error");
            }
            else
            {
                int i;
                Boolean actual=true ;
                for (i = 0; i < listBox1.Items.Count;i++ )
                {
                    if(listBox1.Items[i].Equals(textBox1.Text) )
                    {
                        actual = false ;
                        MessageBox.Show("Alumno ya existe");
                        i = listBox1.Items.Count;
                    }
                }
                if (actual == true)
                {
                    listBox1.Items.Add(textBox1.Text );
                    textBox1.Text = "";
                }
            }
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            string[] lista1 = {"Lucas", "Matias", "Martin", "Pablo"};
            string[] lista2 = {"Turismo","Hotelaria","Electricidad" };
            listBox2.Items.AddRange(lista1);
            listBox4.DataSource = lista2;
            listBox4.ClearSelected();
        }
        private void Delete_Click(object sender, EventArgs e)
        {
            if (listBox1.SelectedIndex != -1)
            {
                listBox1.Items.Remove(listBox1.SelectedItem);
            }
            else
                MessageBox.Show("Debe seleccionar un Item");
        }
        private void button5_Click(object sender, EventArgs e)
        {
            listBox4.SelectionMode = SelectionMode.None;
            listBox4.SelectionMode = SelectionMode.One;
        }
        private void button6_Click(object sender, EventArgs e)
        {
            listBox4.SelectionMode = SelectionMode.None;
            listBox4.SelectionMode = SelectionMode.MultiSimple;
        }
        private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
        {
            MessageBox.Show("APROVECHEN EL CÓDIGO!!!");
        }
    }
}

```

8. Clases y Objetos en Visual C#

Visual C# es un lenguaje orientado a objetos puro, lo que significa que todo con lo que vamos a trabajar en este lenguaje son objetos. Un **objeto** es un agregado de datos y de métodos que permiten manipular dichos datos, y un programa en Visual C# no es más que un conjunto de objetos que interaccionan unos con otros a través de sus métodos.

Una **clase** es la definición de las características concretas de un determinado tipo de objetos. Es decir, de cuáles son los datos y los métodos de los que van a disponer todos los objetos de ese tipo. Por esta razón, se suele decir que el **tipo de dato** de un objeto es la clase que define las características del mismo.

En definitiva, como otros lenguajes modernos, Visual C# agrupa campos relacionados, métodos, propiedades y eventos en estructuras de datos denominadas "clases".

Cuando se define una clase, se utiliza cargándola en la memoria. Una clase que se ha cargado en la memoria se denomina *objeto* o *instancia*.

Se pueden crear instancias de cualquier número de objetos con la misma clase. Es muy común tener matrices o listas que contienen muchos objetos de la misma clase. Cada instancia de la clase ocupa su propio espacio de memoria.

Sintaxis básica de definición de una clase:

```
class <nombreClase>
{
    <miembros>
}
```

Los campos de un objeto son a su vez objetos.

Ejemplo:

```
class Persona
{
    string Nombre;
    int Edad;
    string Apellido;
}
```

De esta forma la clase "Persona" contiene una serie de objetos y cada uno de ellos son de un tipo definido de datos.

Según esta definición, todos los objetos de la clase Persona incorporarán campos que almacenarán cuál es el nombre de la persona que cada objeto representa, cuál es su edad y cuál es su apellido.

Para acceder a un campo de un determinado objeto usaremos el siguiente ejemplo:

```
Public static void Main(string[] args)
{
    Persona persona01 = new Persona();

    Persona.Nombre = "Pepe";
    Persona.Edad = 30;
    Persona.Apellido = "Lopez";
}
```

9. CLASES ABSTRACTAS

POO es el paradigma de programación más usado en el mundo y en la actualidad, incluso desde hace varios años (pueden preguntar a San GOOGLE si tienen dudas). C# está en medio de los primeros 10 más usados dentro de este paradigma.

Clases y Objetos:

Una clase es una plantilla en la que se basa la creación de un Objeto, un Objeto es una instancia de la clase a la que pertenece.

La clase tiene las definiciones de todo lo que compone o hace un Objeto.

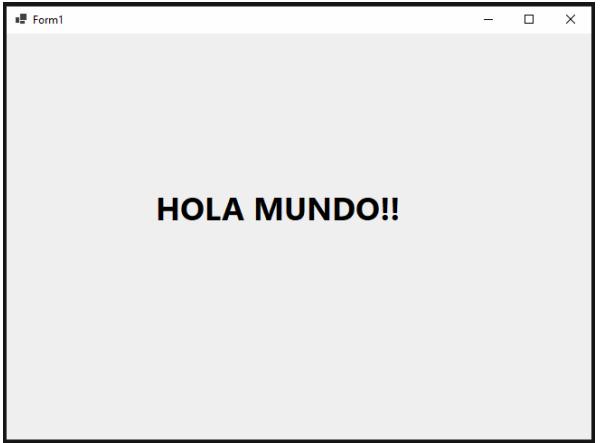
Tipos de clases: Abstractas y Concretas

- Abstractas: Las clases abstractas en C# son un tipo especial de clase que se utiliza para definir la funcionalidad común entre clases relacionadas. Esto significa que una clase abstracta define los términos para una clase, pero no se puede instanciar directamente.
- Concretas: Son las clases que pueden ser instanciadas por lo cual se pueden crear objetos de ellas de forma directa.

Los Objetos dentro de POO:

La Programación Orientada a Objetos es un paradigma de programación que parte del concepto de "objeto" como base, los cuales contienen información en forma de campos (en algunos casos también referidos como Atributos y Propiedades) y código en forma de Métodos.

Ejemplo (Windows Form):

Código	Resultado de ejecución
<pre>using System.Windows.Forms; namespace HolaMundo { public partial class Form1 : Form { public Form1() { InitializeComponent(); } private void Form1_Load(object sender, EventArgs e) { Robot robotin = new Robot(); label1.Text = robotin.Saludar(); } } public abstract class Saludo { public abstract string Saludar(); } public class Robot : Saludo { public override string Saludar() { return "HOLA MUNDO!!"; } } }</pre>	

10. CREANDO Y UTILIZANDO CLASES GLOBALES

Es momento de comenzar a usar más ventajas del lenguaje orientado a objetos, por ello vamos a ver cómo crear una estructura que nos permita declarar variables y funciones que las pueda usar de manera global dentro de toda nuestra solución o programa.

Para poder crear esa estructura tenemos que recordar el concepto de “**clase**”:

*Una **clase** es una estructura de datos que combina estados (campos) y acciones (métodos y otros miembros de función) en una sola unidad. Una **clase** proporciona una definición para instancias de la **clase**, también conocidas como objetos.*

Otra definición de “**clase**” es:

*Una **clase** es una plantilla en la que nos basamos para crear un **objeto**, y un **objeto** es una instancia de la **clase** a la que pertenece.*

Para recordar viejos tiempos, primero vamos a aplicar estos conceptos en un programa de consola. Procedemos a crearla y utilizarla:

```
using System;
namespace EspecialFunciones
{
    public static class Global
    {
        public static string nombre;
    }
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Buen día,\nPor favor Ingrese su nombre y presione ENTER:");
            Global.nombre = Console.ReadLine();
            Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");
            Console.ReadKey();
        }
    }
}
```

Una **static class** es aquella **clase** que se usa sin necesidad de realizar una instanciación de la misma. Se utiliza como una unidad de organización para métodos no asociados a objetos particulares.

En este caso sería **Clase.Atributo => Global.nombre**

En donde “Global” es el nombre de la clase y “nombre” es uno de los atributos que utilizaremos como variable global.

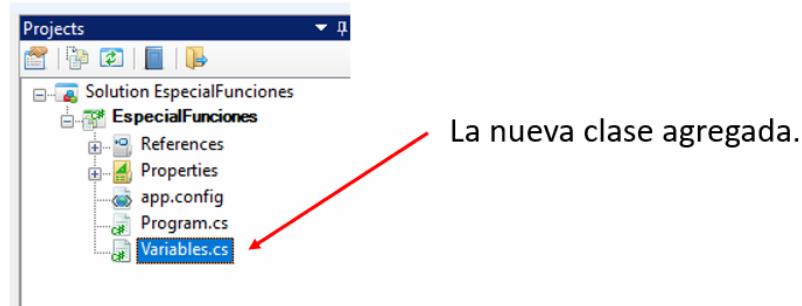
Usamos el atributo “nombre” de la clase “Global” para guardar nuestro dato.

Resultado de la ejecución:

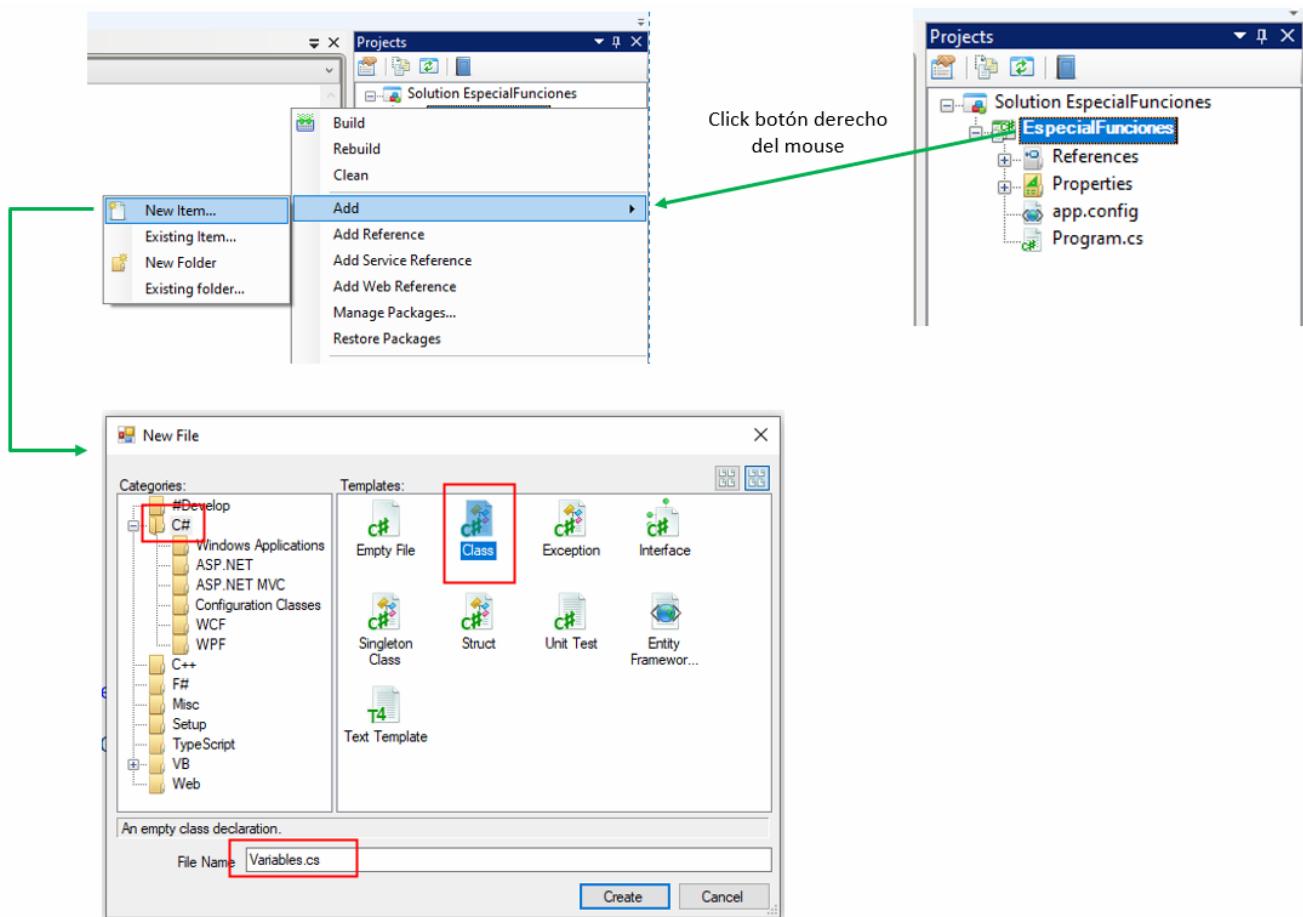
```
C:\Programacion1\EspecialFunciones\EspecialFunciones\EspecialFunciones\bin\Debug\Espec
Buen día,
Por favor Ingrese su nombre y presione ENTER:Leandro
Como esta hoy Sr/a Leandro?
```

La otra opción es crear una clase que puede ser nuestra plantilla de “variables y/o funciones” universal que podemos usar luego en cualquier programa.

Esta clase debe ser creada e invocada como una referencia o librería:



Y como hacemos eso? En la siguiente imagen se muestra los pasos a seguir:



Este es el código que colocaremos, en primera instancia, dentro de esa clase:

```
using System;

namespace Variables
{
    public static class Global
    {
        public static string nombre;
    }
}
```

Y así es como la usaremos desde nuestro programa principal:

La nueva clase agregada:

```
using System;

namespace Variables
{
    public static class Global
    {
        public static string nombre;
    }
}
```

Programa principal:

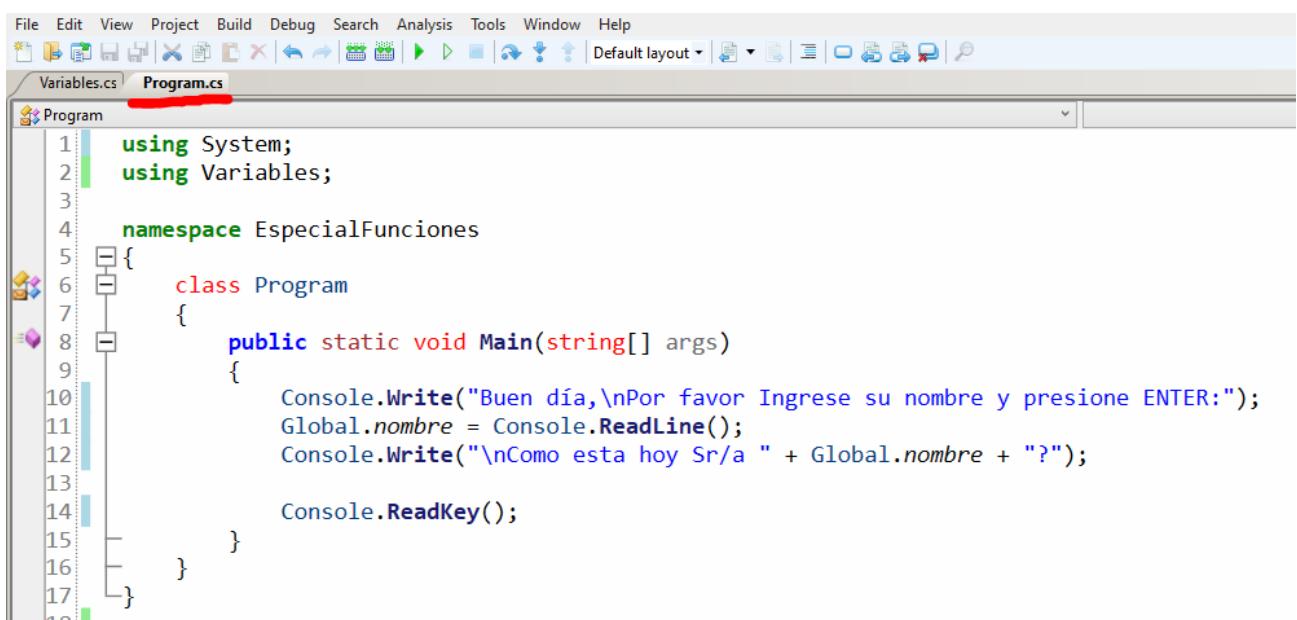
```
using System;
using Variables; // Debemos indicar que use esa clase creada

namespace EspecialFunciones
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Buen día,\nPor favor Ingrese su nombre y presione ENTER:");
            Global.nombre = Console.ReadLine();
            Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");

            Console.ReadKey();
        }
    }
}
```

Entonces:

Contenido del Program.cs:



Contenido de Variables.cs:

```

File Edit View Project Build Debug Search Analysis Tools Window Help
Variables.cs Program.cs
Global
1  using System;
2
3  namespace Variables
4  {
5      public static class Global
6      {
7          public static string nombre;
8      }
9

```

Pasemos ahora el tema “Funciones”:

- Las funciones son algoritmos separados del cuerpo principal del programa que realizan tareas específicas.
- Las funciones hacen que el programa sea más modular.
- Las funciones ayudan a los programadores a ser ordenados en su código y durante todo el desarrollo.
- Las funciones solo deben ser depuradas una vez ya que una vez funcionales no es necesarios volver sobre ellas sino que simplemente se las usa.
- Las funciones ayudan al seguimiento y corrección de errores.

Las mismas pueden ser de los siguientes tipos:

- e- Función que ejecuta ciertas acciones pero que lo hace sin parámetros de entrada ni de salida (no recibe ni devuelve valores).

```

public static void maquillaje()
{
    Console.WriteLine("Buen día, espere por favor... el programa se está inciendo...");
    Thread.Sleep(2500);
}

```

- f- Función que recibe parámetro de entrada, pero no retorna un resultado.

```

public static void SUMA(int valor1, int valor2)
{
    int resultado = valor1 + valor2;
    Console.WriteLine("El resultado es: " + resultado);
}

```

- g- Función que retorna un resultado sin recibir valores (parámetros) de entrada.

```

public static int MULTIPLICA()
{
    int resultado = 45 * 96;
    return resultado;
}

```

- h- Función que recibe valores de entrada y devuelve (retorna) valores resultantes.

```

public static int RESTA(int valor1, int valor2)
{
    int resultado = valor1 - valor2;
    return resultado;
}

```

Como se las utiliza dentro de un programa:

```

public static void Main(string[] args)
{
    maquillaje(); // Llamada a función sin parámetros.

    Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
    Global.nombre = Console.ReadLine();
    Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");

    SUMA(25, 46); // Llamada a función con parámetros de entrada.

    int result_multi = MULTIPLICA(); // Llamada a función con parámetros de salida
    Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);

    int result_resta = RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
    Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);

    Console.ReadKey();
}

```

```

Variables.cs | Program.cs
Program
1  using System;
2  using System.Threading;
3  using Variables;
4
5  namespace EspecialFunciones
6  {
7      class Program
8      {
9          public static void Main(string[] args)
10         {
11             maquillaje(); // Llamada a función sin parámetros.

12             Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
13             Global.nombre = Console.ReadLine();
14             Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");

15             SUMA(25, 46); // Llamada a función con parámetros de entrada.

16             int result_multi = MULTIPLICA(); // Llamada a función con parámetros de salida
17             Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);

18             int result_resta = RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
19             Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);

20             Console.ReadKey();
21         }

22         public static void maquillaje()
23         {
24             Console.WriteLine("Buen día, espere por favor... el programa se está inciando...");
25             Thread.Sleep(2500);
26         }

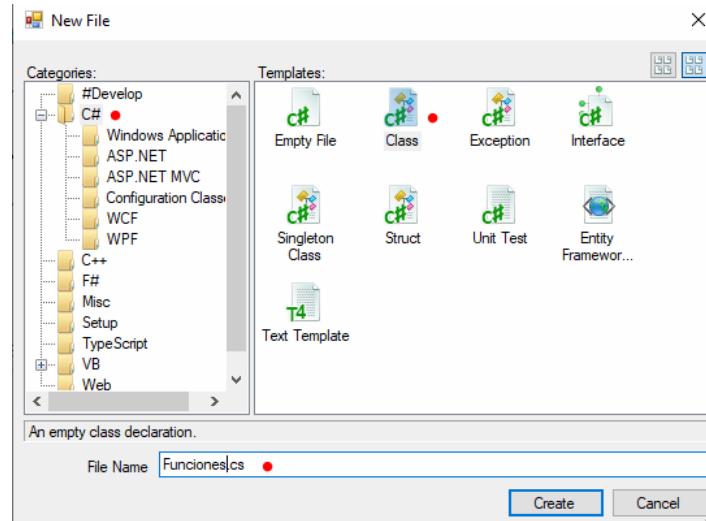
27         public static void SUMA(int valor1, int valor2)
28         {
29             int resultado = valor1 + valor2;
30             Console.WriteLine("\n\nEl resultado de sumar es: " + resultado);
31         }

32         public static int MULTIPLICA()
33         {
34             int resultado = 45 * 96;
35             return resultado;
36         }

37         public static int RESTA(int valor1, int valor2)
38         {
39             int resultado = valor1 - valor2;
40             return resultado;
41         }
42     }
43 }

```

Ahora vamos a unir los dos conceptos, y crearemos una segunda clase que la llamaremos “Funciones” y la utilizaremos desde nuestro programa principal:



Código a colocar en esa clase:

```
using System;
using System.Threading;

namespace Funciones
{
    public static class Funcion
    {
        public static void maquillaje()
        {
            Console.WriteLine("Buen día, espere por favor... el programa se está inciando...");
            Thread.Sleep(2500);
        }

        public static void SUMA(int valor1, int valor2)
        {
            int resultado = valor1 + valor2;
            Console.WriteLine("\n\nEl resultado de sumar es: " + resultado);
        }

        public static int MULTIPLICA()
        {
            int resultado = 45 * 96;
            return resultado;
        }

        public static int RESTA(int valor1, int valor2)
        {
            int resultado = valor1 - valor2;
            return resultado;
        }
    }
}
```

Como las usamos desde nuestro programa principal:

Con el mismo concepto y metodología que usamos con la clase de las Variables...

```

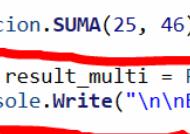
using System;
using System.Threading;
using Variables;
using Funciones;   

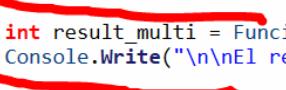
namespace EspecialFunciones
{
    class Program
    {
        public static void Main(string[] args)
        {
            Funcion.maquillaje(); // Llamada a función sin parámetros.  

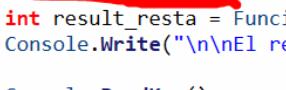
            Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
            Global.nombre = Console.ReadLine();
            Console.WriteLine("\nComo está hoy Sr/a " + Global.nombre + "?");  

            Funcion.SUMA(25, 46); // Llamada a función con parámetros de entrada.  

            int result_multi = Funcion.MULTIPLICA(); // Llamada a función con parámetros de salida
            Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);  

            int result_resta = Funcion.RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
            Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);  

            Console.ReadKey();
        }
    }
}

```

Algunas consideraciones que deben tener en cuenta al momento de usar funciones:

- 3- Verán que en algunos libros o incluso algunos profesores las llaman “métodos” en lugar de funciones. Yo prefiero separar entre “métodos” que hacen referencia a un objeto en particular con “funciones” que solo hacen referencia a una clase que puede no representar un objeto en si mismo.
- 4- Cuando usen funciones con parámetros de salida recuerden que deben respetar:
 - a. El tipo de dato de salida.
 - b. El tipo de dato con el cual es declarada la función (que debe ser el mismo tipo de dato que devuelve la misma).

Ejemplo:

Llamada a función que devuelve un valor de tipo entero, a una variable de tipo entero le asigno el resultado de invocar a dicha función:

```
int result_multi = Funcion.MULTIPLICA();
```

Esa función esta declara como “int” lo que indica que retorna un dato de tipo entero, y en el return de la función devolvemos una variable que ese de tipo entera:

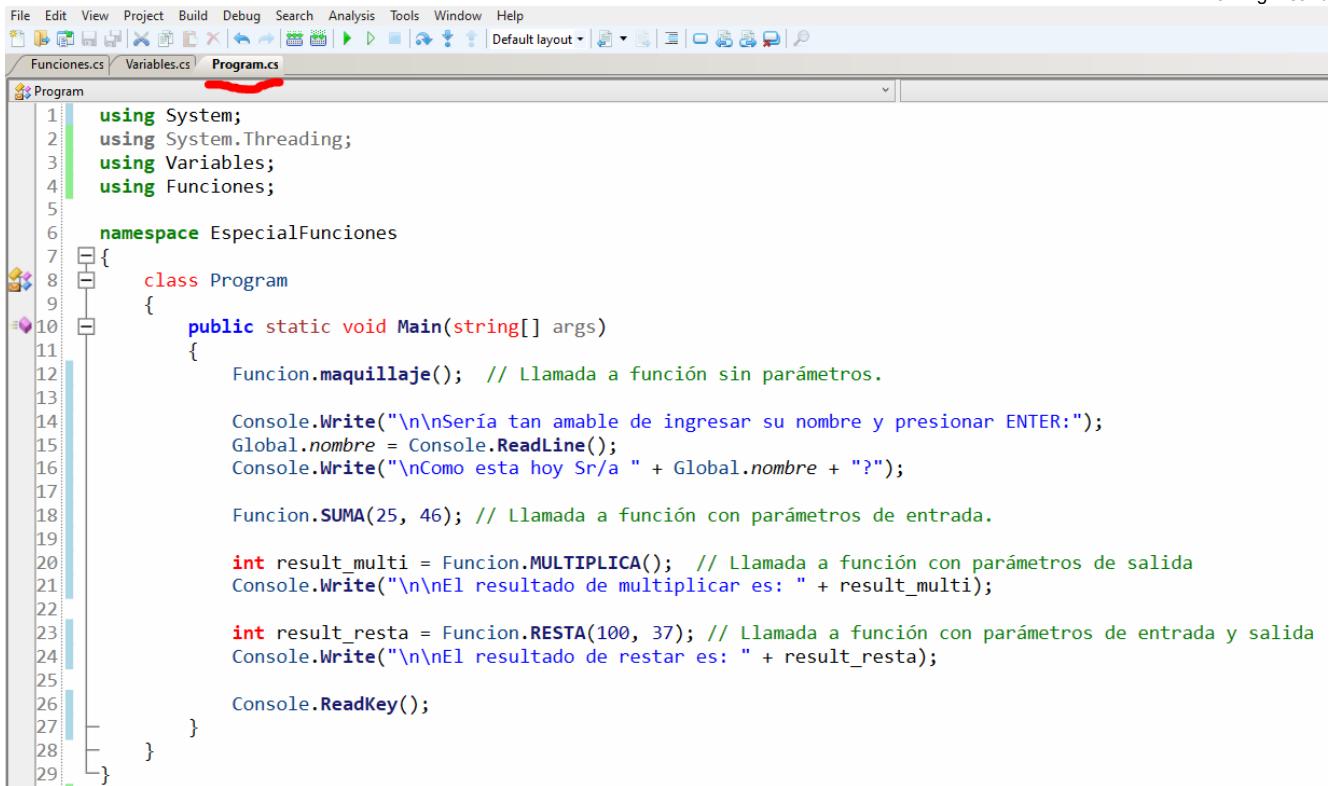
```

public static int MULTIPLICA()
{
    int resultado = 45 * 96;
    return resultado;
}

```

Diferentes vistas del programa terminado...

Vista el programa principal:

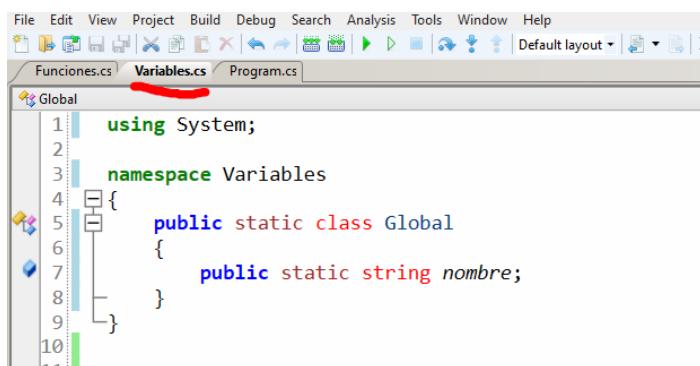


```

1: using System;
2: using System.Threading;
3: using Variables;
4: using Funciones;
5:
6: namespace EspecialFunciones
7: {
8:     class Program
9:     {
10:         public static void Main(string[] args)
11:         {
12:             Funcion.maquillaje(); // Llamada a función sin parámetros.
13:
14:             Console.WriteLine("\n\nSería tan amable de ingresar su nombre y presionar ENTER:");
15:             Global.nombre = Console.ReadLine();
16:             Console.WriteLine("\nComo esta hoy Sr/a " + Global.nombre + "?");
17:
18:             Funcion.SUMA(25, 46); // Llamada a función con parámetros de entrada.
19:
20:             int result_multi = Funcion.MULTIPLICA(); // Llamada a función con parámetros de salida
21:             Console.WriteLine("\n\nEl resultado de multiplicar es: " + result_multi);
22:
23:             int result_resta = Funcion.RESTA(100, 37); // Llamada a función con parámetros de entrada y salida
24:             Console.WriteLine("\n\nEl resultado de restar es: " + result_resta);
25:
26:             Console.ReadKey();
27:         }
28:     }
29:

```

Vista de la Clase Variables:



```

1: using System;
2:
3: namespace Variables
4: {
5:     public static class Global
6:     {
7:         public static string nombre;
8:     }
9:
10:
11:

```

Vista de la clase Funciones:

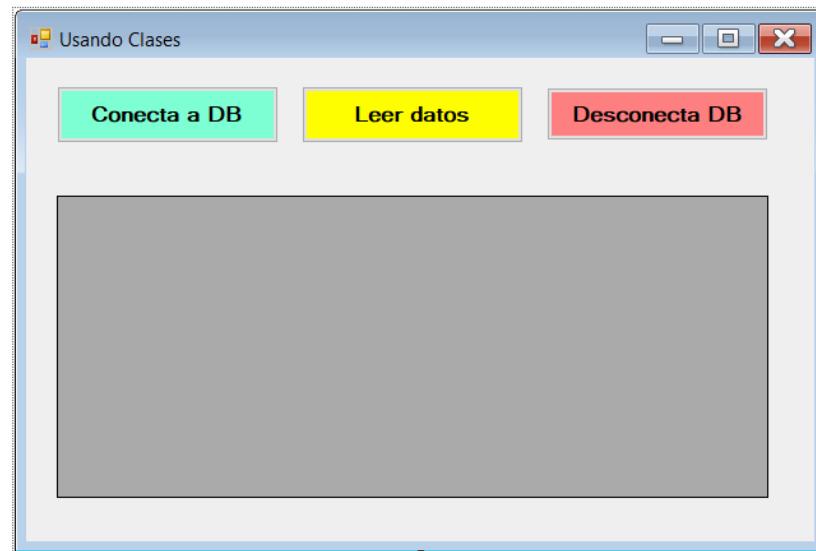
```

1  using System;
2  using System.Threading;
3
4  namespace Funciones
5  {
6      public static class Funcion
7      {
8          public static void maquillaje()
9          {
10             Console.WriteLine("Buen día, espere por favor... el programa se está inciando...");
11             Thread.Sleep(2500);
12         }
13
14         public static void SUMA(int valor1, int valor2)
15         {
16             int resultado = valor1 + valor2;
17             Console.WriteLine("\n\nEl resultado de sumar es: " + resultado);
18         }
19
20         public static int MULTIPLICA()
21         {
22             int resultado = 45 * 96;
23             return resultado;
24         }
25
26         public static int RESTA(int valor1, int valor2)
27         {
28             int resultado = valor1 - valor2;
29             return resultado;
30         }
31     }
32 }
33

```

Ahora, vamos a aplicar los mismos conceptos, pero con una aplicación de Windows Form:

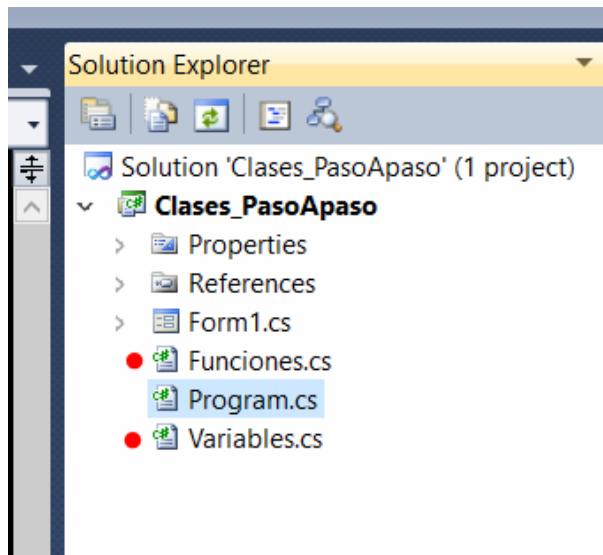
En este ejemplo voy a partir de un desarrollo ya comenzado, en donde simplemente contiene un formulario con los siguientes objetos:



3 Buttons (botones)

1 DataGridView (grilla)

Agregamos dos clases al proyecto:



Una vez agregadas ambas clases, vamos a cambiar/modificar un poco el código por defecto que queda dentro de las mismas, si bien las vamos a usar dentro de nuestra solución la idea es que las mismas queden como clases independientes y no estén atadas a dicha solución.

El código dentro de las mismas nos debería quedar así:

```

Funciones.cs X Variables.cs* Form1.cs [Design]
Funciones.AccesoDB
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Funciones
{
    public static class AccesoDB
    {
    }
}

Variables.paraDB
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Variables
{
    public static class paraDB
    {
    }
}

```

La idea es que, en el cuerpo principal de nuestro programa (la capa de diseño, lo que podríamos llamar FrontEnd aunque ojo, este término es aplicable a desarrollo de aplicaciones o sistemas web con lo cual no es del todo correcto llamarlo así en aplicaciones Windows) nos quede solo la capa de tomar los datos que ingresa el usuario o las acciones sobre los objetos que el mismo ejecuta y, en las clases (las capas de negocio y acceso a Base de Datos a las cuales les podríamos llamar BackEnd con la misma premisa que el concepto anterior) se encuentre toda la lógica del manejo de datos.

Vamos a proceder ahora, paso por paso con el código:

- 1- Lo primero que debemos hacer es completar el Código necesario para que cuando el usuario haga click sobre el botón "Conecta a DB" el programa haga justamente eso, se conecte a la Base de Datos:
 - a- En la clase Variables, vamos a declarar todas las necesarias para establecer la conexión a la Base de Datos (no olvidar agregar la librería necesaria) y les debería quedar de la siguiente manera:

```

using System;
using System.Data.OleDb;

namespace Variables
{
    public static class paraDB
    {
        public static OleDbConnection ConexionConBD;
        public static string strConexion = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=C:\\P00\\Clases_PasoA paso\\AgendaDB.mdb;";
        public static OleDbCommand Orden;
    }
}

```

- b- En la clase Funciones, vamos a incorporar el uso de la clase Variables y vamos a colocar el código necesario para establecer la conexión, les debería quedar de la siguiente manera:

```

using System;
using System.Data.OleDb;
using Variables;

namespace Funciones
{
    public static class AccesoDB
    {
        public static void conectaDB()
        {
            paraDB.ConexionConBD = new OleDbConnection(paraDB.strConexion);
            paraDB.ConexionConBD.Open();
        }
    }
}

```

- c- En el evento Click de nuestro primer botón, vamos a llamar a esa función para que se abra la base de datos, lo haremos además usando un try-catch para asegurarnos de que quede abierta:

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Variables;
using Funciones;

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

        private void btn_conecta_Click(object sender, EventArgs e)
        {
            try
            {
                AccesoDB.conectaDB();
                MessageBox.Show("Base abierta con éxito!!...");
            }
            catch
            {
                MessageBox.Show("ERROR al abrir la DB...");
            }
        }
    }
}

```

- 2- Ahora vamos a completar el Código necesario para que cuando el usuario haga click sobre el botón “Leer Datos” el programa, por medio de la conexión a la DB ya establecida, cargue la grilla como resultado de la ejecución de una consulta sobre una tabla de la DB:

- a- Esta parte es tal vez la que más se les puede/suele complicar, primero vamos a armar la consulta SQL dentro de una variable de tipo string y dentro del evento Click del botón en cuestión:

```
private void btn_leeDB_Click(object sender, EventArgs e)
{
    string consulta = "Select * from Persona";
}
```

- b- En la clase Funciones vamos a crear la función que recibe como parámetro de entrada la consulta, ejecuta la misma sobre la DB ya abierta y devuelve/retorna el resultado de dicha ejecución (y no olvidemos agregar Lector en la clase Variables):

En la clase Variables:

```
public static class paraDB
{
    public static OleDbConnection ConexionConBD;
    public static string strConexion = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=C:\\\\P00\\\\Clases_PasoA paso\\\\AgendaDB.mdb";
    public static OleDbCommand Orden;
    public static OleDbDataReader Lector;
}
```

En la clase Funciones:

```
public static OleDbDataReader lecturaDB(string consulta)
{
    paraDB.Orden = new OleDbCommand(consulta, paraDB.ConexionConBD);
    paraDB.Lector = paraDB.Orden.ExecuteReader();
    return paraDB.Lector;
}
```

- c- Dentro del evento Click del botón:

```
private void btn_leeDB_Click(object sender, EventArgs e)
{
    string consulta = "Select * from Persona";

    DataTable dt = new DataTable();
    dt.Load(AccesoDB.lecturaDB(consulta));
    dtg_datos.DataSource = dt;
    dtg_datos.ClearSelection();
}
```

- 3- Ahora completamos el Código necesario para que cuando el usuario haga click sobre el botón “Desconecta DB” se proceda con el cierre de la Base de Datos:

- a- Dentro de la clase Funciones creamos la nueva función:

```
public static void cerrarDB()
{
    paraDB.ConexionConBD.Close();
}
```

- b- Desde el evento Click sobre el botón en cuestión invocamos dicha función:

```
private void btn_cierraDB_Click(object sender, EventArgs e)
{
    AccesoDB.cerrarDB();
}
```

El código final en cada una de las capas/clases debería ser el siguiente:

```
Form1.cs Funciones.cs Variables.cs Form1.cs [Design]
Funciones.AccesoDB
using System;
using System.Data.OleDb;
using Variables;

namespace Funciones
{
    public static class AccesoDB
    {
        public static void conectarDB()
        {
            paraDB.ConexionConBD = new OleDbConnection(paraDB.strConexión);
            paraDB.ConexionConBD.Open();
        }

        public static OleDbDataReader lecturaDB(string consulta)
        {
            paraDB.Orden = new OleDbCommand(consulta, paraDB.ConexionConBD);
            paraDB.Lector = paraDB.Orden.ExecuteReader();
            return paraDB.Lector;
        }

        public static void cerrarDB()
        {
            paraDB.ConexionConBD.Close();
        }
    }
}
```

```
Form1.cs Funciones.cs Variables.cs Form1.cs [Design]
Variables.paraDB
using System;
using System.Data.OleDb;

namespace Variables
{
    public static class paraDB
    {
        public static OleDbConnection ConexionConBD;
        public static string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=C:\\P00\\Clases_PasoA paso\\AgendaDB.mdb;";
        public static OleDbCommand Orden;
        public static OleDbDataReader Lector;
    }
}
```

Form1.cs X Funciones.cs Variables.cs Form1.cs [Design]

Clases_PasoApaso.Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
using Variables;
using Funciones;

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

        private void btn_conecta_Click(object sender, EventArgs e)
        {
            try
            {
                AccesoDB.conectaDB();
                MessageBox.Show("Base abierta con éxito!!...");
            }
            catch
            {
                MessageBox.Show("ERROR al abrir la DB...");
            }
        }

        private void btn_leeDB_Click(object sender, EventArgs e)
        {
            string consulta = "Select * from Persona";

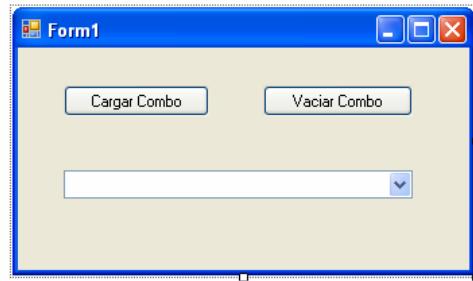
            DataTable dt = new DataTable();
            dt.Load(AccesoDB.lecturaDB(consulta));
            dtg_datos.DataSource = dt;
            dtg_datos.ClearSelection();
        }

        private void btn_cierraDB_Click(object sender, EventArgs e)
        {
            AccesoDB.cerrarDB();
        }
    }
}
```

11. ANEXO 2 - CARGA DE UN ComboBox CON C# PASO a PASO

Un ComboBox puede ser cargado de dos formas: de forma manual (cargando cada dato uno por uno) o de forma automática (en base a una serie de datos que provienen por ejemplo de una consulta a una DB).

Para ello vamos a utilizar el siguiente diseño en donde verás que al formulario agregué 3 objetos: dos botones y un combobox:



Pasos para la carga manual:

La carga manual de un ComboBox es muy simple, solo hay que decirle que agregue el ítem y se hace así:

```
ComboBox.Items.Add(<Datos/elemento>);
```

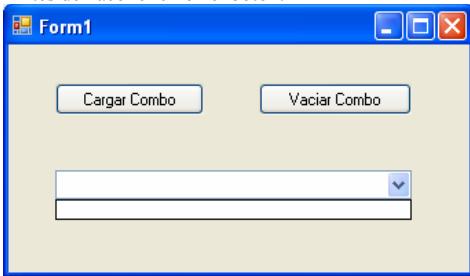
Para borrar los ítems del combo: `ComboBox.Items.Clear();`

Para este ejemplo (en referencia al diseño a utilizar que se muestra en la imagen anterior) la carga se deberá hacer o efectuar cuando se hace click en el botón que dice "Cargar Combo", para ello colocaremos por ejemplo el siguiente código (que como se ve es dentro del evento click del botón):

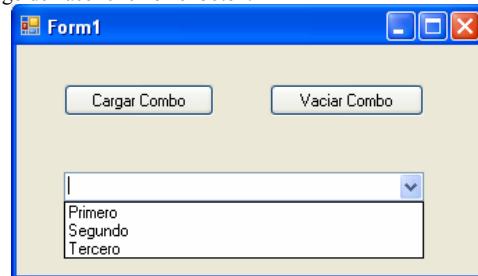
```
private void btn_carga_Click(object sender, EventArgs e)
{
    cmb_1.Items.Add("Primero");
    cmb_1.Items.Add("Segundo");
    cmb_1.Items.Add("Tercero");
}
```

Si vemos causa y efecto en tiempo de ejecución se vería así:

Antes de hacer click en el botón:



Luego de hacer click en el botón:



Pasos para la carga automática:

Modo 1:

Utilizando los mismos métodos y propiedades de la carga manual, ahora usaremos un bucle que me permita recorrer registro a registro que fueron resultantes de una consulta a una tabla de la DB y que dichos registros están cargados en una variable/objeto:

```
while (<Leeo_registro_en_objeto>
{
    ComboBox.Items.Add(<Datos_a_cargar>);
}
```

Si seguimos utilizando el evento click sobre el botón entonces el código sería así:

```
private void btn_carga_Click(object sender, EventArgs e)
{
```

```

ConexionConBD = new OleDbConnection(strConexión);
Consulta = "SELECT * FROM Personas";
Orden = new OleDbCommand(Consulta, ConexionConBD);
ConexionConBD.Open();

Lector = Orden.ExecuteReader();

while (Lector.Read())
{
    cmb_1.Items.Add(Lector["Nombre"]);
}
Lector.Close();
ConexionConBD.Close();
}

```

Por supuestos con la previa declaración de las variables y punteros necesarios:

```

private OleDbConnection ConexionConBD;
private OleDbCommand Orden;
private OleDbDataReader Lector;
private string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=c:\\Registro.mdb;";
private string Consulta;

```

Modo 2:

Este modo es muy similar a la carga de una grilla ya que se usa el objeto de tipo DataTable y se le asigna de forma directa el contenido del mismo, solo hay que indicarle que campo mostrar:

```

Lector = Orden.ExecuteReader();
DataTable dsCombo = new DataTable();
dsCombo.Load(Lector);
cmb_1.DataSource = dsCombo;
cmb_1.DisplayMember = "Nombre"; // Indico que campo de tabla se mostrará

```

Completando con declaraciones de las variables, punteros y demás quedaría algo así:

```

private OleDbConnection ConexionConBD;
private OleDbCommand Orden;
private OleDbDataReader Lector;
private string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" +
    "Data Source=c:\\Registro.mdb;";
private string Consulta;

private void btn_carga_Click(object sender, EventArgs e)
{
    ConexionConBD = new OleDbConnection(strConexión);
    ConexionConBD.Open();
    Consulta = "SELECT * FROM Personas";
    Orden = new OleDbCommand(Consulta, ConexionConBD);

    Lector = Orden.ExecuteReader();
    DataTable dsCombo = new DataTable();
    dsCombo.Load(Lector);
    cmb_1.DataSource = dsCombo;
    cmb_1.DisplayMember = "Nombre";
    cmb_1.ValueMember = "Id"; // Le indico además que el valor de o id de cada ítem es el campo id
    // de la tabla. Más adelante usaremos este dato.

    Lector.Close();
    ConexionConBD.Close();
}

```

12. ANEXO 3 - CARGA DE UNA GRILLA (DataGridView) CON C# PASO a PASO

Para poder realizar la carga de una grilla con datos que en este caso provienen de una DB es necesario contar con una serie de objetos que me permitan: leer los datos de la DB, cargar los datos en un objeto intermedio entre la DB y la grilla, asignar o cargar esos datos en la grilla.

Variables necesarias (de tipo datos, punteros, etc.) que se deben declarar:

```
private OleDbConnection ConexionConBD; // Puntero de la conexión a la DB
private OleDbCommand Orden; // Puntero que tendrá los datos de la conexión y la
                           // consulta a realizar
private OleDbDataReader Lector; // Objeto receptor de los datos resultantes de la
                               // consulta a la DB
private string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" +_
                             "Data Source=c:\\clinica.mdb";
```

Veamos entonces, cada uno de los objetos:

- 1- Leer los datos de la DB: para leer los datos de una DB es necesario poder ejecutar una consulta sobre la misma, para ellos usaremos los siguientes objetos/elementos:

Declaración y carga de una variable de tipo string con la consulta a realizar:

```
string consulta = "Select campo1, campo2, campoX from Tabla";
```

- 2- Carga de los datos en un objeto intermedio entre la DB y la grilla:

```
ConexionConBD = new OleDbConnection(strConexión); // Objeto de Conexión
Orden = new OleDbCommand(consulta, ConexionConBD); // Instanciación del puntero
ConexionConBD.Open(); // Abre la conexión con la DB
Lector = Orden.ExecuteReader(); // Lector recibe los datos de la consulta
```

En este momento tenemos los datos que se reciben de la consulta pero son solo una serie de registros sin estructura. Para poder asignar esos datos de forma directa a una grilla es necesario darles formato/estructura de tabla. Para ello es necesario usar un nuevo objeto que de esa estructura a los datos para luego volcarlos sobre la grilla:

```
DataTable Tabla = new DataTable(); // Se declara un objeto de tipo Tabla de Datos
Tabla.Load(Lector); // Se carga el objeto con el contenido de Lector
```

- 3- Una vez que tenemos los datos en un objeto que tiene el formato de una tabla, y que contiene ya los datos dentro de él, lo que se debe hacer simplemente es asignar esos datos a la grilla:

```
// Se le indica al objeto DataGridView que el origen o proveedor de sus datos es // el
// objeto Tabla, o sea que el DataSource es el objeto Tabla.
dataGridView1.DataSource = Tabla;
```

En este punto ya tenemos todo lo necesario para que los datos que traemos en nuestra consulta se muestren en una grilla una vez que se ejecuta nuestra aplicación como se realiza en el ejemplo que les muestro a continuación.

Tienen que tener bien en claro cuál es el orden correcto para hacer la carga de los datos y que objetos se necesitan para tal fin.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private OleDbConnection ConexionConBD;
        private OleDbCommand Orden;
        private OleDbDataReader Lector;
        private string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0";
                                         "Data Source=c:\\ejemplo.mdb;";
        DataTable Tabla = new DataTable();
        private string Consulta;

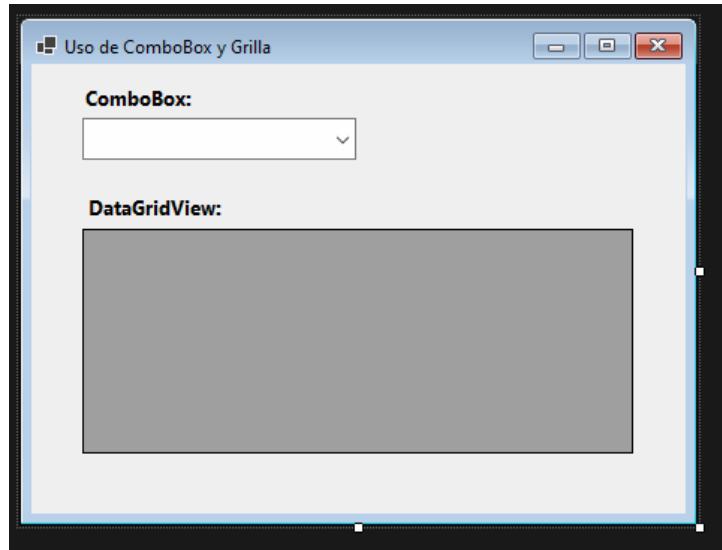
        private void Form1_Load(object sender, EventArgs e)
        {
            ConexionConBD = new OleDbConnection(strConexión);
            Consulta = "SELECT * FROM Tabla";
            Orden = new OleDbCommand(Consulta, ConexionConBD);
            ConexionConBD.Open();

            Lector = Orden.ExecuteReader();
            Tabla.Load(Lector);
            dataGridView1.DataSource = Tabla;
            Lector.Close();

            ConexionConBD.Close();
        }
    }
}
```

13. ANEXO 4 - TIPS USANDO DATAGRIDVIEW Y COMBOBOX

- 1- En un proyecto vacío agregar un ComboBox y un DataGridView (no olvidar renombrar los objetos antes de intentar cargar código) y que quede más o menos así:



- 2- Cargar una lista de elementos al combo, indicando valores para “ValueMember” (un valor oculto al usuario que nos sirve como ID o clave que identifica el dato mostrado) y “DisplayMember” (el valor o dato que se muestra al usuario) al mismo (notar que el código está en el evento Load del formulario):

```
private void frm_principal_Load(object sender, EventArgs e)
{
    // El datagridview (la grilla) no tiene columnas ni filas, con lo cual debemos agregarle las mismas.
    dtg_grilla.Columns.Add("columnDNI", "DNI");
    dtg_grilla.Columns.Add("columnNombre", "Nombre");
    dtg_grilla.Columns.Add("columnIndiceCombo", "Indice del Combo");
    dtg_grilla.Columns[1].Width = 200;

    // Creamos un objeto de tipo tabla de datos (DataTable) que será el origen de los datos del combobox.
    // En este caso creo la tabla de forma manual pero se podría usar un resultado de una consulta a una base de datos.

    DataTable dt = new DataTable();
    dt.Columns.AddRange(new DataColumn[] { new DataColumn("DNI", typeof(int)), new DataColumn("Nombre", typeof(string)) });
    dt.Rows.Add(11111111, "Pepe Perez");
    dt.Rows.Add(22222222, "Rene Le-Puag");
    dt.Rows.Add(33333333, "Carlos Vistaloca");
    dt.Rows.Add(44444444, "Roberto Patadepalo");

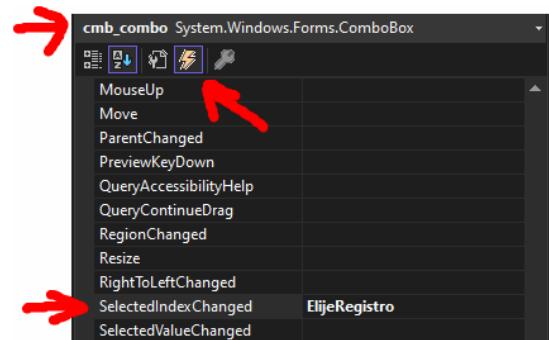
    // Esto lo que hace es agregar una fila que será la que siempre aparece por defecto en el combo
    DataRow row = dt.NewRow();
    row[0] = 0;
    row[1] = "Seleccione un dato";
    dt.Rows.InsertAt(row, 0);

    // Vamos a indicarle al ComboBox quien es su DataSource y que valor es el que se muestra y cual es el que no
    cmb_combo.DataSource = dt;
    cmb_combo.DisplayMember = "Nombre"; // valor a mostrar
    cmb_combo.ValueMember = "DNI"; // valor oculto

    //Quita todas las filas de la grilla
    dtg_grilla.Rows.RemoveAt(0);
}
```

- 3- La idea ahora es, que cuando seleccione un registro en el ComboBox, se cargue el DataGridView mostrando los diversos datos que el combo nos puede ofrecer para su uso:

- a- Vamos a colocarle un “nombre” al evento SelectedIndexChanged del combo:

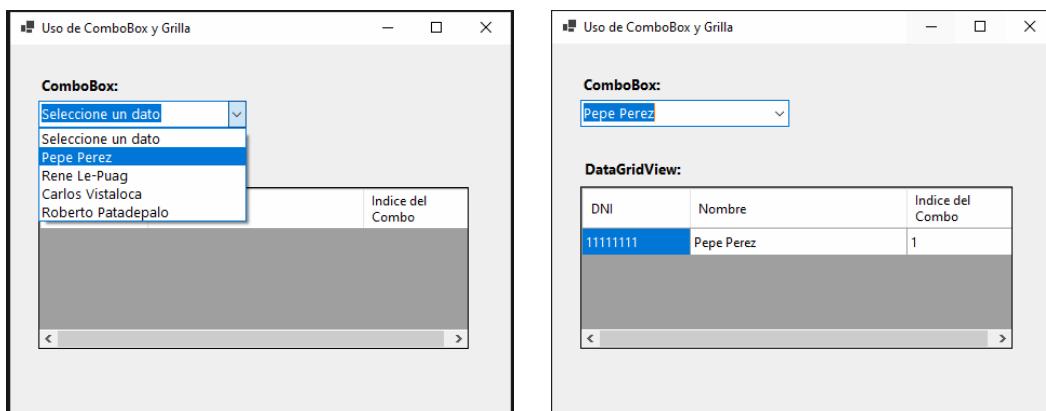


- b- Ahora procedemos a colocar el código necesario dentro de dicho evento/método:

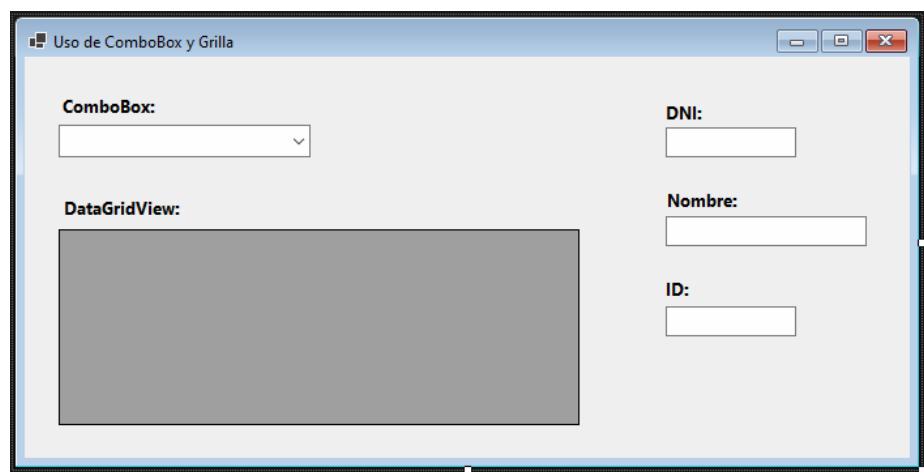
```
private void ElijeRegistro(object sender, EventArgs e)
{
    if(dtg_grilla.Rows.Count == 0)
        dtg_grilla.Rows.Add();

    dtg_grilla.Rows[0].Cells[0].Value = cmb_combo.SelectedValue;
    dtg_grilla.Rows[0].Cells[1].Value = cmb_combo.Text;
    dtg_grilla.Rows[0].Cells[2].Value = cmb_combo.SelectedIndex;
}
```

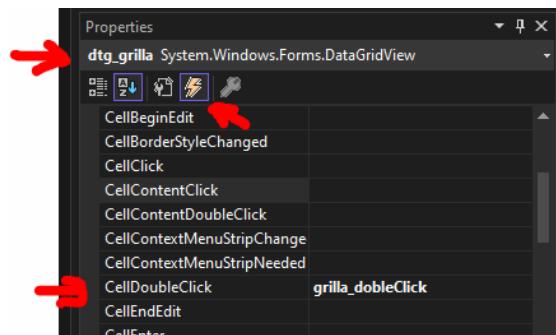
- c- Testigo funcional:



- 4- Ahora vamos a agregar 3 TextBox y la idea es que cuando se haga doble click en la fila de la grilla los valores de dicha fila se repartan en los TextBox:



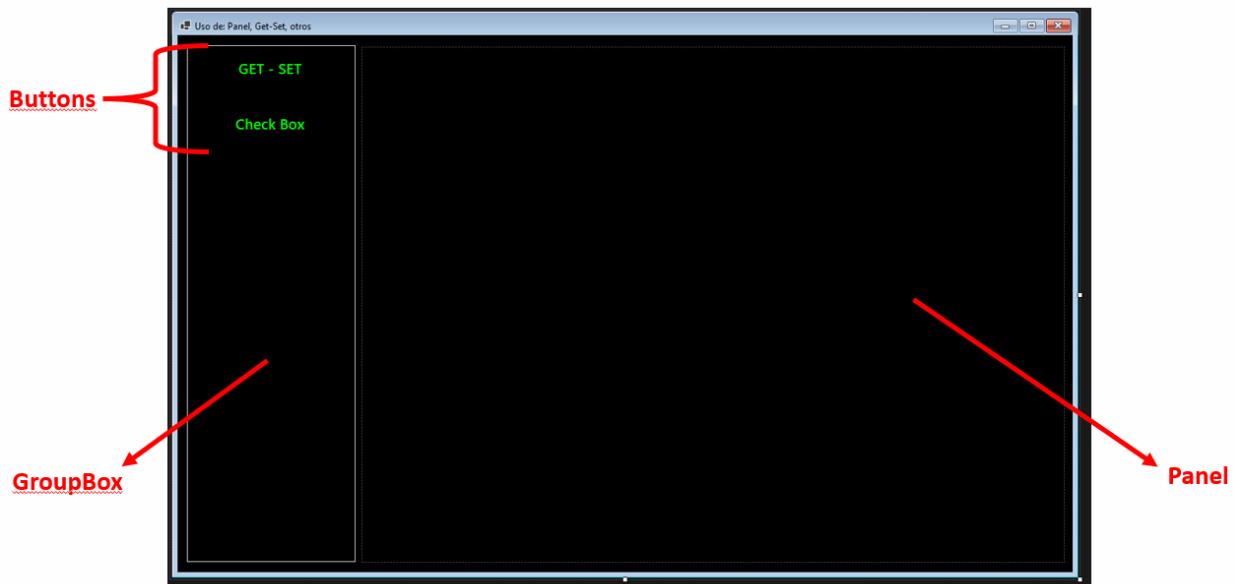
Código para agregar:



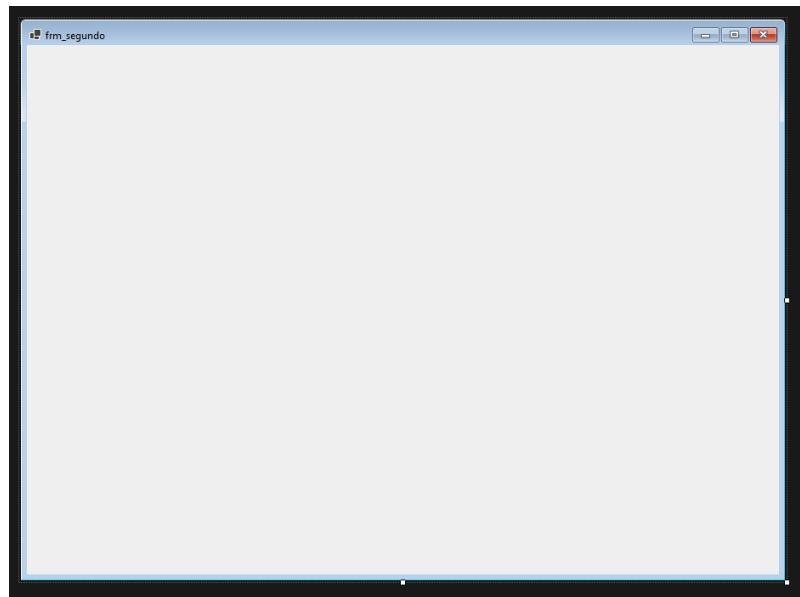
```
private void grilla_dobleClick(object sender, DataGridViewCellEventArgs e)
{
    // El e.RowIndex devuelta el indice de la fila sobre la cual se hizo dobleclick.
    // Para el caso de la columns (Cells) le paso el valor indicando la columna que quiero tomar.
    txt_dni.Text = dtg_grilla.Rows[e.RowIndex].Cells[0].Value.ToString();
    txt_nombre.Text = dtg_grilla.Rows[e.RowIndex].Cells[1].Value.ToString();
    txt_id.Text = dtg_grilla.Rows[e.RowIndex].Cells[2].Value.ToString();
}
```

14. PANEL, GET-SET Y OTROS

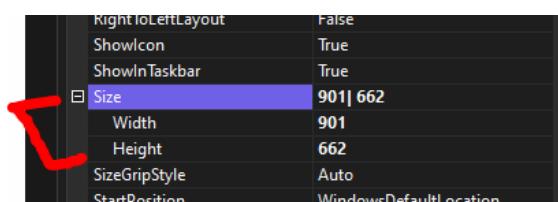
- 1- En un proyecto de Windows Form, agregar los siguientes Botones y customizar sus diseños a un estilo similar al que se muestra en la siguiente imagen (o a gusto). No olvidar renombrar los objetos antes de intentar cargar código.



- 2- Agregamos un segundo formulario al proyecto:



- 3- Le hacemos algunas customizaciones:



Le damos el mismo tamaño que el Panel



Ponemos el valor en None de esa propiedad

- 4- En el evento click del botón “GET – SET” procedemos a cargar y abrir el segundo formulario dentro del panel:

```
private void btn_getset_Click(object sender, EventArgs e)
{
    // Antes de cargar al panel un form me aseguro de que no tenga ninguno cargado
    if(panel.Controls.Count > 0)
        panel.Controls.RemoveAt(0);

    // Se procede a cargar al form dentro del panel y se setean algunas propiedades
    var form = Application.OpenForms.OfType<frm_segundo>().FirstOrDefault();
    frm_segundo segundoFrm = form ?? new frm_segundo();
    segundoFrm.TopLevel = false;
    segundoFrm.FormBorderStyle = FormBorderStyle.None;
    segundoFrm.Dock = DockStyle.Fill;
    panel.Controls.Add(segundoFrm);
    panel.Tag = segundoFrm;
    segundoFrm.Show();
}
```

- 5- Usando Get y Set (en windows form algo muy poco usado dado que es utilizado para valorizar variables o propiedades privadas de objetos):

Agregar estos objetos al segundo formulario:



Código para agregar:

```
private void button1_Click(object sender, EventArgs e)
{
    P obj = new P();
    obj.Name = txt_entrada.Text;
    txt_salida.Text = obj.Name;
}

class P
{
    private string pname;
    public string Name
    {
        get
        {
            return pname;
        }
        set
        {
            pname = value;
        }
    }
}
```

6- Vamos a enviar datos desde el segundo formulario al form principal:

a- Agrego estos objetos al segundo formulario:

SEGUNDO FORMULARIO

Uso de Get - Set:

Pasar

Enviando datos al form principal:

Dato 1:Dato 2:Enviar



b- Agrego estos objetos al form principal:

■ Uso de: Panel, Get-Set, otros

GET - SET

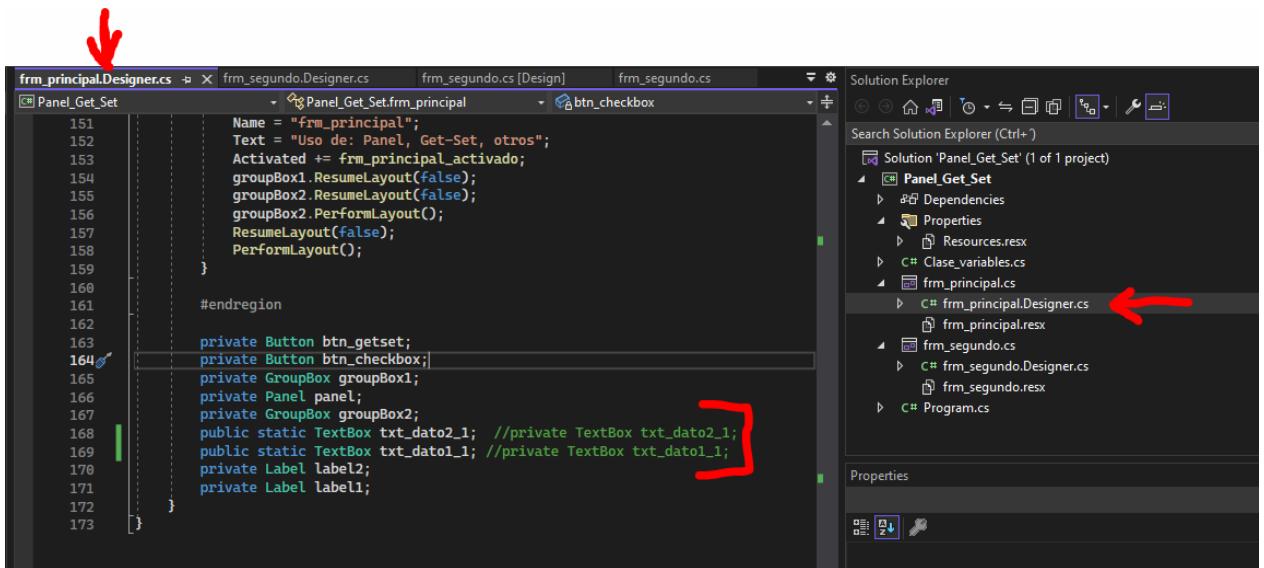
Check Box

Datos recibidos:

Datos 1:Datos 2:



- c- En el frm_principal.Designer.cs cambiar de “private” a “public static” los dos textbox (vamos a pasar datos del form hijo al form padre en este caso):



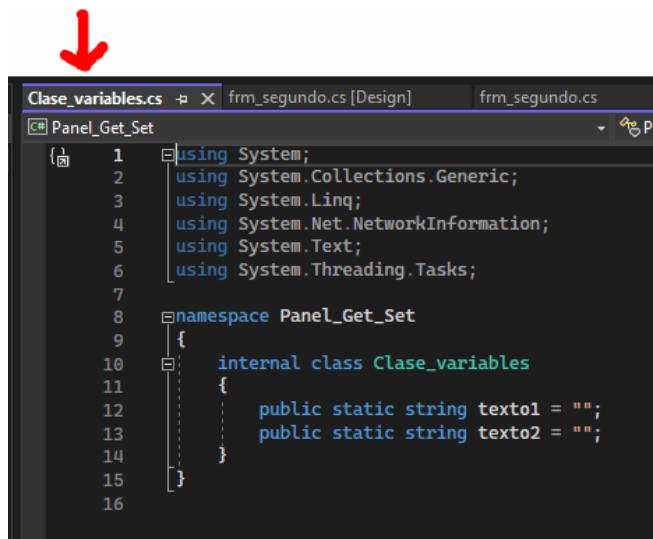
```

151     Name = "frm_principal";
152     Text = "Uso de: Panel, Get-Set, otros";
153     Activated += frm_principal_activado;
154     groupBox1.ResumeLayout(false);
155     groupBox2.ResumeLayout(false);
156     groupBox2.PerformLayout();
157     ResumeLayout(false);
158     PerformLayout();
159   }
160
161 #endregion
162
163 private Button btn_getset;
164 private Button btn_checkbox;
165 private GroupBox groupBox1;
166 private Panel panel;
167 private GroupBox groupBox2;
168 public static TextBox txt_dato2_1; //private TextBox txt_dato2_1;
169 public static TextBox txt_dato1_1; //private TextBox txt_dato1_1;
170 private Label label2;
171 private Label label1;
172
173 }

```

- d- En este caso, y solo por un tema de disponibilidad de los datos en variables globales evitando dedos del usuario, utilizo una Clase de variables que las uso de intermediarias entre los objetos de los formularios:

La clase:

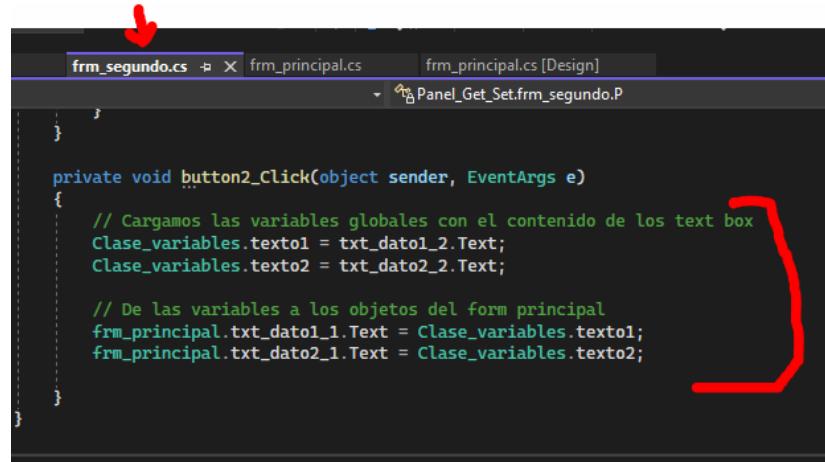


```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Net.NetworkInformation;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace Panel_Get_Set
9 {
10   internal class Clase_variables
11   {
12     public static string texto1 = "";
13     public static string texto2 = "";
14   }
15
16

```

El código en el frm hijo:



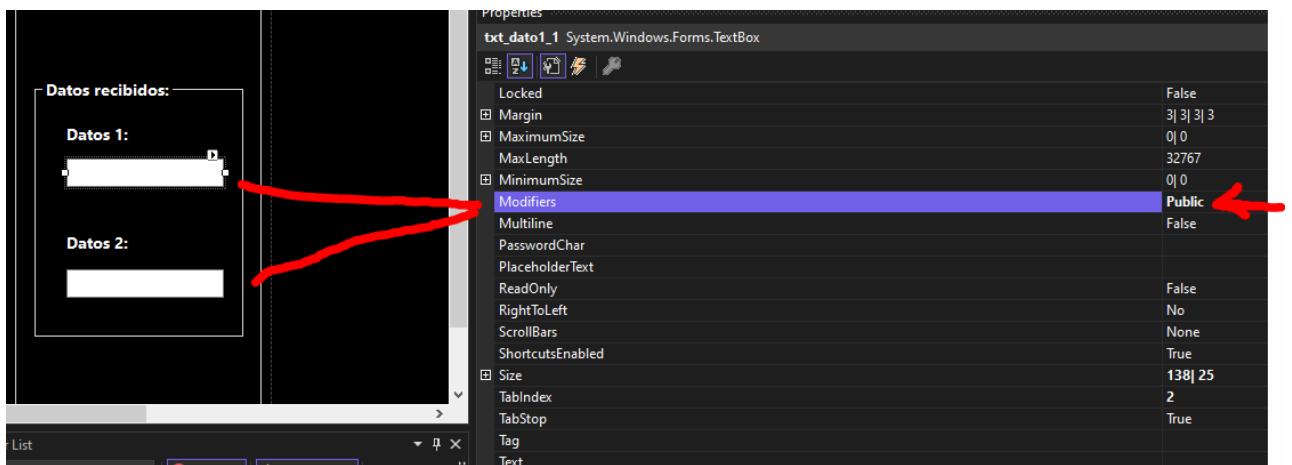
```

private void button2_Click(object sender, EventArgs e)
{
    // Cargamos las variables globales con el contenido de los text box
    Clase_variables.texto1 = txt_dato1_2.Text;
    Clase_variables.texto2 = txt_dato2_2.Text;

    // De las variables a los objetos del form principal
    frm_principal.txt_dato1_1.Text = Clase_variables.texto1;
    frm_principal.txt_dato2_1.Text = Clase_variables.texto2;
}

```

- e- Ver que los TextBox del formulario padre tengan esta propiedad en Public (esto permite que el contenido de los mismos sea público más allá de los objetos en si mismos):



- 7- Ahora vamos a ver usos del checkBox y el radioButton, para eso agregamos un nuevo form con las mismas modificaciones en las propiedades del que ya agregamos antes y procedemos a colocar el código en el botón que hará la carga del mismo dentro del Panel:

```

frm_principal.cs [Design] frm_segundo.cs [Design] frm_principal.cs ✎ X
        ↓ Panel_Get_Set.frm_principal
        ↓ btn_checkbox_Click

    InitializeComponent();
}

private void btn_getset_Click(object sender, EventArgs e)
{
    // Antes de cargar al panel un form me aseguro de que no tenga ninguno cargado
    if (panel.Controls.Count > 0)
        panel.Controls.RemoveAt(0);

    // Se procede a cargar al form dentro del panel y se setean algunas propiedades
    var form = Application.OpenForms.OfType<frm_segundo>().FirstOrDefault();
    frm_segundo segundoFrm = form ?? new frm_segundo();
    segundoFrm.TopLevel = false;
    segundoFrm.FormBorderStyle = FormBorderStyle.None;
    segundoFrm.Dock = DockStyle.Fill;
    panel.Controls.Add(segundoFrm);
    panel.Tag = segundoFrm;

    segundoFrm.Show();
}

private void btn_checkbox_Click(object sender, EventArgs e)
{
    // Antes de cargar al panel un form me aseguro de que no tenga ninguno cargado
    if (panel.Controls.Count > 0)
        panel.Controls.RemoveAt(0);

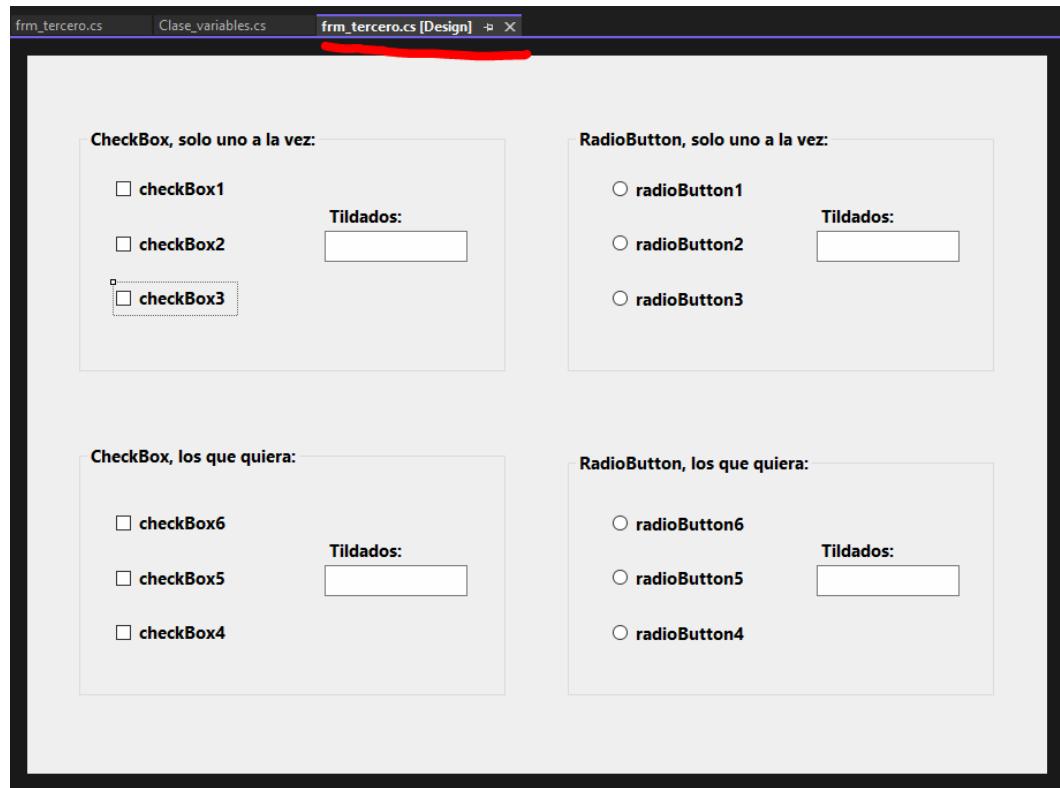
    // Se procede a cargar al form dentro del panel y se setean algunas propiedades
    var form = Application.OpenForms.OfType<frm_tercero>().FirstOrDefault();
    frm_tercero tercerFrm = form ?? new frm_tercero();
    tercerFrm.TopLevel = false;
    tercerFrm.FormBorderStyle = FormBorderStyle.None;
    tercerFrm.Dock = DockStyle.Fill;
    panel.Controls.Add(tercerFrm);
    panel.Tag = tercerFrm;

    tercerFrm.Show();
}

```

- 8- Aprovechando la clase que ya tenemos creada (Clase_variables) vamos a usarla para crear una función que nos ayude con los checkBox y los radioButtons.

Así es como quedó mi formulario:



Código para el caso “CheckBox, solo uno a la vez” en la clase:

```
=namespace Panel_Get_Set
{
    public class Clase_variables
    {
        public static string texto1 = "";
        public static string texto2 = "";

        public static string Funcion_check(int idgroup, int idChecked)
        {
            string resultado = "";

            switch(idgroup)
            {
                case 1:
                    if (idChecked == 1) resultado = "1;";
                    if (idChecked == 2) resultado = "2;";
                    if (idChecked == 3) resultado = "3;";
                    break;
                case 2:
                    break;
                case 3:
                    break;
                case 4:
                    break;
            }

            return (resultado);
        }
    }
}
```

Código para el caso “CheckBox, solo uno a la vez” en el formulario y dentro del evento checked de cada objeto:

```

private void click_checkBox1(object sender, EventArgs e)
{
    if (this.checkBox1.Checked)
    {
        checkBox2.Checked = false;
        checkBox3.Checked = false;
        textBox1.Text = Clase_variables.Fucion_check(1, 1);
    }
    else
        textBox1.Text = "";
}

private void click_checkBox2(object sender, EventArgs e)
{
    if (this.checkBox2.Checked)
    {
        checkBox1.Checked = false;
        checkBox3.Checked = false;
        textBox1.Text = Clase_variables.Fucion_check(1, 2);
    }
    else
        textBox1.Text = "";
}

private void click_checkBox3(object sender, EventArgs e)
{
    if (this.checkBox3.Checked)
    {
        checkBox1.Checked = false;
        checkBox2.Checked = false;
        textBox1.Text = Clase_variables.Fucion_check(1, 3);
    }
    else
        textBox1.Text = "";
}

```

Código para el caso “CheckBox, los que quiera” en la clase:

```

public static string Fucion_check(int idgroup, int idChecked)
{
    string resultado = "";

    switch(idgroup)
    {
        case 1:
            if (idChecked == 1) resultado = "1;";
            if (idChecked == 2) resultado = "2;";
            if (idChecked == 3) resultado = "3;";
            break;
        case 2:
            if (idChecked == 6) resultado = "6;";
            if (idChecked == 5) resultado = "5;";
            if (idChecked == 4) resultado = "4;";
            break;
        case 3:
            break;
        case 4:
            break;
    }

    return (resultado);
}

```

Código para el caso “CheckBox, los que quiera” en el formulario y dentro del evento checked de cada objeto:

```

private void click_checkBox6(object sender, EventArgs e)
{
    if (this.checkBox6.Checked)
        textBox2.Text += Clase_variables.Fucion_check(2, 6);
    else if (checkBox5.Checked && (checkBox4.Checked == false))
        textBox2.Text = Clase_variables.Fucion_check(2, 5);
    else if (checkBox4.Checked && (checkBox5.Checked == false))
        textBox2.Text = Clase_variables.Fucion_check(2, 4);
    else if (checkBox4.Checked && checkBox5.Checked)
    {
        textBox2.Text = Clase_variables.Fucion_check(2, 5);
        textBox2.Text += Clase_variables.Fucion_check(2, 4);
    }
    else
        textBox2.Text = "";
}

private void click_checkBox5(object sender, EventArgs e)
{
    if (this.checkBox5.Checked)
        textBox2.Text += Clase_variables.Fucion_check(2, 5);
    else if (checkBox6.Checked && (checkBox4.Checked == false))
        textBox2.Text = Clase_variables.Fucion_check(2, 6);
    else if (checkBox4.Checked && (checkBox6.Checked == false))
        textBox2.Text = Clase_variables.Fucion_check(2, 4);
    else if (checkBox4.Checked && checkBox6.Checked)
    {
        textBox2.Text = Clase_variables.Fucion_check(2, 6);
        textBox2.Text += Clase_variables.Fucion_check(2, 4);
    }
    else
        textBox2.Text = "";
}

private void click_checkBox4(object sender, EventArgs e)
{
    if (this.checkBox4.Checked)
        textBox2.Text += Clase_variables.Fucion_check(2, 4);
    else if (checkBox6.Checked && (checkBox5.Checked == false))
        textBox2.Text = Clase_variables.Fucion_check(2, 6);
    else if (checkBox5.Checked && (checkBox6.Checked == false))
        textBox2.Text = Clase_variables.Fucion_check(2, 5);
    else if (checkBox5.Checked && checkBox6.Checked)
    {
        textBox2.Text = Clase_variables.Fucion_check(2, 6);
        textBox2.Text += Clase_variables.Fucion_check(2, 5);
    }
    else
        textBox2.Text = "";
}

```

PD: Como podrán ver, se deben manejar los diferentes estados de cada objeto.

Código para el caso “RadioButton, solo uno a la vez” en la clase:

```

    if (idChecked == 0) resultado = "0";
    if (idChecked == 4) resultado = "4";
    break;
case 3:
    if (idChecked == 1) resultado = "1";
    if (idChecked == 2) resultado = "2";
    if (idChecked == 3) resultado = "3";
    break;
case 4:
    break;
}

```



Código para el caso “RadioButton, solo uno a la vez” en el formulario: los RadioButton tienen un comportamiento muy distinto a los CheckBox, por eso debemos manejar los mismos con dos eventos de forma paralela lo que no permitirá hace check y uncheck de los mismos:

```
bool rb1_isChecked = false;

private void check_RADIOBUTTON1(object sender, EventArgs e)
{
    rb1_isChecked = radioButton1.Checked;
}

private void click_RADIOBUTTON1(object sender, EventArgs e)
{
    if (radioButton1.Checked && !rb1_isChecked)
    {
        radioButton1.Checked = false;
        textBox3.Text = "";
    }
    else
    {
        radioButton1.Checked = true;
        rb1_isChecked = false;
        radioButton2.Checked = false;
        radioButton3.Checked = false;
        textBox3.Text = Clase_variables.Fucion_check(3, 1);
    }
}
```

```
bool rb2_isChecked = false;

private void check_RADIOBUTTON2(object sender, EventArgs e)
{
    rb2_isChecked = radioButton2.Checked;
}

private void click_RADIOBUTTON2(object sender, EventArgs e)
{
    if (radioButton2.Checked && !rb2_isChecked)
    {
        radioButton2.Checked = false;
        textBox3.Text = "";
    }
    else
    {
        radioButton2.Checked = true;
        rb2_isChecked = false;
        radioButton1.Checked = false;
        radioButton3.Checked = false;
        textBox3.Text = Clase_variables.Fucion_check(3, 2);
    }
}
```

```
bool rb3_isChecked = false;

private void check_RADIOBUTTON3(object sender, EventArgs e)
{
    rb3_isChecked = radioButton3.Checked;
}

private void click_RADIOBUTTON3(object sender, EventArgs e)
{
    if (radioButton3.Checked && !rb3_isChecked)
    {
        radioButton3.Checked = false;
        textBox3.Text = "";
    }
    else
    {
        radioButton3.Checked = true;
        rb3_isChecked = false;
        radioButton1.Checked = false;
        radioButton2.Checked = false;
        textBox3.Text = Clase_variables.Fucion_check(3, 3);
    }
}
```

Código para el caso “RadioButton, los que quiera” en la clase:

```

        if (idChecked == 3) resultado = "3;";
        break;
    case 4:
        if (idChecked == 6) resultado = "6;";
        if (idChecked == 5) resultado = "5;";
        if (idChecked == 4) resultado = "4;";
        break;
    }
}

```

Código para el caso “RadioButton, los que quiera” en el formulario: los RadioButton tienen un comportamiento tal que no permiten, estando dentro de un mismo grupo de objetos, estar tildado o checked más de uno al mismo tiempo con lo cual no podemos hacer ese ejemplo...



Les dejo el proyecto, realizado en VS2022:



15. EVENTOS VALIDATING Y VALIDATED

Cuando trabajas con formularios, es muy común que necesites **validar** los datos que el usuario ingresa en los controles (como cajas de texto, listas, etc.) antes de realizar alguna acción, como guardar o enviar los datos.

En **Windows Forms**, los eventos Validating y Validated te permiten gestionar y controlar la **validación** de los datos de manera sencilla.

Evento Validating:

El evento Validating se activa cuando el control (por ejemplo, una caja de texto) pierde el enfoque, es decir, cuando el usuario deja de interactuar con él y hace clic en otro control del formulario o presiona Tab para pasar al siguiente control.

Este evento es útil para **verificar si el valor ingresado es válido** antes de permitir que el usuario pase al siguiente control. Si el dato no es válido, puedes evitar que el control pierda el foco (usando el método Cancel), lo que obligaría al usuario a corregir el dato.

Ejemplo:

Imagina que tienes una caja de texto (TextBox) y quieres validar que el usuario ingrese un número válido:

```
private void textBox1_Validating(object sender, CancelEventArgs e)
{
    // Comprobar si el valor ingresado es un número
    if (!int.TryParse(textBox1.Text, out _))
    {
        MessageBox.Show("Por favor, ingrese un número válido.");
        e.Cancel = true; // Cancela el evento y mantiene el foco en el TextBox
    }
}
```

En este ejemplo, el evento Validating se activa cuando el usuario intenta salir del textBox1. Si el valor no es un número, se muestra un mensaje de advertencia y se evita que el control pierda el foco (e.Cancel = true).

Evento Validated:

El evento Validated es el opuesto del evento Validating. Se activa **después de que el control ha pasado la validación**. Es decir, si la validación en el evento Validating es exitosa (no se ha cancelado el evento), entonces el evento Validated se ejecutará.

Este evento no se usa para realizar la validación en sí, sino para ejecutar código que dependa de que el control haya sido validado correctamente.

Ejemplo:

Siguiendo el ejemplo anterior, puedes usar el evento Validated para realizar una acción después de que el dato en el control ha sido validado correctamente:

```
private void textBox1_Validate(object sender, EventArgs e)
{
    // Aquí puedes hacer algo cuando el dato sea válido
    MessageBox.Show("El número ingresado es válido.");
}
```

En este ejemplo, si el valor en el TextBox es un número válido, el evento Validated se dispara, y puedes realizar cualquier acción adicional (como mostrar un mensaje de éxito).

Flujo de los eventos:

1. El usuario ingresa datos en un control (por ejemplo, un TextBox).
2. El usuario intenta salir del control (presionando Tab, haciendo clic en otro control o perdiendo el foco de alguna otra manera).
3. Se dispara el evento **Validating**:
 - Si la validación falla (por ejemplo, el valor no es válido), puedes evitar que el control pierda el foco usando e.Cancel = true.
 - Si la validación es exitosa, el control pierde el foco.
4. Se dispara el evento **Validated**, solo si el control ha pasado la validación.

Resumiendo:

- **Validating:** Se usa para realizar la validación de datos cuando el control pierde el foco. Si el dato no es válido, puedes evitar que el control pierda el foco y pedir al usuario que lo corrija.
- **Validated:** Se usa para realizar acciones después de que el control haya sido validado correctamente.

Actividad sugerida:

Crear un formulario de registro de usuario que tenga los siguientes controles:

1. Un TextBox para ingresar un nombre de usuario.
2. Un TextBox para ingresar una edad.
3. Un botón para guardar.

Ayuda: Para el TextBox de la edad, por ejemplo, pueden usar el evento Validating para asegurarse de que el valor ingresado sea un número válido, y en el evento Validated pueden mostrar un mensaje indicando que el dato fue validado correctamente.

16. STRUCT Y RECORDS

El manejo de datos se puede tornar tedioso cuando los mismos son muchos (por ejemplo, cuando se interactúa con una base de datos o bien cuando se usa una serie de propiedades de un objeto o cosa). Para agilizar este tema existen las “estructuras” (struct) y los “registros” (record).

Con **las clases y las estructuras** tenemos el problema de que **pueden ser alterados**. Los objetos de tipo clase son tipos por referencia, mientras que las estructuras son tipos por valor, que lo más que se pueden acercar a un objeto inmutable es declarándolas como readonly.

Las struct no son clases, son estructuras. Una estructura es una agrupación lógica y tiene varias propiedades parecidas a una clase, pero no tiene las mismas características.

Los objetos de tipo record, **son objetos por referencia** que vienen a solucionar el problema existente a la hora de **generar objetos inmutables**, esto es, objetos que no pueden variar (lo que significa que sus propiedades no pueden cambiar una vez que se establecen).

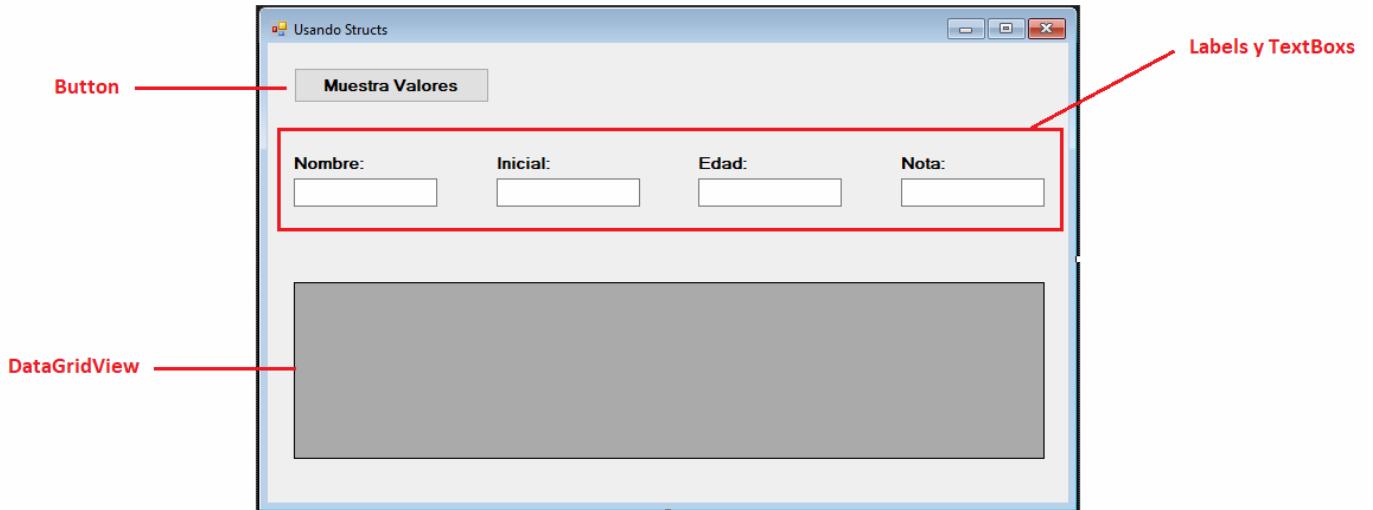
A tener en cuenta:

- Hasta C# 8, para almacenar un conjunto de datos podemos usar class o struct.
- En C# 9 y posteriores podemos usar record.
- Siendo:

Visual Studio 2008	C# 3.0
Visual Studio 2010	C# 4.0
Visual Studio 2012	C# 5.0
Visual Studio 2013	C# 5.0
Visual Studio 2015	C# 6.0
Visual Studio 2017	C# 7.0 a 7.3
Visual Studio 2019	C# 7.3 a 8.0
Visual Studio 2022	C# 9.0 a 11.0
Visual Studio 2024	+ Versiones Futuras

Ejemplo de uso usando Struct:

- **Diseño:** Algo simple, lo justo y necesario para este ejemplo



- **Eventos que se van a disparar y consecuencia de dichos eventos:**

La idea es que al hacer click en el Boton se carguen los textos con valores y se agreguen columnas y una fila a la Grilla con otro conjunto de valores.

En el evento Load del Formulario se crearán las columnas en la grilla.

- **Código:**

En el Load del Formulario:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Agrego columnas al DataGridView: primero el nombre de la columna y luego el caption o lo que el usuario ve
    dtg_datos.Columns.Add("nombre", "Nombre");
    dtg_datos.Columns.Add("inicial", "Inicial");
    dtg_datos.Columns.Add("edad", "Edad");
    dtg_datos.Columns.Add("nota", "Nota");
}
```

Declarando una estructura (struct), nótese que hay una parte que solo es usada si queremos asignar el conjunto de valores (llamado Caso2 dentro del código) sin tener que ir uno a uno (o sea, sin tener que hacer “campo = valor” uno por uno):

```
public Form1()
{
    InitializeComponent();
}

// Se declara una estructura (struct) llamada "tipoPersona" con una serie de campos
struct tipoPersona
{
    public string nombre;
    public char inicial;
    public int edad;
    public float nota;

    // Constructor que inicializa todas las propiedades, SOLO necesario para el Caso2
    // =====
    public tipoPersona(string Nombre, char Inicial, int Edad, float Nota)
    {
        nombre = Nombre;
        inicial = Inicial;
        edad = Edad;
        nota = Nota;
    }
    // =====
}

private void Form1_Load(object sender, EventArgs e)
{
    // Agrego columnas al DataGridView: primero el nombre de la columna y luego el caption
    dtg_datos.Columns.Add("nombre", "Nombre");
```

Nota:

Dentro del código, el Caso1 carga los valores en los TextBox, el Caso2 carga los valores en la grilla

Código en el evento Click del Botón:

```

// Evento click del botón:
private void btn_muestraValores_Click(object sender, EventArgs e)
{
    // Caso1: Valor por valor:
    // =====
    // Creo una instancia de la estructura que la llamo "persona" y luego asingo valores campo a campo
    tipoPersona persona;

    persona.nombre = "Juan";
    persona.inicial = 'J';
    persona.edad = 20;
    persona.nota = 7.5f;

    // Valores a los textBox;
    txt_nombre.Text = persona.nombre;
    txt_inicial.Text = persona.inicial.ToString();
    txt_edad.Text = persona.edad.ToString();
    txt_nota.Text = persona.nota.ToString();
    // =====

    // Caso2: Los valores en el orden según los campos
    // =====
    // Creo una instancia de la estructura que la llamo "persona2" y asingo valores en el orden correcto
    tipoPersona persona2 = new tipoPersona("pepe", 'p', 10, 2);

    // Cargo valores a la grilla:
    dtg_datos.Rows.Add(persona2.nombre, persona2.inicial, persona2.edad, persona2.nota);
    // =====
}

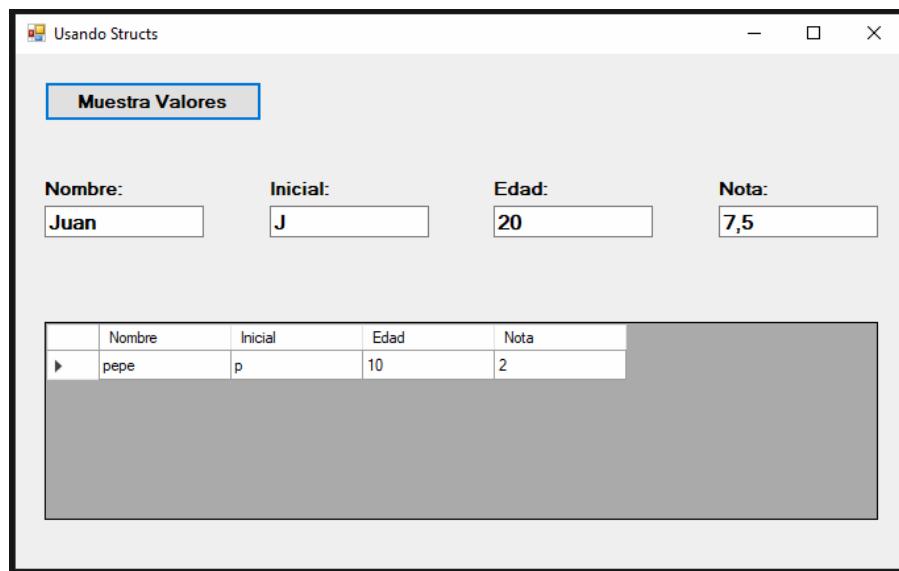
```

Notas:

El Caso1 primero instancia una estructura llamada “persona” que hereda las propiedades de la struct “tipoPersona”, se le asigna un valor a cada propiedad una a una y luego a los textBox en su propiedad Text se les asigna el contenido de cada propiedad de la struct “persona”.

El Caso2 instancia una estructura llamada “persona2” que hereda las propiedades de la struct “tipoPersona”, y al mismo momento se le pasan los valores en orden según el orden de los campos y propiedades de la struct padre.

- **Ejemplo de ejecución y luego de haber hecho Click en el Botón:**

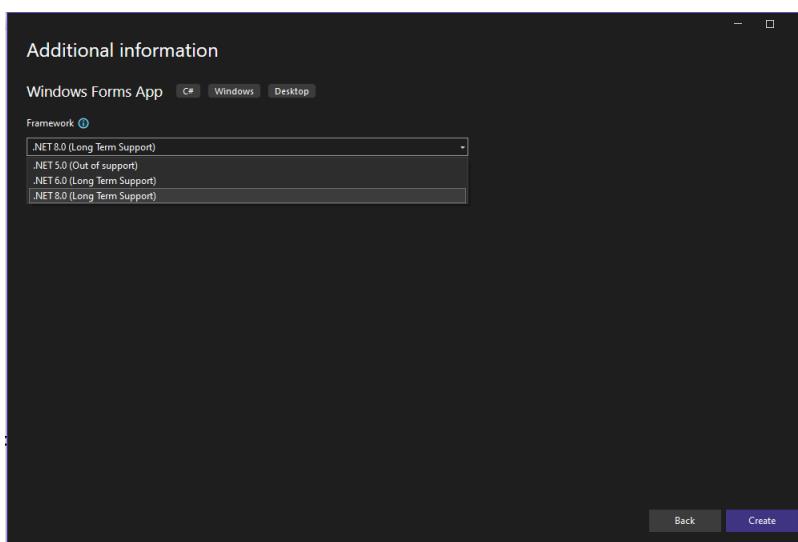
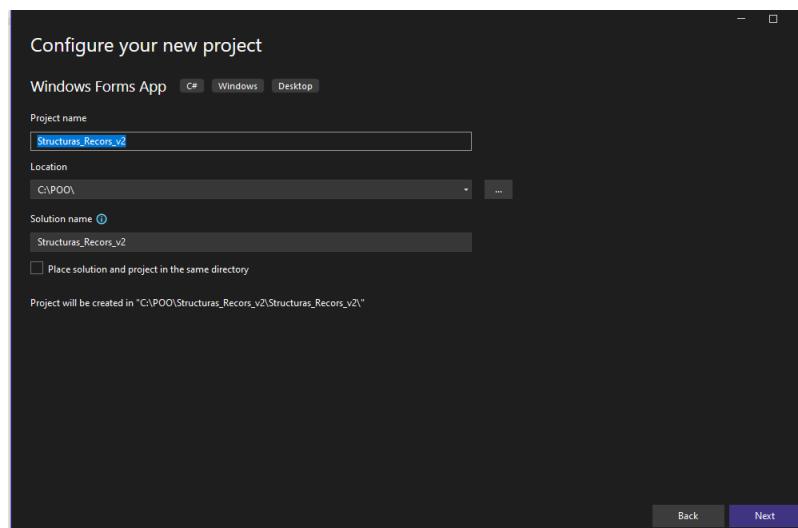
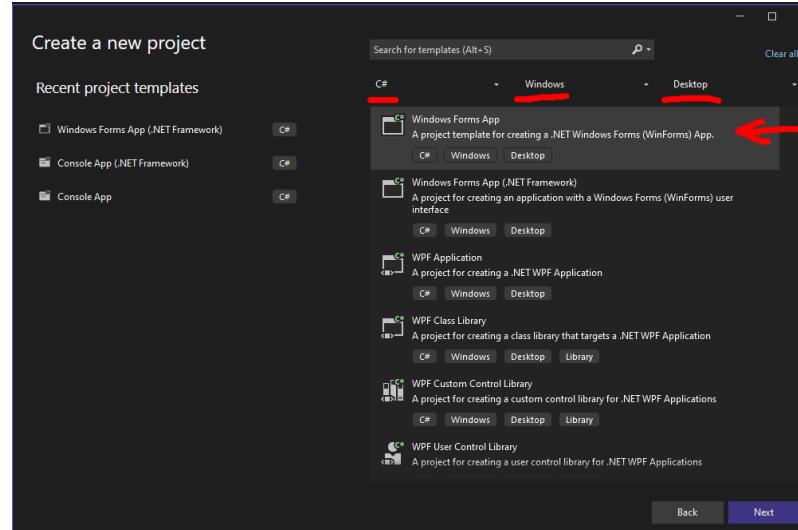


Ejemplo de uso usando Record:

- Para poder usar “record” debemos tener instalado Visual Studio 2022 y el .Net 5.0 como mínimo.

<https://dotnet.microsoft.com/en-us/download/dotnet/5.0>

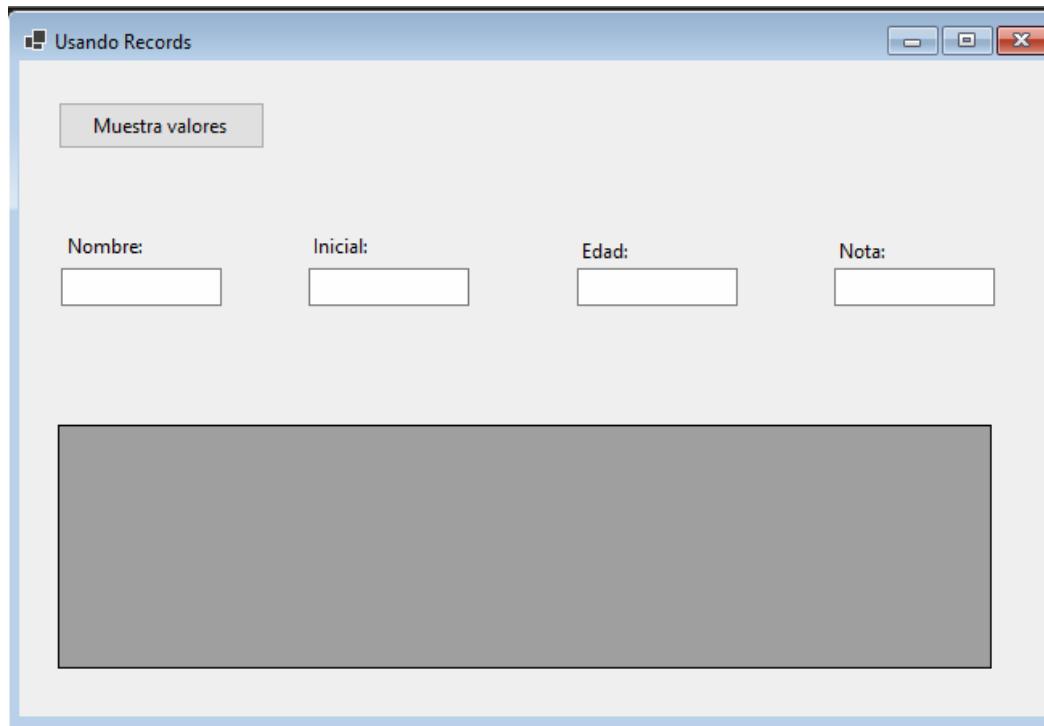
- Al crear un nuevo proyecto, para este caso se debe hacer de la siguiente manera:



Nota:

.NET Framework 5 no existe, la ultima versión es 4.8. .NET 5, 6, etc. son parte de la plataforma .NET moderna, mientras que .NET Framework 4.8 es una versión anterior y distinta.
 Para usar .NET 5 o superior en lugar de .NET Framework 4.8 en tu proyecto de Visual Studio 2022, tendrás que crear un nuevo proyecto de .NET 5 o superior.

- **Diseño: Mismo Diseño que el ejemplo anterior**



- **Eventos que se van a disparar y consecuencia de dichos eventos:**

La idea es que al hacer click en el Boton se carguen los textos con valores y se agreguen columnas y una fila a la Grilla con otro conjunto de valores.

En el evento Load del Formulario se crearán las columnas en la grilla.

- **Código:**

Van a notar que en este caso declara el record es mucho más simple que declarar una struct.

Siguiendo el mismo ejemplo que en el caso del struct (misma idea, misma lógica, mismo diseño) solo tenemos que reemplazar la declaración de la struct por la de un record.

La linea de código necesaria es la siguiente:

```
// Se declara un record llamado "TipoPersona" con una serie de propiedades
public record TipoPersona(string Nombre, char Inicial, int Edad, float Nota);
```

El código completo quedaría así:

```

namespace Structuras_Recors_v2
{
    public record Persona(string Nombre, char Inicial, int Edad, float Nota);

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

        // Se declara un record llamado "TipoPersona" con una serie de propiedades
        public record TipoPersona(string Nombre, char Inicial, int Edad, float Nota); ----->

        private void Form1_Load_1(object sender, EventArgs e)
        {
            // Agrego columnas al DataGridView: primero el nombre de la columna y luego el caption o lo que el usuario ve
            dtg_datos.Columns.Add("nombre", "Nombre");
            dtg_datos.Columns.Add("inicial", "Inicial");
            dtg_datos.Columns.Add("edad", "Edad");
            dtg_datos.Columns.Add("nota", "Nota");
        }

        // Evento click del boton:
        private void btn_muestraValores_Click_1(object sender, EventArgs e)
        {
            // Caso1: Valor por valor:
            // =====
            // Creo una instancia del record "persona" con los valores especificados
            var persona = new TipoPersona("Juan", 'J', 20, 7.5f);

            // Valores a los textBox;
            txt_nombre.Text = persona.Nombre;
            txt_inicial.Text = persona.Inicial.ToString();
            txt_edad.Text = persona.Edad.ToString();
            txt_nota.Text = persona.Nota.ToString();
            // =====

            // Caso2: Los valores en el orden según los campos
            // =====
            // Creo una instancia del record "persona2" con los valores especificados
            var persona2 = new TipoPersona("pepe", 'p', 10, 2);

            // Cargo valores a la grilla:
            dtg_datos.Rows.Add(persona2.Nombre, persona2.Inicial, persona2.Edad, persona2.Nota);
            // =====

            dtg_datos.ClearSelection();
        }
    }
}

```

Nota:

Como podrán ver, además ahora se usa una variable de tipo “var” que es como una estructura con valores variables que hereda la estructura del record al declararla e instanciarla a la cual le paso los valores:

```
// Creo una instancia del record "persona" con los valores especificados
var persona = new TipoPersona("Juan", 'J', 20, 7.5f);
```

17. ACCESO A BASE DE DATOS USANDO OLEDB

OLE DB (algunas veces escrito como **OLEDB** u **OLE-DB**) es la sigla de *Object Linking and Embedding for Databases* ("Enlace e incrustación de objetos para bases de datos") y es una tecnología desarrollada por Microsoft usada para tener acceso a diferentes fuentes de información, o bases de datos, de manera uniforme.

OLE DB permite separar los datos de la aplicación que los requiere. Esto se hizo así ya que diferentes aplicaciones requieren acceso a diferentes tipos y almacenes de datos, y no necesariamente desean conocer cómo tener acceso a cierta funcionalidad con métodos de tecnologías específicas. OLE DB está conceptualmente dividido en **consumidores y proveedores**; el consumidor es la aplicación que requiere acceso a los datos y el proveedor es el componente de software que expone una interfaz OLE DB a través del uso del Componente Object Model (COM).

OLE DB hace parte de los "Componentes de Microsoft para Acceso a Datos" o Microsoft Data Access Components (MDAC); MDAC es un grupo de tecnologías de Microsoft que interactúan en conjunto como una infraestructura que brinda a los programadores de la nueva era una forma para desarrollar aplicaciones con acceso a casi cualquier almacén de datos. Los proveedores OLE DB pueden ser creados para tener acceso a almacenes de datos que van desde simples archivos de texto y hojas de cálculo, hasta bases de datos complejas como Oracle, Microsoft SQL Server o Sybase ASE.

Como las diferentes fuentes de datos pueden tener diferentes capacidades, es posible que los proveedores OLE DB no implementen todas las interfaces posibles para OLE DB. Las capacidades disponibles son implementadas a través del uso de objetos COM - el proveedor OLE DB asocia la funcionalidad de una tecnología a una interfaz COM particular.

Microsoft califica la disponibilidad de una interfaz como "específica del proveedor", ya que puede no ser aplicable dependiendo de la tecnología de base de datos involucrada. Adicionalmente, los proveedores pueden aumentar las capacidades de una fuente de datos - capacidades conocidas como **servicios**, usando la jerga de Microsoft.

Ejemplo Acceso a Base de Datos Acces usando OLE DB (salida de datos por consola):

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Threading;

namespace LectorBD
{
    public class BaseDeDatos
    {
        private OleDbConnection ConexionConBD;
        private OleDbCommand Orden;
        private OleDbDataReader Lector;

        public void LeerDeBaseDeDatos()
        {
            // Crear la conexión con la base de datos
            string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=C:\\tfnos.mdb;";
            ConexionConBD = new OleDbConnection(strConexión);

            // Crear una consulta
            string Consulta = "SELECT nombre, telefono FROM telefonos";
            Orden = new OleDbCommand(Consulta, ConexionConBD);

            // Abrir la base de datos
            ConexionConBD.Open();
            // ExecuteReader hace la consulta y devuelve un OleDbDataReader
            Lector = Orden.ExecuteReader();
            // Llamar siempre a Read antes de acceder a los datos
            while (Lector.Read()) // siguiente registro
                Console.WriteLine(Lector["nombre"] + " " + Lector["telefono"]);

            // Llamar siempre a Close una vez finalizada la lectura
            Lector.Close();
        }

        public void CerrarConexion()
        {
            // Cerrar la conexión cuando ya no sea necesaria
            if (Lector != null)
                Lector.Close();
            if (ConexionConBD != null)
                ConexionConBD.Close();
        }

        public static void Main()
        {
            BaseDeDatos bd = new BaseDeDatos();
            try
            {
                bd.LeerDeBaseDeDatos();
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
            }
            finally
            {
                bd.CerrarConexion();
            }
            Thread.Sleep(3000);
        }
    }
}
```

18. Ejemplo simple de conexión a un archivo de DB Access

Para el ejemplo utilizaremos el siguiente diseño:



Paso1: Conectarse a la DB

La conexión a la DB se efectuará solo cuando se haga click en el botón que dice "Conecta DB". Para ello veremos a continuación que es lo que se necesita a nivel código para poder realizar dicha acción.

Para poder usar las clases y objetos necesarios para conectarse a una DB debemos agregar una nueva librería o colección de clases y es la siguiente:

```
using System.Data.OleDb;
```

Que significa: Object Linking and Embedding for Databases ("Enlace e incrustación de objetos para bases de datos") y es una tecnología desarrollada por Microsoft usada para tener acceso a diferentes fuentes de información, o bases de datos, de manera uniforme.

Ahora es el momento de declarar las variables u objetos de conexión que me permitirán interactuar con la DB ya sea para conectarme, para generar una orden (de lectura o escritura) y para leer o escribir en ella:

```
private OleDbConnection ConexionConBD;
private OleDbCommand Orden;
private OleDbDataReader Lector;
```

Que significan:

OleDbConnection: es para poder establecer la conexión con la DB.

OleDbCommand: es la orden o comando a ejecutar sobre la DB.

OleDbDataReader: es quien recibirá la respuesta de la orden ejecutada.

Ahora el código que va dentro del botón:

```
private void btn_conecta_Click(object sender, EventArgs e)
{
    string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=C:\\\\Ejemplo_Conexión_DB\\\\MiDB.mdb;";
    ConexionConBD = new OleDbConnection(strConexión);
    try
    {
        ConexionConBD.Open();
        MessageBox.Show("Base de datos abierta!!!");
    }
    catch (Exception)
    {
        MessageBox.Show("Error al abrir DB.");
    }
}
```

Veamos para que sirve cada cosa:

La variable “strConexión”: esta variable (que luego servirá de dato para un puntero) contiene todos los datos de conexión a la DB, contiene cual será el motor de DB a utilizar, cual es la ruta del archivo de DB y cuál es el nombre completo de dicho archivo.

El objeto “ConexionConBD”: este objeto es ahora un puntero que me servirá de nexo entre mi aplicación y la DB en sí.

“ConexionConBD.Open();”: aquí es donde abre la conexión hacia la DB con todos los datos que se le pasaron como parámetros.

Paso2: Ejecutar una consulta para Leer la DB

Es el momento de hacer la consulta sobre la DB, traer los datos resultantes y enviárselos a la Grilla para que el usuario los pueda ver.

Para ello vamos a precisar algunas cosas como por ejemplo:

- Una variable que contenga la consulta SQL a realizar sobre la DB.
- Instanciación del objeto “Orden”.
- Instanciación del objeto “Lector”.
- Uso de un objeto del tipo DataTable (“Tabla de Datos”).

El orden en que se deben hacer las cosas es el siguiente:

- Declaro la variable de la consulta y le asigno el string que representa una consulta en SQL a ejecutar sobre la DB.
- Instancio el objeto “Orden” pasándole como parámetros la consulta y la conexión.
- Instancio el objeto “Lector” con la “Orden” y la acción para dicha orden.
- Creo un objeto del tipo DataTable que recibirá los datos del “Lector” respetando la estructura de los datos.
- Al DataSource (el origen o proveedor de los datos) de la grilla le asigno el contenido del objeto DataTable. De esa forma la grilla hereda el formato y los datos resultantes de la consulta realizada.
- Cierro el Lector.

Ahora el código que va dentro del botón:

```
private void btn_carga_Click(object sender, EventArgs e)
{
    string consulta = "SELECT * FROM Mi_Tabla;";
    Orden = new OleDbCommand(consulta, ConexionConBD);
    Lector = Orden.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(Lector);
    grilla.DataSource = dt;
    grilla.ClearSelection();
    Lector.Close();
}
```

Como verán no es necesario darle formato la grilla en modo diseño ya que la misma puede heredar el formato de los datos resultantes de una consulta sobre una DB. Si no quieren heredar formato y lo quieren dar ustedes se debería hacer algo así en este botón (claro que deberían previamente agregar las columnas en el diseño de la grilla):

```
string consulta = "SELECT * FROM Mi_Tabla;";
Orden = new OleDbCommand(consulta, ConexionConBD);
Lector = Orden.ExecuteReader();
while (Lector.Read())
{
    grilla.Rows.Add();
    grilla[0, grilla.Rows.Count - 1].Value = Lector["Nombre"];
    grilla[1, grilla.Rows.Count - 1].Value = Lector["Apellido"];
    grilla[2, grilla.Rows.Count - 1].Value = Lector["DNI"];
}
grilla.ClearSelection();
Lector.Close();
```

Como podrán ver, para la carga de una grilla no es muy cómodo realizarlo de esta forma ya que se debe indicar que campo va en que columna y demás. Este método si es usado para cargar un ComboBox.

Paso3: Desconectarse de la DB

En este botón solo tenemos que desconectarnos de la DB, pero es necesario hacer algunos controles para evitar errores de cierre de dicha DB.

Los controles a realizar deberían ser:

- Controlar que el objeto “Lector” ya esté cerrado.
- Ver si alguna vez se abrió la DB para luego cerrarla.
- Informar al usuario.

Ahora el código que va dentro del botón:

```
private void btn_desc_Click(object sender, EventArgs e)
{
    if (Lector != null)
        Lector.Close();

    if (ConexionConBD != null)
    {
        try
        {
            ConexionConBD.Close();
            MessageBox.Show("Base de datos cerrada!!!");
        }
        catch (Exception)
        {
            MessageBox.Show("Error al cerrar DB.");
        }
    }

    if (ConexionConBD == null)
    {
        MessageBox.Show("No hay ninguna base abierta!!!");
    }
}
```

Possible falla dependiendo la máquina en donde estén programando:

Muchas veces, dependiendo de la arquitectura del procesador y del SO que tengan instalado o bien de la versión de Office que tengan en sus equipos suele generarse una falla que dice algo como:

“El proveedor 'Microsoft.Jet.OLEDB.4.0' no está registrado en el equipo local”

Eso puede ser tanto porque están en Windows de 64 bits como porque tiene instalado Office 2010. Como solución a ese problema hay dos opciones, una más simple que la otra pero ambas tienen sus ventajas y/o desventajas:

La forma más simple es cambiar el motor de conexión a la DB de Access:

En lugar de poner: Provider=Microsoft.Jet.OLEDB.4.0;

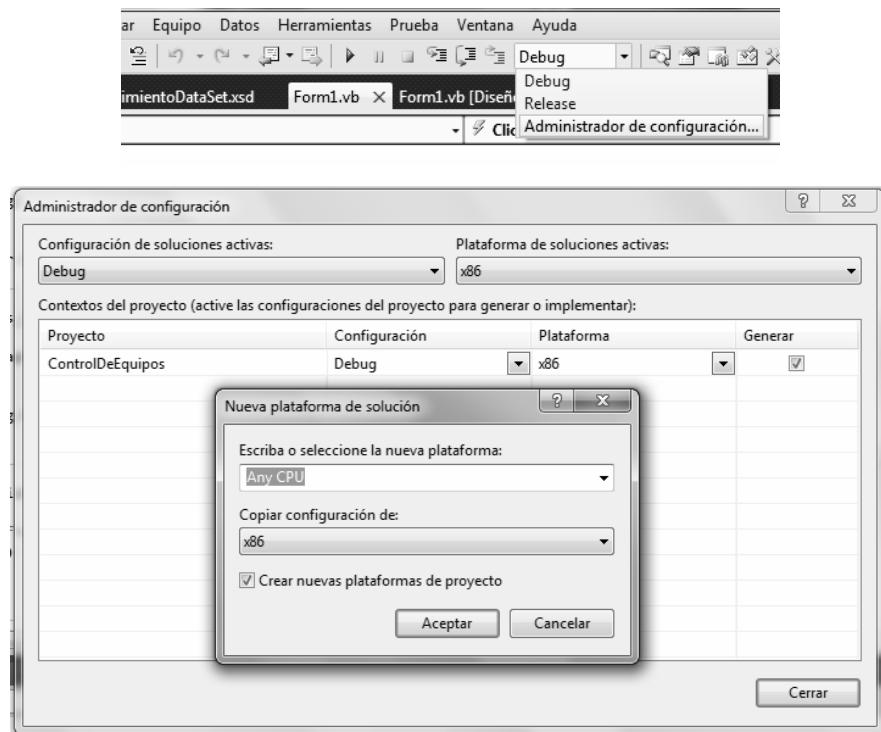
Deberán poner: Provider=Microsoft.ACE.OLEDB.12.0;

La forma compleja es cambiando unas propiedades del proyecto pero esto limita a que el mismo se pueda ejecutar tanto en x86 como en x64.

De todos modos esta es la segunda solución: <http://support.microsoft.com/kb/942977/es>

O bien, pueden seguir estos pasos si es que no tienen acceso a internet:

Donde aparece la lista de bug o reléase, a un lado del ícono de ejecución de la aplicación, selecciona la pestaña y elijen "Administrador de configuración" allí les aparecerá algo así, y deben darle nueva plataforma y seleccionar ya sea x86 o any cpu:



Si no tienen eso en la barra de herramientas, entonces sigan estos pasos:

Hay que ir a: Herramientas > Opciones > Proyectos y soluciones > General y seleccionar Mostrar configuraciones de compilación avanzadas.

Además también hay que seleccionar Mostrar todas las configuraciones.

A continuación hacer click en Generar > Administrador de configuración y donde dice Plataforma de soluciones activas darle a Nueva y en Seleccionar nueva plataforma elegir x86.

Después de todo eso, compila el proyecto y no da ningún error.

19. CONECTANDO A UNA DB SQL

- 9- Mi Base de Datos está así:

The screenshot shows the 'Database Properties - DemoDB' window in SQL Server Management Studio. The left pane lists database properties: General, Files, Filegroups, Options, Change Tracking, Permissions, Extended Properties, Mirroring, Transaction Log Shipping, and Query Store. The right pane shows the 'General' tab with the following details:

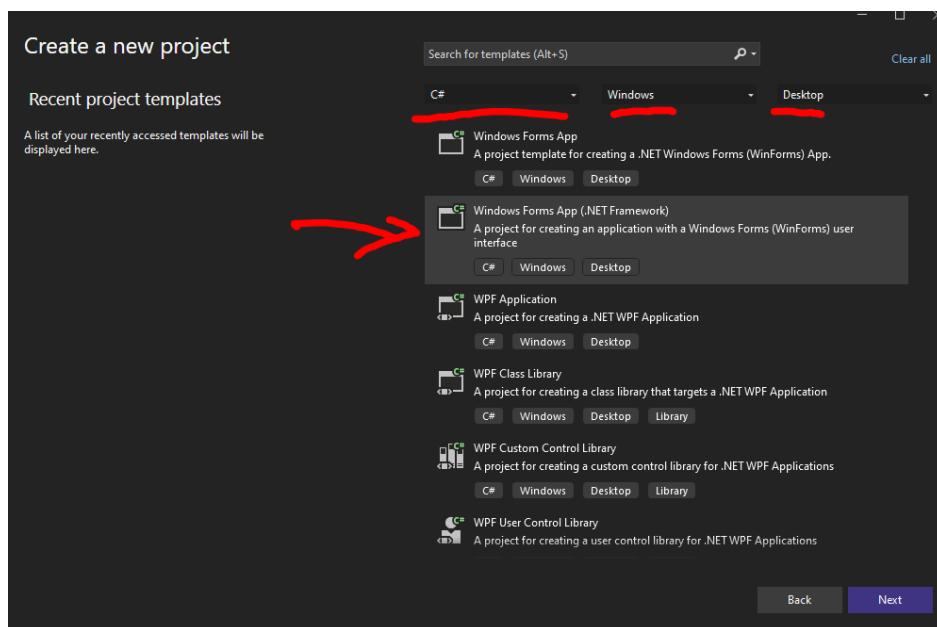
	Value
Name	DemoDB
Status	Normal
Owner	pinil
Date Created	21/5/2024 14:39:21
Size	16,00 MB
Space Available	4,22 MB
Number of Users	4
Memory Allocated To Memory Optimized Objects	0,00 MB
Memory Used By Memory Optimized Objects	0,00 MB

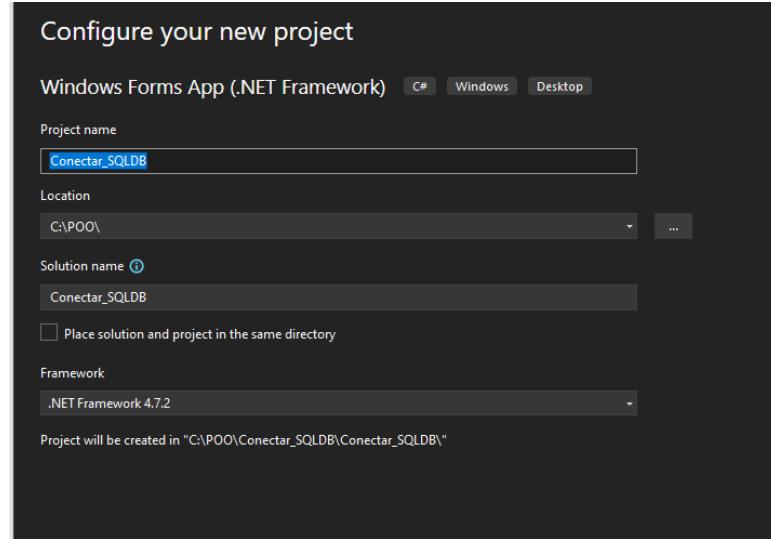
SQLSERVERLAP => nombre de la instancia SQL (como es local, delante usaré **localhost** para establecer la conexión)

DmoDB => nombre que le puse a la Base de Datos

pinil => es el Owner o dueño de la DB pero como es el mismo usuario de Windows usará la autenticación del Sistema Operativo

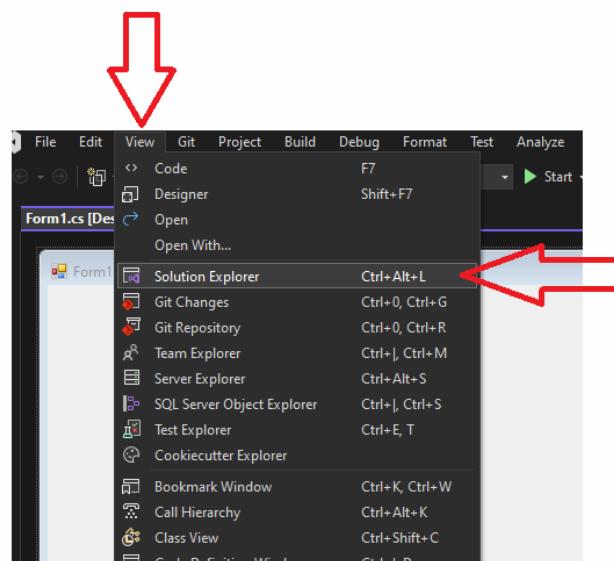
- 10- Crear un nuevo proyecto de Windows forms (en mi caso usaré de ejemplo VS2022 pero es lo mismo hacerlo en el VS2010):



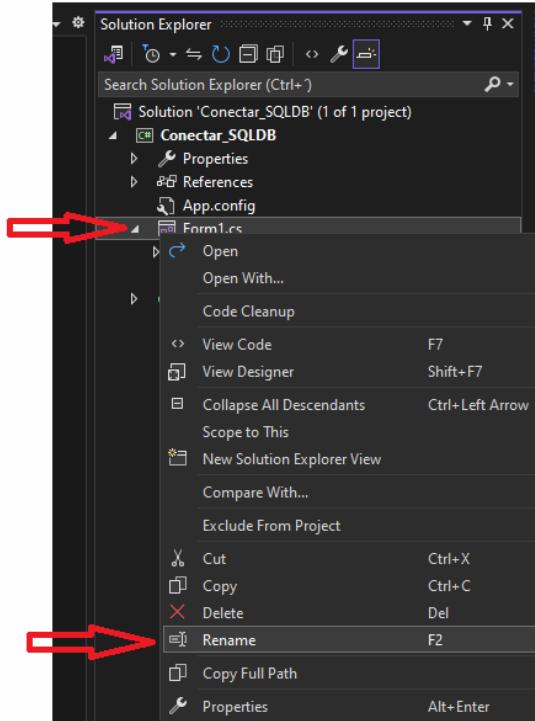


11- Antes de agregar objetos al formulario, vamos a cambiarle el nombre al mismo:

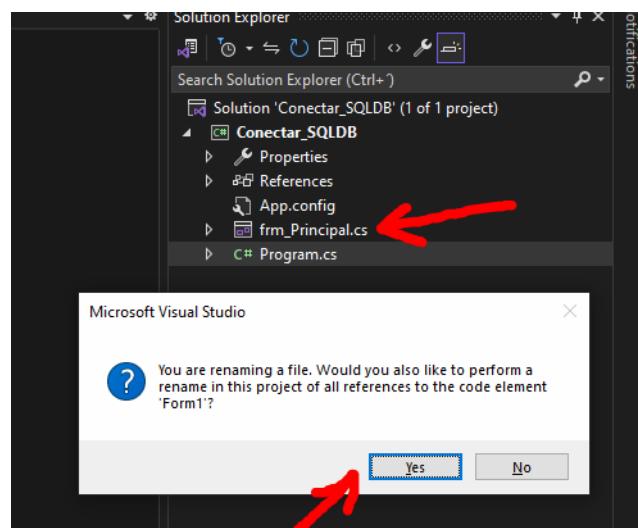
Para ello ojo lo que hacen!! Deben visualizar todos los componentes de la solución usando esta opción del menú:



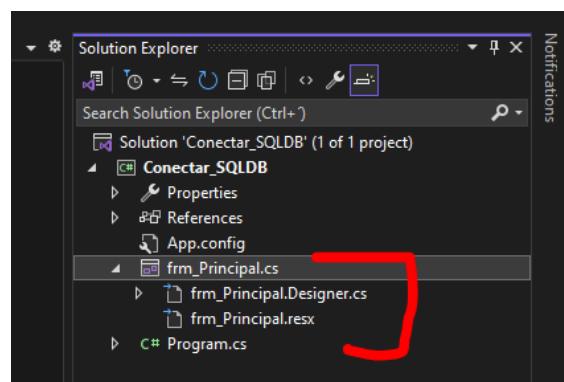
Luego, en la vista que les abre de los componentes de la solución, deberán situar el ícono del mouse sobre el nombre del formulario, hacer click con el botón derecho del mismo y de las opciones que se despliegan seleccionar “Rename” o “Renombrar” o “Cambiar de Nombre”:



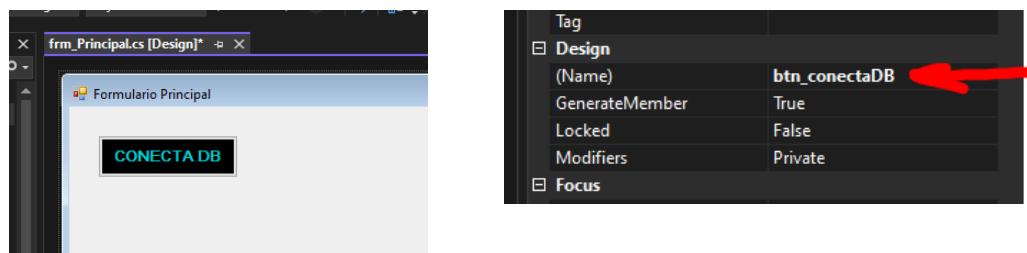
Y le colocan el nombre que quieran, yo le puse "frm_Principal" y ponen que SI a la pregunta:



Y les quedará así:



- 12- Agregarán un Botón que servirá en este caso para establecer la conexión a la Base de Datos (no olviden darle un nombre luego de colocarlo en el formulario y antes de ingresar a codificar):



- 13- Dentro del evento click del Botón, van a colocar el siguiente código:

```

C# Conectar_SQLDB
Conectar_SQLDB.frm_Principal
btn_conectaDB_Click(object sender, EventArgs e)
  
```

```

1  using System;
2  using System.Data.SqlClient;
3  using System.Windows.Forms;
4
5  namespace Conectar_SQLDB
6  {
7      public partial class frm_Principal : Form
8      {
9          public frm_Principal()
10         {
11             InitializeComponent();
12         }
13
14         private void btn_conectaDB_Click(object sender, EventArgs e)
15         {
16             string connetionString;
17             SqlConnection cnn;
18             connetionString = @"Data Source=localhost\SQLSERVERLAP;Initial Catalog=DemoDB;Trusted_Connection=True";
19             cnn = new SqlConnection(connetionString);
20             cnn.Open();
21             MessageBox.Show("Connection Open !");
22             cnn.Close();
23         }
24     }
25 }
26
  
```

En donde:

Data Source=localhost\SQLSERVERLAP => máquina local + nombre del servicio SQL

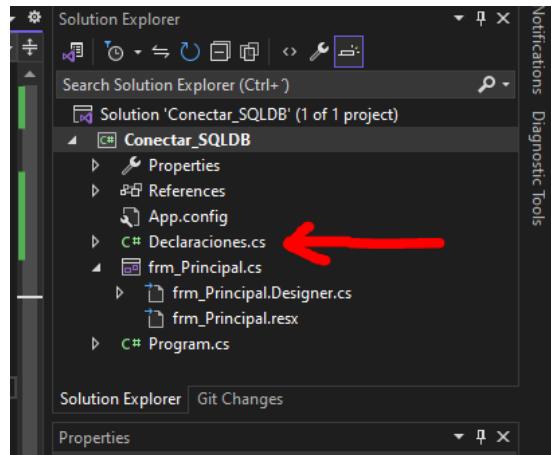
Initial Catalog=DemoDB => nombre de la DB

Trsuted_Connection=True => si la conexión a la DB es con Windows Authentication

Si no es con Windows Authentication usar: User ID=sa;Password=demol23 (colocando el user y la pass correspondiente).

- 14- Una vez que veamos el mensaje de OK quiere decir que la conexión a la DB fue exitosa. Entonces vamos a separar un poco ese código:

- a- Primero vamos a colocar las declaraciones de manera global para que puedan ser usadas desde cualquier parte, para ello agregar una clase pública:



Donde vamos a dejar el siguiente código:

```

using System;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace Conectar_SQLDB
{
    public static class Declaraciones
    {
        public static string connectionString;
        public static SqlConnection cnn;
    }
}

```

b- En el formulario principal vamos a hacer lo siguiente:

En el Botón de conectar dejaremos este código:

```

private void btn_conectaDB_Click(object sender, EventArgs e)
{
    Declaraciones.connectionString = @"Data Source=localhost\SQLSERVERLAP;Initial Catalog=DemoDB;Trusted_Connection=True;";
    Declaraciones.cnn = new SqlConnection(Declaraciones.connectionString);
    Declaraciones.cnn.Open();
    MessageBox.Show("Connection Open !");
}

```

Agregaremos un segundo Botón para desconectar y dejaremos este código:

```

private void btn_desconectaDB_Click(object sender, EventArgs e)
{
    Declaraciones.cnn.Close();
    MessageBox.Show("Connection Closed !");
}

```

Ojo cuando usan las variables y punteros, referenciar nombre de clase + nombre de objeto.

c- Al momento todo el código del formulario principal es este:

```

using System;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace Conectar_SQLDB
{
    public partial class frm_Principal : Form
    {
        public frm_Principal()
        {
            InitializeComponent();
        }

        private void btn_conectaDB_Click(object sender, EventArgs e)
        {
            Declaraciones.connectionString = @"Data Source=localhost\SQLSERVERLAP;Initial Catalog=DemoDB;Trusted_Connection=True;";
            Declaraciones.cn = new SqlConnection(Declaraciones.connectionString);
            Declaraciones.cn.Open();
            MessageBox.Show("Connection Open !");
        }

        private void btn_desconectaDB_Click(object sender, EventArgs e)
        {
            Declaraciones.cn.Close();
            MessageBox.Show("Connection Closed !");
        }
    }
}

```

Y su diseño solo esto:



15- Ahora vamos a cargar el contenido de una tabla dentro de una Grilla (DataGridView):

- a- Agregar un DataGridView al formulario
- b- Agregar un nuevo Botón que ejecutará la carga del DataGridView



- c- En el Botón el siguiente código:

```

private void btn_cargaGrilla_Click(object sender, EventArgs e)
{
    DataTable tabla = new DataTable();
    string consulta = "SELECT * from Personas";
    SqlCommand comando = new SqlCommand(consulta, Declaraciones.cn);
    SqlDataReader reader = comando.ExecuteReader();
    tabla.Load(reader);
    dtg_datos.DataSource = tabla;
}

```

PD: está claro que la tabla Personas debe existir en la DB y tener campos y registros con datos.

=====

Pre requisitos para que esto funcione:

Deben tener instalado el SQLServer y con el Management Studio.

Yo use estos:



Y validar que tengan estos servicios en Ejecución:

Spot Verifier	Verifies potential file system corruptions.	Manual (Trigger St
SQL Server (SQLSERVERLAP)	Provides storage, processing and controlled access...	Manual
SQL Server Agent (SQLSERVERLAP)	Executes jobs, monitors SQL Server, fires alerts, and...	Manual
SQL Server Browser	Provides SQL Server connection information to clie...	Disabled
SQL Server CEIP service (SQLSERVERLAP)	CEIP service for Sql server	Manual
SQL Server VSS Writer	Provides the interface to backup/restore Microsoft ...	Automatic

20. ANEXO 4 - Carga de un combobox usando DataSource

```
try
{
    DataSet dsCombo = new DataSet();

    //SQL.ComboBoxData es una funcion que le paso el string y me debuelve un DataSet segun el
    //string que le pase.
    dsCombo = SQL.ComboBoxData("Select * From Clientes");

    //Carga del combo con todos los registros que devuelve la consulta
    cboCliente.DataSource = dsCombo;

    // De los campos que trae la consulta le indico cual va a ser visible al usuario
    cboCliente.DisplayMember = "Clientes.Descripcion";

    // De los campos que trae la consulta le indico cual es el indice de acceso a los mismos
    cboCliente.ValueMember = "Clientes.IDCliente";

}

catch (Exception ex)
{
    // En caso de error muestro un mensaje del error en si.
    MessageBox.Show(ex.Message);
}
```

21. ANEXO 5 - Carga de una grilla (DataGridView) con una consulta a una DB

```

private void cargarGoleadores()
{
    // Declaro en un string la consulta a ejecutar sobre la DB
    string consulta = "Select InsNombre,GolNombre,GolGoles from Goleadores where TorId=" + TorId
                      + " order by GolGoles DESC";

    // Declaro un objeto del tipo que permita almacenar los datos de la DB y la consulta.
    OleDbDataReader Datos = bd.consulta(conn, consulta);

    // Declaro un objeto de tipo tabla de datos.
    DataTable dt = new DataTable();

    // Cargo el objeto tabla con los datos resultantes de la consulta.
    dt.Load(Datos);

    // Le indico a la grilla quien le provee los datos y los cargo.
    dgvGoleadores.DataSource = dt;

    // Ordeno los datos cargados en la grilla por la columna 3 de la misma.
    dgvGoleadores.Sort(dgvGoleadores.Columns[3]);
    // Indico en orden ascendente la muestra de datos en la grilla.
    ListSortDirection.Descending();

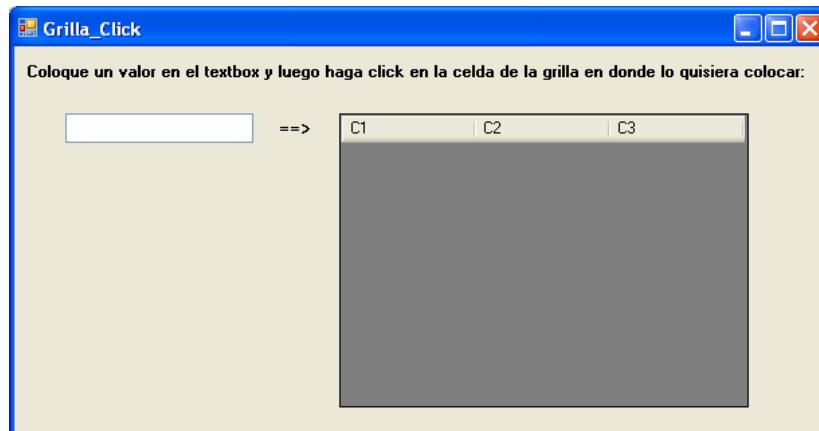
    // En base al orden dado por las sentencias anteriores le asigno un número a cada linea de
    // la tabla como indicando posiciones en este caso tabla de posiciones según cantidad de
    // goles.
    for (int i = 0; i < dgvGoleadores.Rows.Count ;i++)
    {
        int j = i + 1;
        dgvGoleadores.Rows[i].Cells["colPosG"].Value = j.ToString();
    }
}

```

22. ANEXO 6 - Jugando con el DataGridView

Cargando un valor en una celda con un solo click:

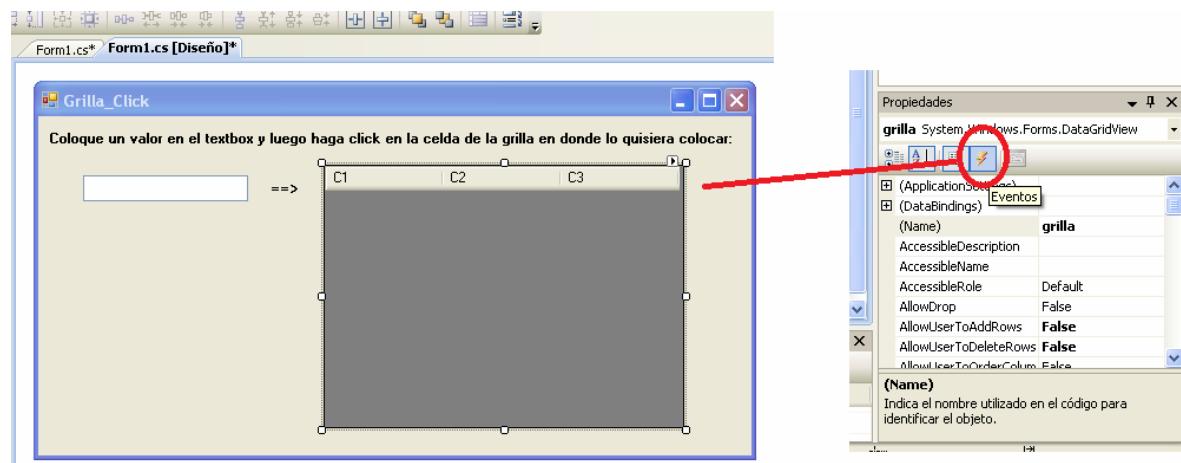
Vamos a utilizar el siguiente diseño para el ejemplo:



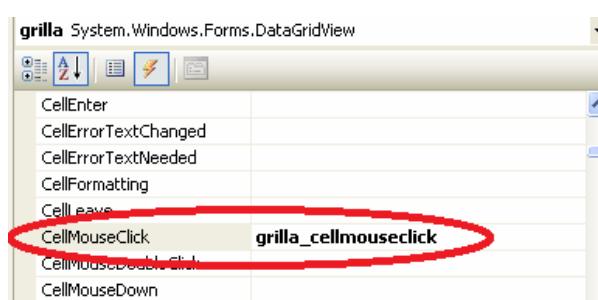
La idea es, tal como dice en el label agregado, que cuando se haga click en una celda de la grilla se guarde en dicha celda el contenido del textbox.

Para ello todo lo que necesitamos es programar dentro del evento que corresponda, que en este caso por tratarse del evento click sobre la grilla utilizando el mouse debemos habilitar este evento de forma tal que luego lo podamos usar.

Para habilitarlo, tenemos que dejar seleccionada la grilla y en la ventana de propiedades debemos hacer click en el ícono con un rayo tal como se ve en la siguiente imagen:



Eso nos mostrará la lista de posibles eventos que se pueden programar para este objeto grilla, vamos a utilizar uno de ellos que para poder usarlo se debe activar y para ello simplemente le debemos dar un nombre. El evento a utilizar se llama "CellMouseClick" y le vamos a dar un nombre:



Verán que luego de colocar el nombre y presionar la tecla “enter” nos llevará directo al código de ese evento. Es ahí en donde debemos comenzar a programar.

Primero agregaremos unas filas a la grilla y que las mismas aparezcan cuando se inicia el programa, por eso el código va dentro del load del formulario:

```
private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 1; i <= 9; i++)
        grilla.Rows.Add();
}
```

Ahora sí, pasaremos a codificar/programar las acciones que dispararán nuestro evento habilitado:

```
private void grilla_CellMouseClick(object sender, DataGridViewCellEventArgs e)
{
    if (Convert.ToString(grilla[e.ColumnIndex, e.RowIndex].Value) == "")
    {
        grilla[e.ColumnIndex, e.RowIndex].Value = textBox1.Text;
        grilla.ClearSelection();
    }
    else
    {
        MessageBox.Show("Celda ya ocupada.");
    }
}
```

Explicación de las líneas de código:

Debemos preguntar o saber si en la celda donde se hizo click ya hay un valor ya que la idea no es reemplazar el contenido sino completar una celda vacía. El if utilizado contiene el código necesario para ello en donde se utiliza la variable “e” que contiene toda la información que nos da el evento disparado sobre la grilla:

```
if (Convert.ToString(grilla[e.ColumnIndex, e.RowIndex].Value) == "")
```

Si la respuesta a esa pregunta es verdadera entonces procede a colocar el valor en la celda y luego limpia la selección para que dicha celda no quede seleccionada y pintada de color azul:

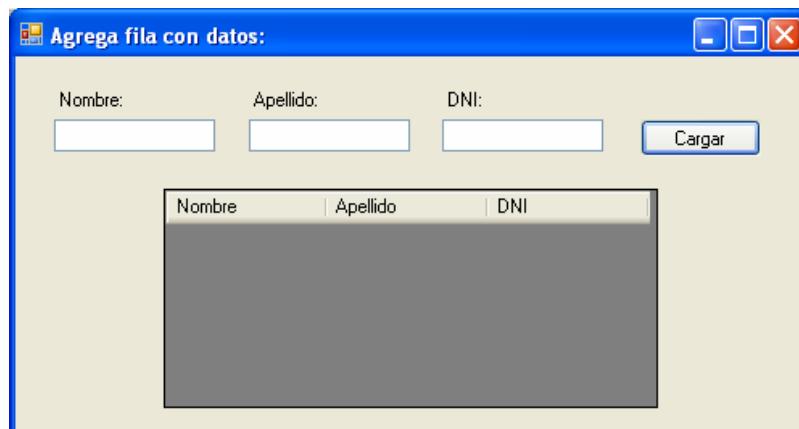
```
grilla[e.ColumnIndex, e.RowIndex].Value = textBox1.Text;
grilla.ClearSelection();
```

En caso de que no sea verdadera la respuesta enviaremos un mensaje al usuario que indique que esa celda ya está ocupada:

```
MessageBox.Show("Celda ya ocupada.");
```

Agregando una fila con valores en cada celda de la misma:

Vamos a utilizar el siguiente diseño para el ejemplo:



La idea es que cuando se coloquen los valores en los textbox y se haga click en el botón se inserte una nueva fila en la grilla y que en cada columna contenga el valor que corresponde según indica su nombre y en referencia a cada textbox.

Como todas las acciones se realizaran cuando se haga un click en el botón todo el código deberá estar dentro de ese evento de ese objeto.

```
private void btn_cargar_Click(object sender, EventArgs e)
{
    string nom = txt_nom.Text;
    nom = nom.Replace(" ", "");
    string ape = txt_ape.Text;
    ape = ape.Replace(" ", "");
    string dni = txt_dni.Text;
    dni = dni.Replace(" ", "");

    if ((nom == "") || (ape == "") || (dni == ""))
    {
        MessageBox.Show("Debe ingresar los 3 datos para poder insertar la fila.");
    }
    else
    {
        grilla.Rows.Add();
        int fila = grilla.Rows.Count - 1;
        grilla[0, fila].Value = txt_nom.Text;
        grilla[1, fila].Value = txt_ape.Text;
        grilla[2, fila].Value = txt_dni.Text;
        grilla.ClearSelection();
    }
}
```

Lo primero que debemos asegurar y validar es que el usuario ingrese los tres valores y que los mismos no sean solo una cadena de espacios en blanco, para ello utilizamos lo siguiente:

Declaramos una variable de tipo string a la cual le movemos/asignamos el contenido del textbox, luego a esa variable le aplicamos el método “Replace” que sirve para reemplazar un carácter por otro dentro de dicha cadena, en este caso le decimos que reemplace un espacio por un vacío:

```
string nom = txt_nom.Text;
nom = nom.Replace(" ", "");
```

También lo podemos hacer en una sola línea de código:

```
string nom = txt_nom.Text.Replace(" ", "");
```

También podemos no declarar una nueva variable y evaluarlo al mismo tiempo que reemplazando:

```
if (txt_nom.Text.Replace(" ", "") == "")
```

Luego preguntamos si el contenido de alguna de esas variables es vacío y si es así enviamos un mensaje al usuario, caso contrario (que el contenido de ninguna de esas variables sea vacío) procedemos a asignar el valor a cada columna de la nueva fila que se inserta:

```
if ((nom == "") || (ape == "") || (dni == ""))
{
    MessageBox.Show("Debe ingresar los 3 datos para poder insertar la fila.");
}
else
{
    grilla.Rows.Add();
    int fila = grilla.Rows.Count - 1;
    grilla[0, fila].Value = txt_nom.Text;
    grilla[1, fila].Value = txt_ape.Text;
    grilla[2, fila].Value = txt_dni.Text;
    grilla.ClearSelection();
}
```

23. APlicando FILTROS SOBRE UN DATAGRIDVIEW

Supongamos que se necesita filtrar los registros de un DataGridView por contener un valor determinado en una de sus columnas. El usuario puede introducir una subcadena (substring o parte de la cadena) en un TextBox, y que la grilla se vaya filtrando a medida que el usuario va escribiendo cada nuevo carácter.

Esta funcionalidad se puede implementar de forma muy sencilla. En muchos casos la grilla está vinculada a un origen de datos (como un DataSource de tipo DataTable, lo llamado ADO.NET: *conjunto de clases que exponen servicios de acceso a datos para programadores de .NET Framework*), y podemos explotar la funcionalidad de filtrado incorporada a través de la propiedad DefaultView.RowFilter de un DataTable para filtrar la grilla.

Pasos a seguir para lograr eso (el proyecto de base usado para este paso a paso es el “*Conectarse a una DB de SQL Server desde un proyecto en Windows Form*” entregado y desarrollado en la clase del 22 de Mayo):

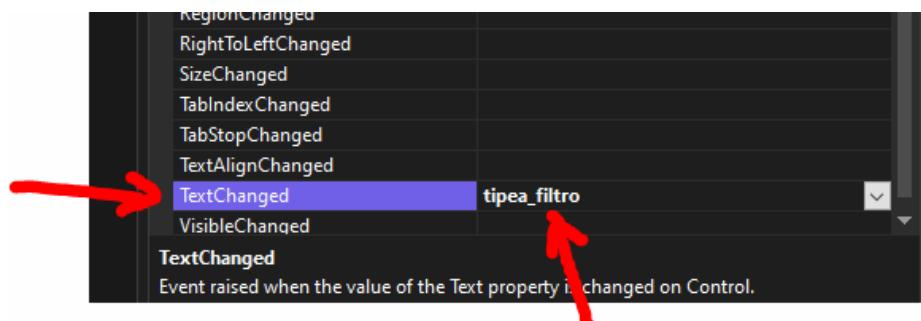
- 1- Cambio de lugar de algunas declaraciones para que los objetos/variables/estructuras sean de uso público:

<p>En el formulario principal, y dentro del botón de carga de grilla teníamos esto:</p>	<pre>private void btn_cargaGrilla_Click(object sender, EventArgs e) { DataTable tabla = new DataTable(); string consulta = "SELECT * from Personas"; SqlCommand comando = new SqlCommand(consulta, Declaraciones.cn); SqlDataReader reader = comando.ExecuteReader(); tabla.Load(reader); dtg_datos.DataSource = tabla; }</pre>
<p>La idea es mover esa declaración a la clase de Declaraciones para usar luego el objeto desde donde se lo necesite:</p>	<pre>public static class Declaraciones { public static string connectionString; public static SqlConnection cnn; public static DataTable tabla = new DataTable(); ↑ }</pre>
<p>Y ahora cambiamos en el evento click del botón mencionado:</p>	<pre>private void btn_cargaGrilla_Click(object sender, EventArgs e) { string consulta = "SELECT * from Personas"; SqlCommand comando = new SqlCommand(consulta, Declaraciones.cn); SqlDataReader reader = comando.ExecuteReader(); Declaraciones.tabla.Load(reader); dtg_datos.DataSource = Declaraciones.tabla; }</pre>

- 2- Agregaremos un Label a modo etiqueta de ayuda visual y un TextBox que lo llamaré txt_filtro:



- 3- En las propiedades del TextBox y particularmente en su lista de posibles eventos buscaremos el siguiente (y le colocaremos ese nombre, luego presionamos Enter y nos llevará directo al código de dicho evento):



- 4- Dentro del código del evento recién nombrado:

```
private void tipea_filtro(object sender, EventArgs e)
{
    Declaraciones.tabla.DefaultView.RowFilter = string.Format("[{0}] LIKE '%{1}%', "Apellido", txt_filtro.Text);
    dtg_datos.ClearSelection();
}
```

En donde:

Declaraciones.tabla.DefaultView.RowFilter

Declaraciones: clase usada para declarar variables, objetos, etc.

tabla: uno de los objetos declarados

DefaultView: si vista completa de datos/campos/registros.

RowFilter: método que permite aplicar un filtro de un campo de los registros.

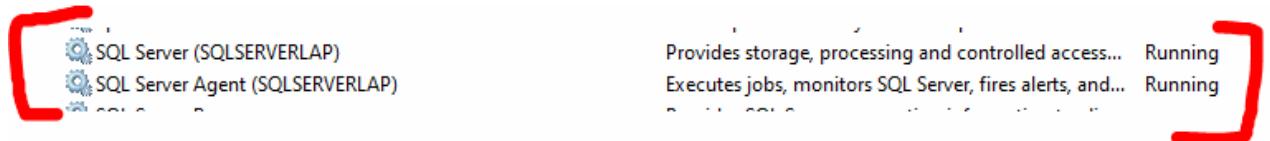
```
string.Format("{0} LIKE '{1}%', "Apellido", txt_filtro.Text);
```

txt_filtro.Text: de donde tomará el valor (string) para usar como filtro.

"Apellido": la columna/campo del objeto Tabla (nombre del campo como viene de resultado de la consulta SQL).

"[{0}] LIKE '%{1}%'" : es un string armado por partes en donde una parte es el campo o valor {0} y el otro es el valor {1} (el {0} es el "Apellido" y el {1} es el contenido del textbox).

- 5- No olvidar tener los servicios de DB en Ejecución:



- 6- Prueba funcional:

Conectamos a DB, Cargamos la grilla con los datos de la consulta a la misma:



Tipeamos en el TextBox por lo que queramos filtrar:



- 7- Posible falla en o con el filtro dependiendo del tipo de dato:

Tener en cuenta que el nombre del campo en la expresión del filtro debe ir entre corchetes para evitar problemas si el nombre del mismo contiene espacios (algo que no debería pasar en una DB pero si se renombra el campo para mostrarlo con un nombre distinto y agradable al usuario...).

Además hay un pequeño problema: el operador LIKE sólo funciona para valores de cadena, pero no para valores de tipo enteros o fecha. La propiedad RowFilter soporta la sintaxis de la propiedad System.Data.DataColumn.Expression que nos provee la función **Convert**. Podemos utilizarla para convertir valores de campo a cadenas sin analizar el tipo de columna, por lo que la sentencia de asignación a la propiedad RowFilter debería quedar así:

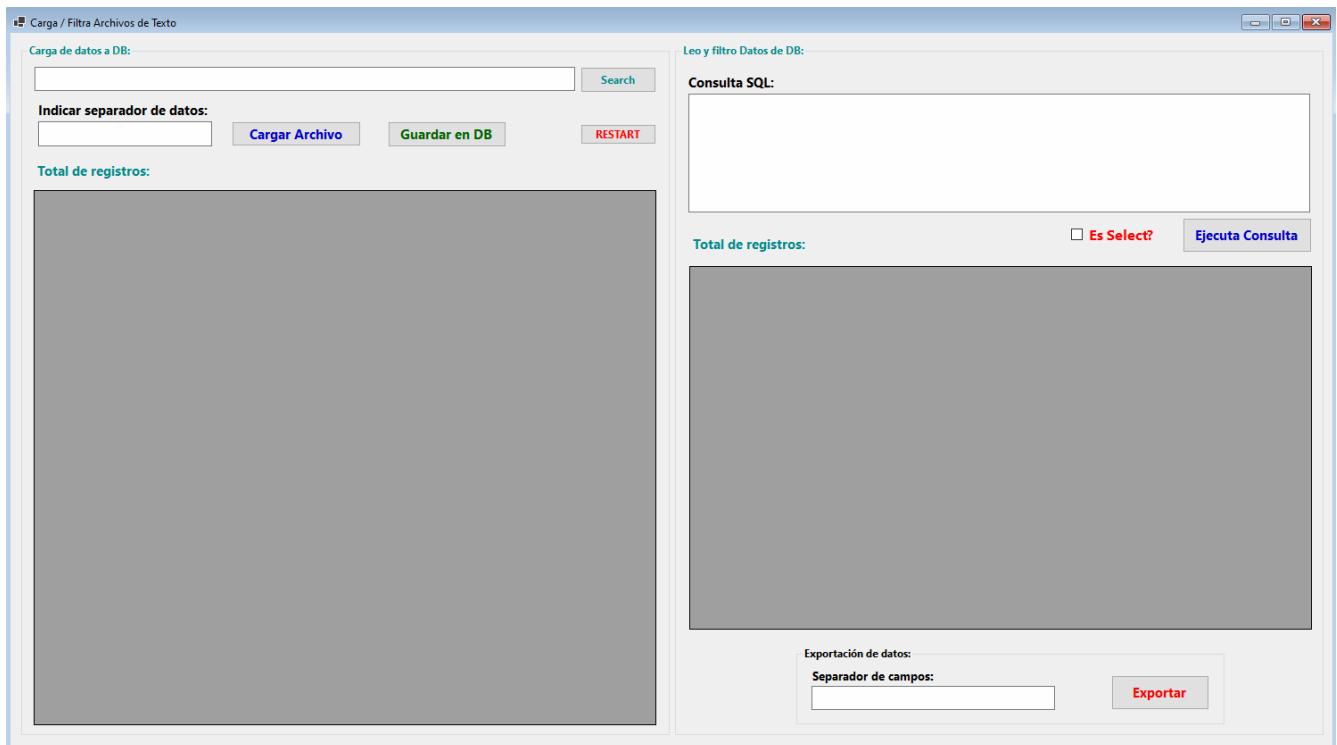
```
private void tipea_filtro(object sender, EventArgs e)
{
    Declaraciones.tabla.DefaultView.RowFilter = string.Format("Convert([{0}], 'System.String') LIKE '{1}%', "Apellido", txt_filtro.Text);
    dtg_datos.ClearSelection();
}
```

Con aumento para quienes andan cortos de vista (y por la cantidad de comillas y demás que tiene):

```
= string.Format("Convert([{0}], 'System.String') LIKE '{1}%', "Apellido", txt_filtro.Text);
```

24. CARGA TXT A DB SQL Y APLICANDO FILTROS

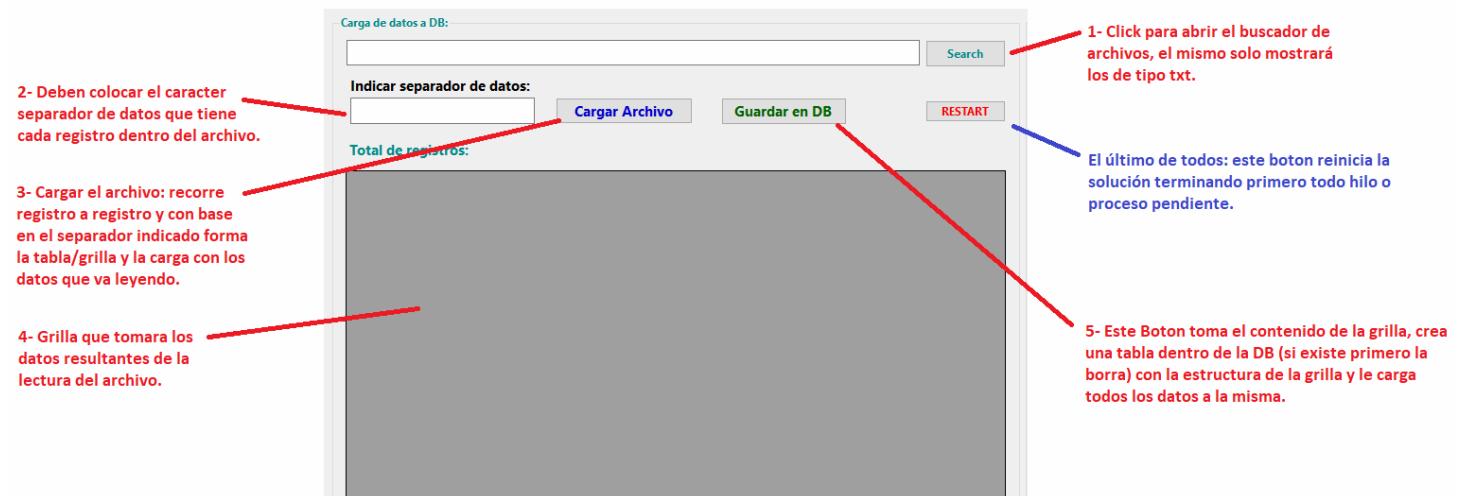
Como podrán ver el diseño incluye un conjunto de objetos que ya conocen (labels, botones, grillas, etc). El tema es entender como funciona y que hace:



Es como tener dos programas en uno, del lado de la izquierda sería el Programa A (de ahora en más llamaremos PA) y del lado derecho el Programa B (de ahora en más lo llamaremos PB).

Vamos a comenzar con el PA:

Esta parte del programa toma un archivo de texto plano, lo muestra en una grilla y permite cargar el contenido de la misma a una tabla dentro de una DB (en este caso DB de SQL Server):



Código del PA (cada punto hará referencia al número en la imagen anterior):**1- El buscador de archivos:**

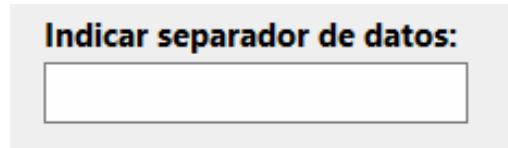
En el evento click del botón, invocaremos al openFileDialog (propio del Sistema Operativo) y le aplicaremos algunos filtros como por ejemplo en que ruta por defecto abrir o mostrar y que tipo de archivos queremos ver, una vez que se seleccione al archivo colocará la ruta completa y el nombre del mismo dentro del txt_file:

```
private void btn_serchfile_Click(object sender, EventArgs e)
{
    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.InitialDirectory = "C:\\\\";
        openFileDialog.Filter = "txt files (*.txt)|*.txt";
        openFileDialog.FilterIndex = 2;
        openFileDialog.RestoreDirectory = true;

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            txt_file.Text = openFileDialog.FileName;
        }
    }
}
```

2- El separador de importación de datos:

Es solo un textBox en el cual debemos escribir el separador de campos. Ejemplo un “;”. No tiene/posee código alguno:

**3- Carga del archivo a la grilla:**

Una parte compleja tal vez, porque debe limpiar la grilla + tomar el file y recorrerlo registro a registro + identificar los campos + separarlos + dimensionar la tabla/grilla + cargar la grilla + etc.

La siguiente imagen contiene todo el código dentro del evento click del botón y está explicado cada parte del mismo (puntos extras para el profe Pini que se toma ese trabajo... ¿no?):

```

private void btn_loadfile_Click(object sender, EventArgs e)
{
    while (dtg_viewfile.Rows.Count > 0)
    {
        dtg_viewfile.Rows.RemoveAt(0);
    }

    while (dtg_viewfile.Columns.Count > 0)
    {
        dtg_viewfile.Columns.RemoveAt(0);
    }

    dtg_viewfile.Refresh();
    dtg_viewfile.DataSource = null;

    if (txt_file.Text.Trim() != "")
    {
        string[] columnas;
        StreamReader L = new StreamReader(txt_file.Text);
        string s = L.ReadLine();
        if (s != null)
        {
            if (txt_separador.Text.Length > 0)
                columnas = s.Split(txt_separador.Text);
            else
                columnas = new string[1];

            for (int i = 0; i < columnas.Count(); i++)
                dtg_viewfile.Columns.Add(i.ToString(), ("Data" + i));
        }

        for ( ; ; )
        {
            if (s == null) break;

            if (txt_separador.Text.Length > 0)
            {
                columnas = s.Split(txt_separador.Text);
                dtg_viewfile.Rows.Add(columnas);
            }
            else
                dtg_viewfile.Rows.Add(s);

            s = L.ReadLine();
        }

        L.Close();
        dtg_viewfile.ClearSelection();

        lbl_totreg.Text = dtg_viewfile.Rows.Count.ToString();
    }
}

```

Si el archivo está en blanco no hace nada.

Lee una linea

Si no está o es nula hace...

Con la longitud del array crea las columnas en la grilla.

Habilita botón

Cierra puntero

Limpia la grilla. En VS2010 no hace nada.

Luego de limpiar refresca la misma y pone en null su origen de datos.

Declara un array que usará de contenedor de datos registro a registro.

Puntero al archivo.

Si se indicó un separador hace el split de la linea leída usando el mismo y con eso dimenciona e instancia el array. Sinó, lo dimensiona de una sola posición.

El for, cuyo desde y hasta es definido por el principio y fin del archivo, recorre registro a registro y por cada uno inserta una nueva fila en la grilla con los datos resultantes de cada split usando el separador indicado.

Limpia celda seleccionada.

Muestra total de registros de la grilla.

4- La grilla que tendrá el contenido resultante del punto 3. No posee código propio.

5- El botón que carga la grilla a la DB:

```

private void btn_cargaDB_Click(object sender, EventArgs e)
{
    Declaraciones.conexionDb.Open();

    string createQry = "", dropQry = "DROP TABLE Datos;", insertQry = "INSERT INTO Datos VALUES(";

    // Drop table...
    try
    {
        Declaraciones.comando = new SqlCommand(dropQry, Declaraciones.conexionDb);
        Declaraciones.comando.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        MessageBox.Show("DROP Table no hizo nada. La tabla ya no existe en la DB, se procede a creación y carga.");
    }

    // Create table...
    createQry = Funciones.armaQry(dtg_viewfile.ColumnCount);
    Declaraciones.comando = new SqlCommand(createQry, Declaraciones.conexionDb);
    Declaraciones.comando.ExecuteNonQuery();

    // Insert...
    for(int f = 0; f < dtg_viewfile.Rows.Count; f++)
    {
        for (int c = 0; c < dtg_viewfile.ColumnCount; c++)
        {
            if (c < (dtg_viewfile.ColumnCount - 1))
                insertQry = insertQry + "" + dtg_viewfile[c, f].Value + "" + ", ";
            else
                insertQry = insertQry + "" + dtg_viewfile[c, f].Value + "" + ");";
        }

        Declaraciones.comando = new SqlCommand(insertQry, Declaraciones.conexionDb);
        Declaraciones.comando.ExecuteNonQuery();
        insertQry = "INSERT INTO Datos VALUES(";

        Declaraciones.conexionDb.Close();
        MessageBox.Show("Fin Carga a DB.");
    }

    btn_cargaDB.Enabled = false;
}

```

Abre la DB.

Declaraciones.conexionDb.Open();

Declaraciones.comando = new SqlCommand(dropQry, Declaraciones.conexionDb);
Declaraciones.comando.ExecuteNonQuery();

Si la tabla existe la borra.

Crea de nuevo la tabla (usa una función que luego mostraré el código).

Recorre cada fila de la grilla y toma el dato de cada columna armando el Qry de INSERT. Una vez armada la ejecuta.

Cierra la conexión a la DB.
Mensaje al usuario de OK.
Deshabilita el botón de carga.

6- El último de todos:

Primera línea: cierra todo hilo pendiente o en ejecución de la aplicación.

Segunda línea: reinicia la aplicación.

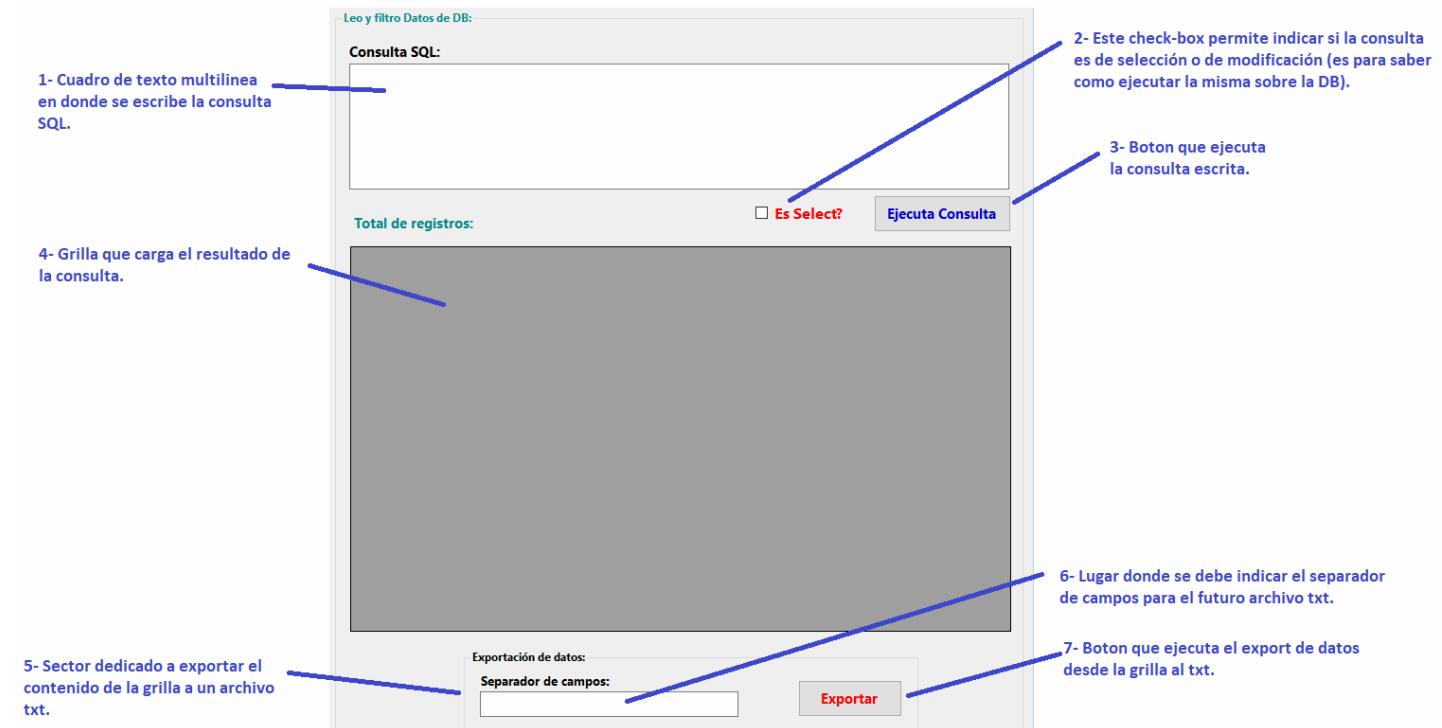
```
private void button1_Click(object sender, EventArgs e)
{
    Application.ExitThread();
    Application.Restart();
}
```

===== FIN LADO O PARTE PA =====

El lado PB:

Esta parte del programa permite:

- 1- Escribir y ejecutar consultas SQL sobre la DB que se tiene abierta (o mejor dicho que usa).
- 2- El resultado de la consulta es mostrado en la grilla.
- 3- Permite además bajar el contenido de la grilla a un nuevo archivo txt.



Código del PB (cada punto hará referencia al número en la imagen anterior):

- 1- El TextBox multilínea: Sin código propio, solo Multilínea = True en las propiedades del mismo.
- 2- El CheckBox: no tiene código propio, solo se toma su valor Checked desde otro método.

3- Botón que ejecuta el SQL escrito:

```

private void button3_Click(object sender, EventArgs e)
{
    Declaraciones.conexionDb.Open(); Abre la conexión a la DB.

    while (dtg_datosDB.Rows.Count > 0)
    {
        dtg_datosDB.Rows.RemoveAt(0);
    }

    while (dtg_datosDB.Columns.Count > 0)
    {
        dtg_datosDB.Columns.RemoveAt(0);
    }

    dtg_datosDB.Refresh();
    dtg_datosDB.DataSource = null; Elimina las filas y columnas que tenga la grilla.

    if (txt_consulta.Text.Trim() != "")
        Declaraciones.consulta = txt_consulta.Text;
    else
        Declaraciones.consulta = "select * from Datos"; Refresh a la grilla y quita el DataSource que tenga.

    int tipo = 0;
    if(check_select.Checked == true)
        tipo = 1; Si se escribió algo lo toma como consulta escrita sino por defecto pone un "Select * from <tabla>"

    dtg_datosDB.DataSource = Funciones.cargaTabla(Declaraciones.consulta, tipo); Usando el checkbox define que tipo de consulta es.
    dtg_datosDB.ClearSelection();

    Declaraciones.conexionDb.Close(); Carga la grilla declarando como DataSource el resultado o bien lo que devuelve la función que ejecuta la consulta en la DB.
    lbl_totregfinal.Text = dtg_datosDB.Rows.Count.ToString(); Luego limpia la celda que queda seleccionada.

}

```

Cierra la conexión a la DB.
Muestra total de registros en la grilla.

- 4- La grilla que carga el resultado: no posee código propio, solo se estructura y carga desde el punto anterior.
- 5- Sector dedicado al export de datos.
- 6- TextBox en donde se debe colocar un separador de datos para la exportación de los mismos. No posee código propio.
- 7- Botón que ejecuta el export de datos:

Este botón:

- Toma el separador ingresado en el textbox.
- Toma la grilla
- Leyendo registro a registro de la grilla arma un string concatenando las columnas y colocando en medio el separador indicado.
- Ese string resultante lo escribe en el nuevo archivo de texto.

```

private void btn_exporta_Click(object sender, EventArgs e)
{
    if (dtg_datosDB.Rows.Count > 0)
    {
        StreamWriter W = new StreamWriter("C:\\P00\\Filtira_Registros\\ExportData.txt");

        string reg = "";
        for (int f = 0; f < dtg_datosDB.Rows.Count; f++)
        {
            for (int c = 0; c < dtg_datosDB.ColumnCount; c++)
            {
                if (c < (dtg_datosDB.ColumnCount - 1))
                    reg = reg + dtg_datosDB[c, f].Value + text_sepExport.Text;
                else
                    reg = reg + dtg_datosDB[c, f].Value;
            }
            W.WriteLine(reg);
            reg = "";
        }

        W.Close();
        MessageBox.Show("Exportado a: C:\\P00\\Filtira_Registros\\ExportData.txt");
    }
}

```

Solamente hace si hay registros o filas en la grilla.

Puntero de escritura.

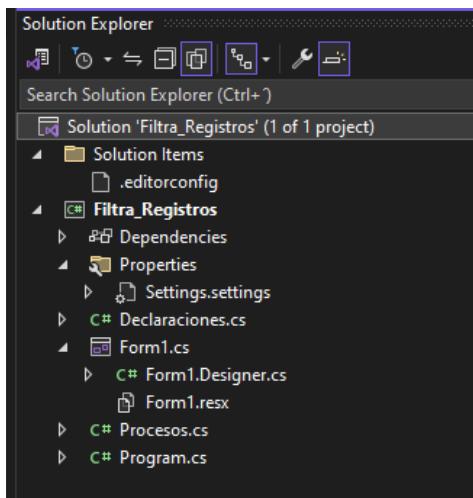
Recorre fila por fila de la grilla. Arma el registro concatenando las columnas + el separador. Escribe en archivo el registro armado.

Cierra puntero.

Mensaje informativo al usuario.

===== FIN LADO O PARTE PB =====

La estructura del proyecto/solución:



El contenido del las clases usadas:

- Se gregaron dos clases al proyecto: "Procesos.cs" y "Declaraciones.cs"

Contenido de "Declaraciones.cs":

```

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

namespace Variables
{
    23 references
    internal class Declaraciones
    {

        public static DataTable tabla = new DataTable();
        public static string connectionString = @"Data Source=localhost\SQLSERVERLAP;Initial Catalog=DemoDB;Trusted_Connection=True";
        public static SqlConnection conexionDb = new SqlConnection(Declaraciones.connectionString);
        public static string consulta = "";
        public static DataTable dt = new DataTable();
        public static SqlCommand comando;
    }
}

```

Contenido de "Procesos.cs":

```

using System;
using System.Data;
using System.Data.SqlClient;
using Variables;

namespace Procesos
{
    2 references
    internal class Funciones
    {
        1 reference
        public static DataTable cargaTabla(string consulta, int tipo)
        {
            SqlDataReader dataReader;

            Declaraciones.comando = new SqlCommand(consulta, Declaraciones.conexionDb);
            if (tipo == 1)
            {
                try
                {
                    dataReader = Declaraciones.comando.ExecuteReader();
                    Declaraciones.dt.Load(dataReader);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString());
                }
            }
            else
            {
                try
                {
                    Declaraciones.comando.ExecuteNonQuery();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString());
                }
            }

            return Declaraciones.dt;
        }

        1 reference
        public static string armaQry( int cantCol)
        {
            string qry = "CREATE TABLE Datos (";
            for (int i = 0; i < cantCol; i++)
            {
                if (i < (cantCol - 1))
                    qry = qry + "campo" + i + " TEXT" + ',';
                else
                    qry = qry + "campo" + i + " TEXT";
            }

            qry = qry + ")";

            return qry;
        }
    }
}

```

El proyecto completo (Descomprimir en C:\POO):



Filtrar_Registros.rar

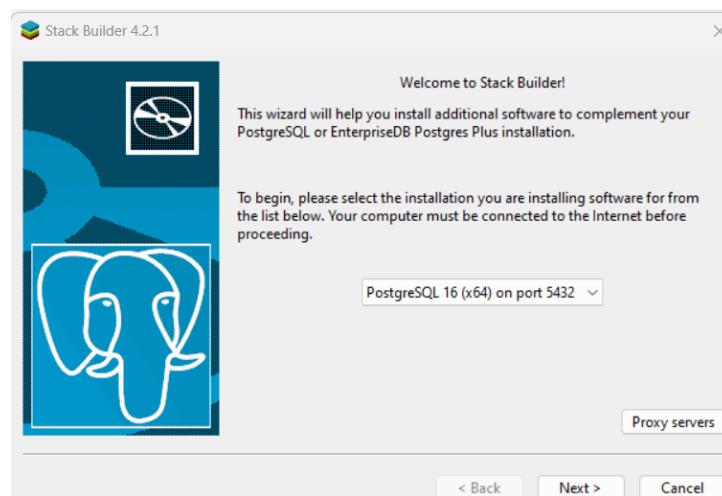
25. CONECTANDO A UNA DB DE PostgreSQL

Instalando PostgreSQL: De la pagina oficial, la última versión...

The screenshot shows the EnterpriseDB PostgreSQL download page. It features a large "Download PostgreSQL" button at the top. Below it, a table lists PostgreSQL versions and their availability across different platforms:

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
16.3	postgresql.org	postgresql.org	postgresql.org	postgresql.org	Not supported
15.7	postgresql.org	postgresql.org	postgresql.org	postgresql.org	Not supported
14.12	postgresql.org	postgresql.org	postgresql.org	postgresql.org	Not supported

Instalar de forma completa, una vez que finaliza dejar que abra el Stack Builder...



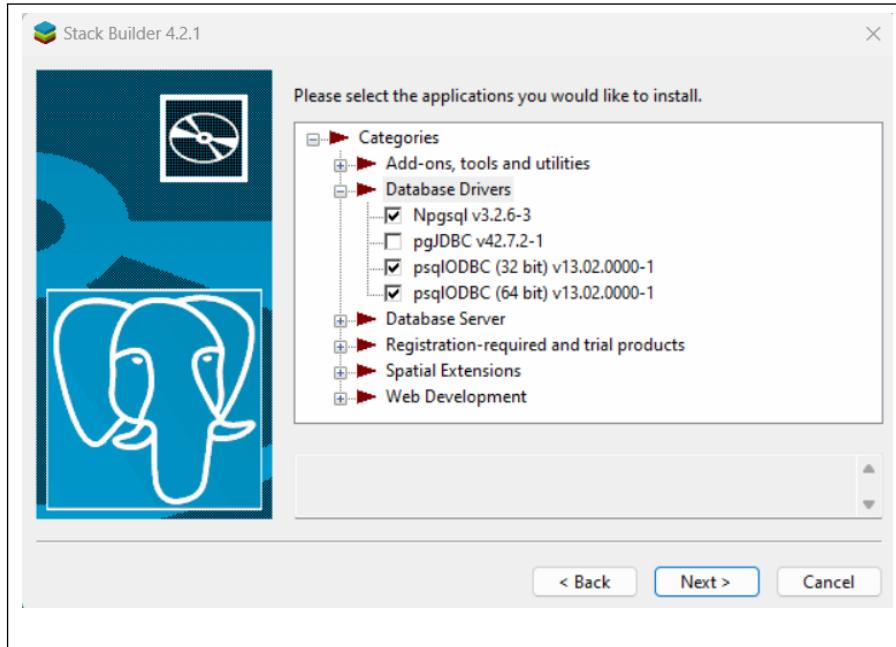
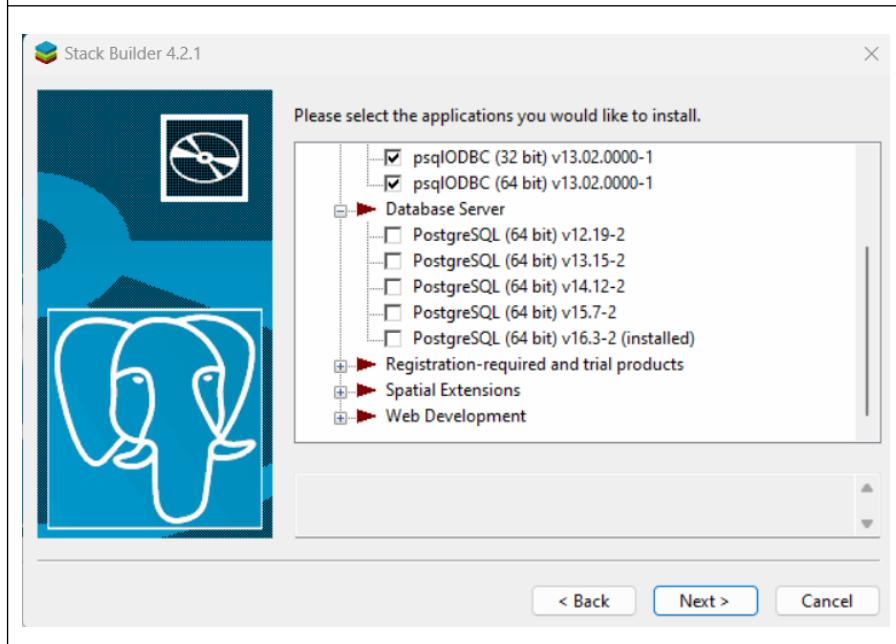
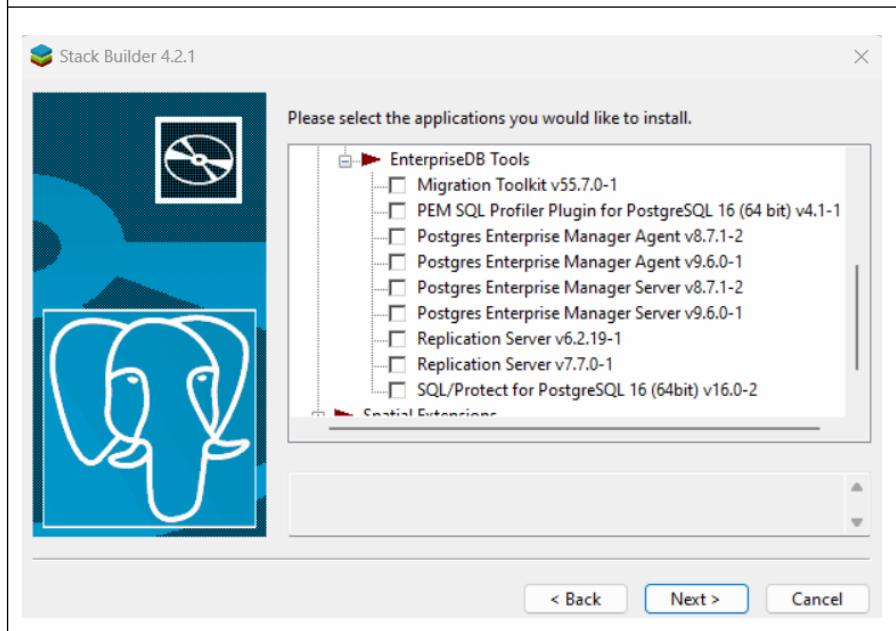
De las opciones que brinda el Stack tenemos:

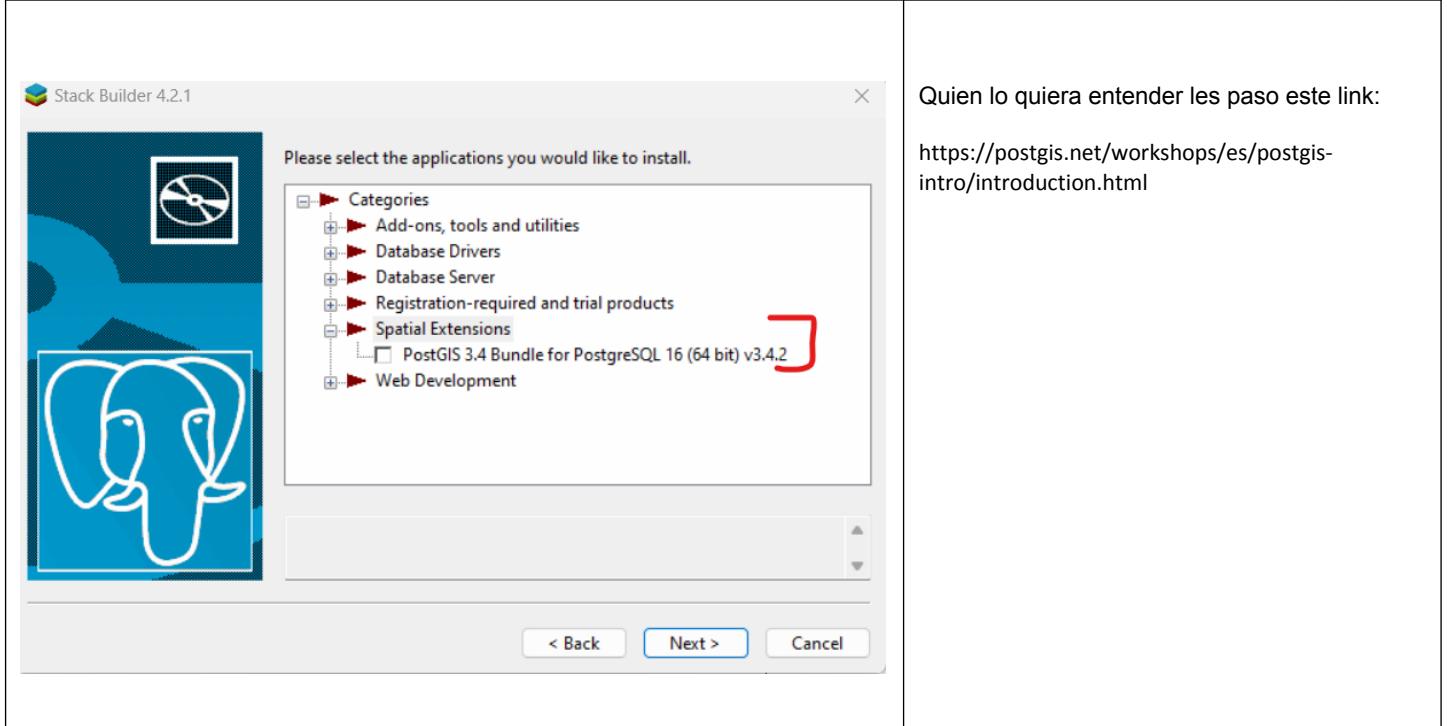
The screenshot shows the Stack Builder application window. On the left, there's a sidebar with the PostgreSQL logo and the text "Stack Builder 4.2.1". The main area has a title "Please select the applications you would like to install." and a tree view of categories:

- Categories
 - Add-ons, tools and utilities
 - EDB Language Pack v4.3-1
 - pgAgent (64 bit) for PostgreSQL 16 v4.2.2-1
 - pgBouncer v1.22.1-1
 - Database Drivers
 - Database Server
 - Registration-required and trial products
 - Spatial Extensions
 - Web Development

On the right side of the window, there are two columns of text describing specific packages:

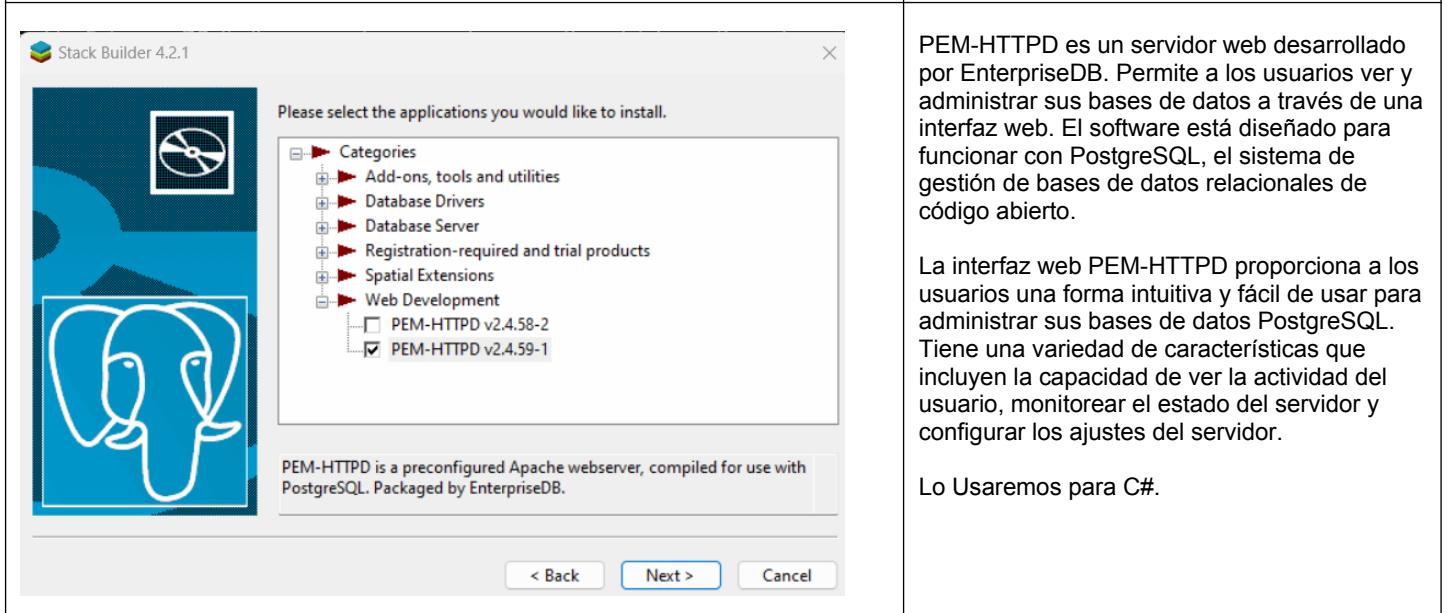
- EDB Language Pack:** brinda soporte sobre lenguajes como Perl, Python y Tcl.
- pgAgent:** un cron para generar tareas programadas sobre la DB.
- pgBouncer:** sirve para generar y mantener una serie de conexiones abiertas disponibles para los usuarios y administrar su uso.

	<p>Npgsql: .NET para el Servidor (vamos a precisarlo para C#).</p> <p>pgJDBC: OBDC para Java</p> <p>psqlODBC: Driver de ODBC (lo vamos a precisar)</p>
	<p>Diferentes versiones disponibles de Servidor Postgres (nosotros ya tendremos la última instalada).</p>
	<p>Es para generar automatización de operaciones, aplicación de parches al motor de DB, y otras cosas que es mas que nada de uso empresarial.</p>



Quien lo quiera entender les paso este link:

<https://postgis.net/workshops/es/postgis-intro/introduction.html>

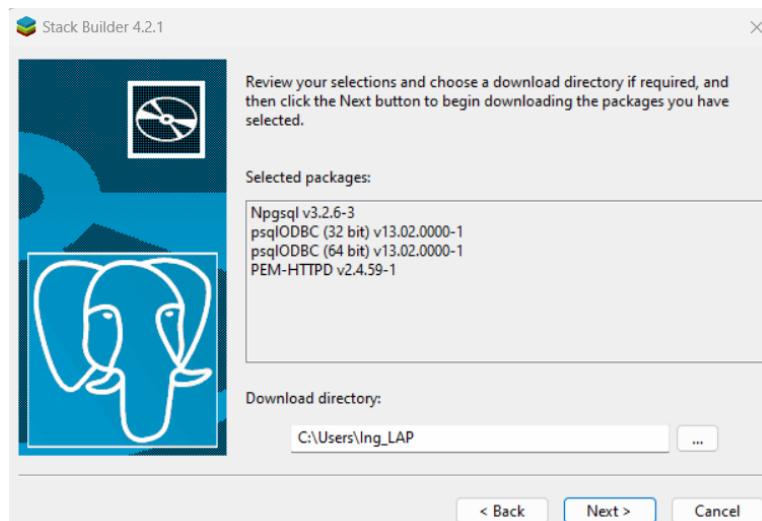


PEM-HTTPD es un servidor web desarrollado por EnterpriseDB. Permite a los usuarios ver y administrar sus bases de datos a través de una interfaz web. El software está diseñado para funcionar con PostgreSQL, el sistema de gestión de bases de datos relacionales de código abierto.

La interfaz web PEM-HTTPD proporciona a los usuarios una forma intuitiva y fácil de usar para administrar sus bases de datos PostgreSQL. Tiene una variedad de características que incluyen la capacidad de ver la actividad del usuario, monitorear el estado del servidor y configurar los ajustes del servidor.

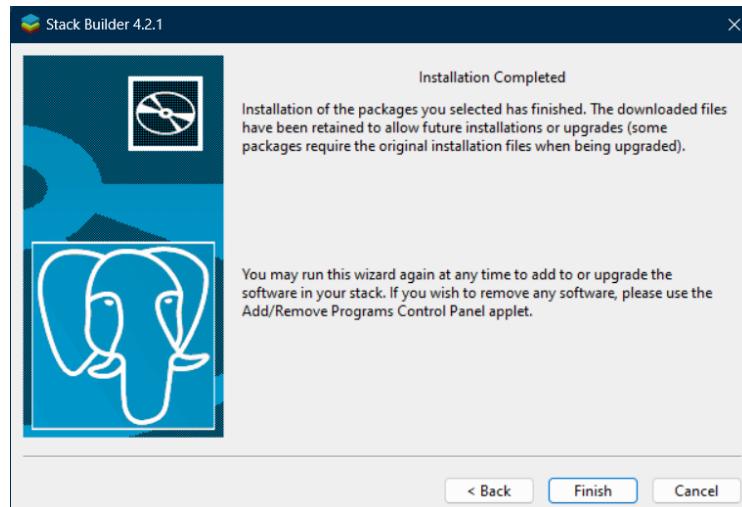
Lo Usaremos para C#.

Luego de seleccionar y dar Siguiente queda que se instalará esto:

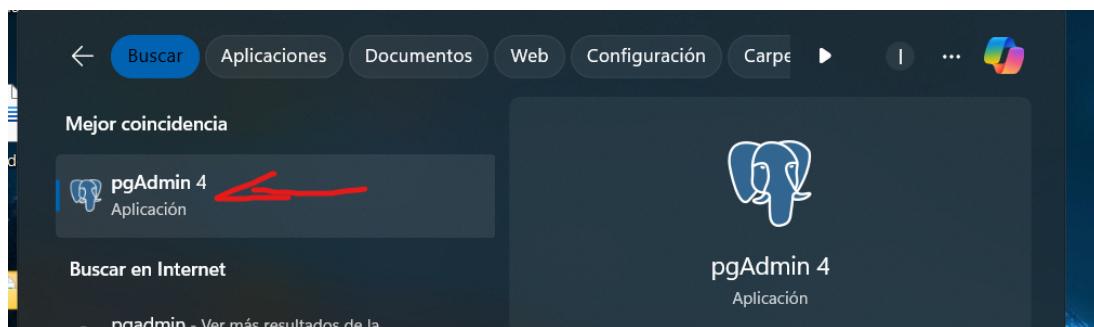


Luego siempre darle Next a cada paquete de instalación....

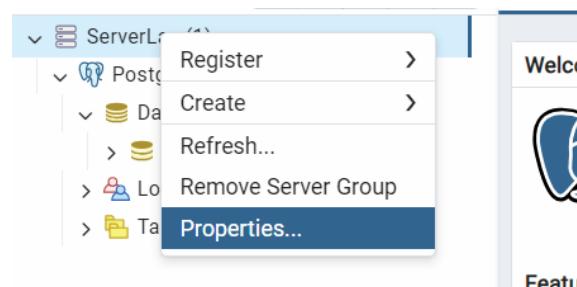
Hasta que aparezca que terminó:



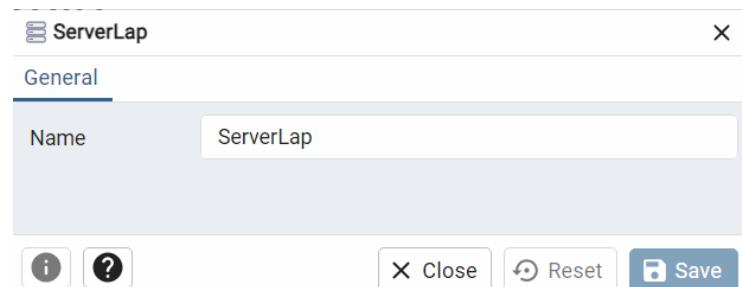
Luego abrir el PgAdmin:



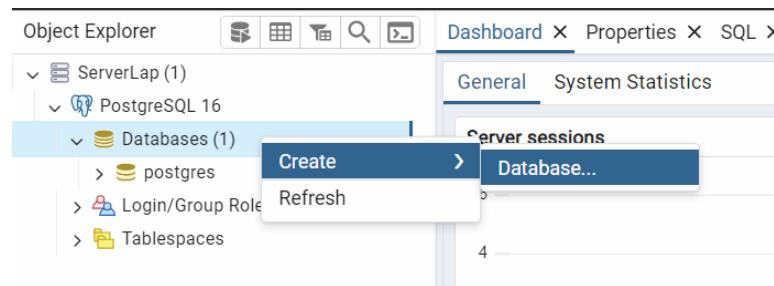
Una vez abierto recomiendo cambiar el nombre del Servidor:



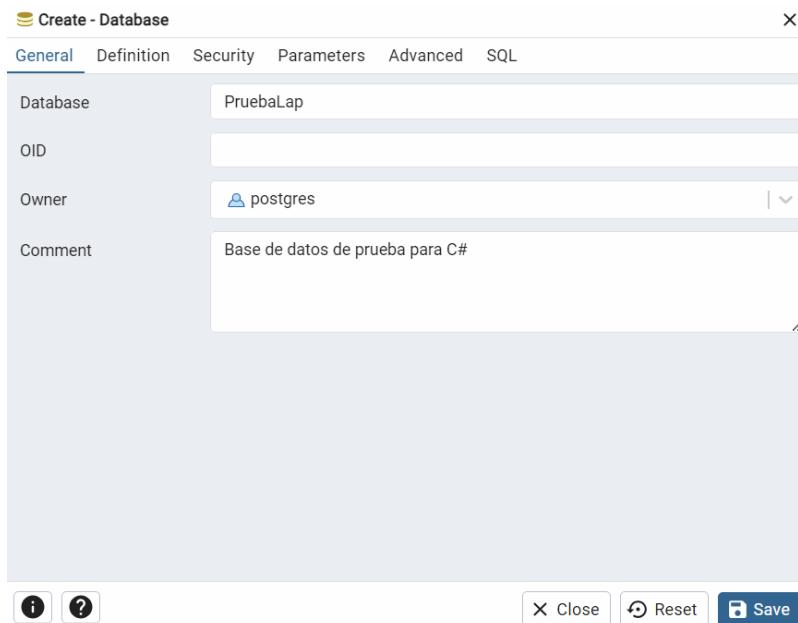
Yo le puse así:



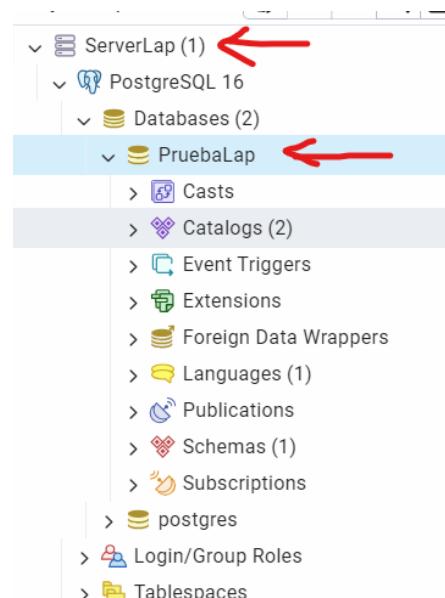
Ahora vamos a crear una nueva DB:



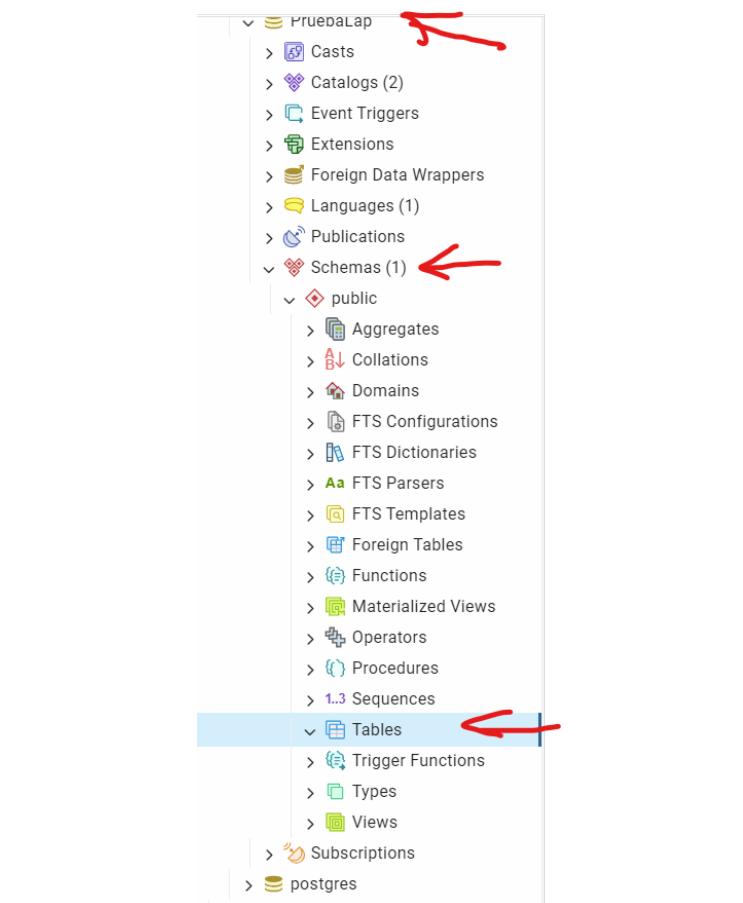
Yo le puse de nombre PruebaLap:



Y me quedó así:



Crearé una tabla:



Create - Table

General Columns Advanced Constraints Partitions Parameters Security SQL

Name	Personas
Owner	postgres
Schema	public
Tablespace	pg_default
Partitioned table?	<input checked="" type="checkbox"/>
Comment	<input type="text"/>

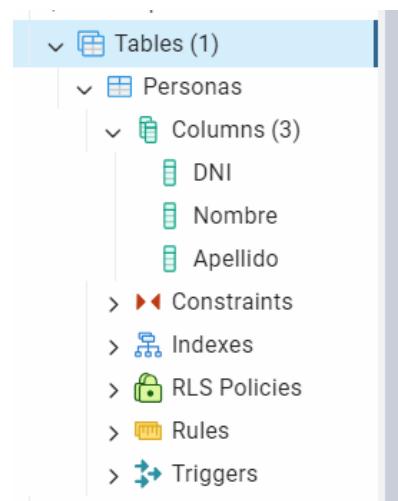
Buttons: Close, Reset, Save

The screenshot shows the 'Create - Table' dialog with the 'Columns' tab selected. The table structure is defined as follows:

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
<input checked="" type="checkbox"/>	DNI	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	Nombre	text			<input type="checkbox"/>	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	Apellido	text			<input type="checkbox"/>	<input type="checkbox"/>	

Buttons at the bottom include 'Close', 'Reset', and 'Save'.

Quedó así:



Probando una consulta sobre la Tabla creada:

The screenshot shows a PostgreSQL client interface with the following details:

- SQL tab: The query is `select * from public.\"Personas\";`.
- Connection: PruebaLap/postgres@PostgreSQL 16*
- Toolbar: Includes icons for file operations, search, and navigation.
- Result pane: Shows the table structure:

DNI	Nombre	Apellido
-----	--------	----------
- Bottom tabs: Data Output, Messages, Notifications.

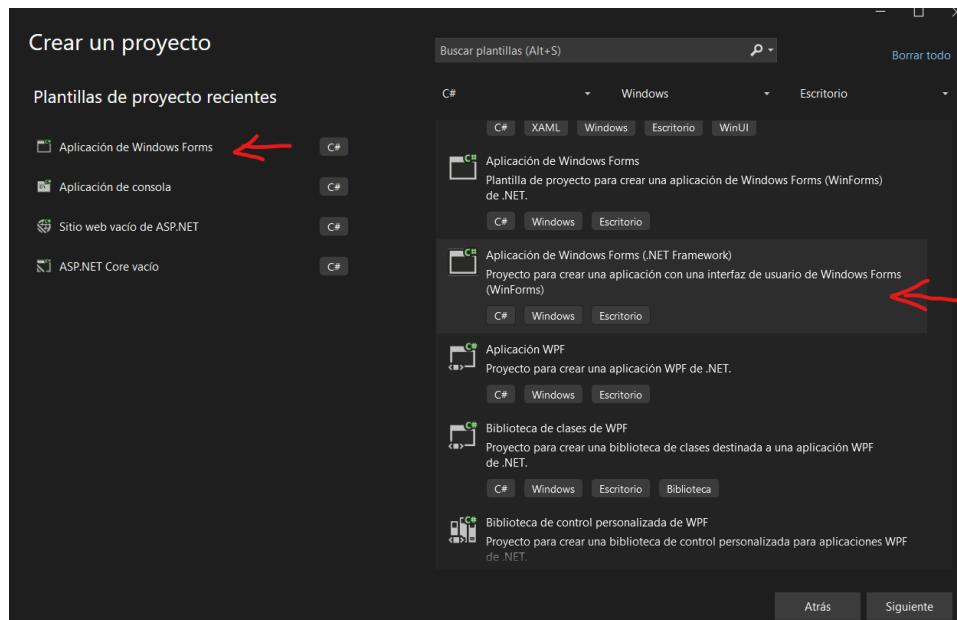
Insertando un registro:

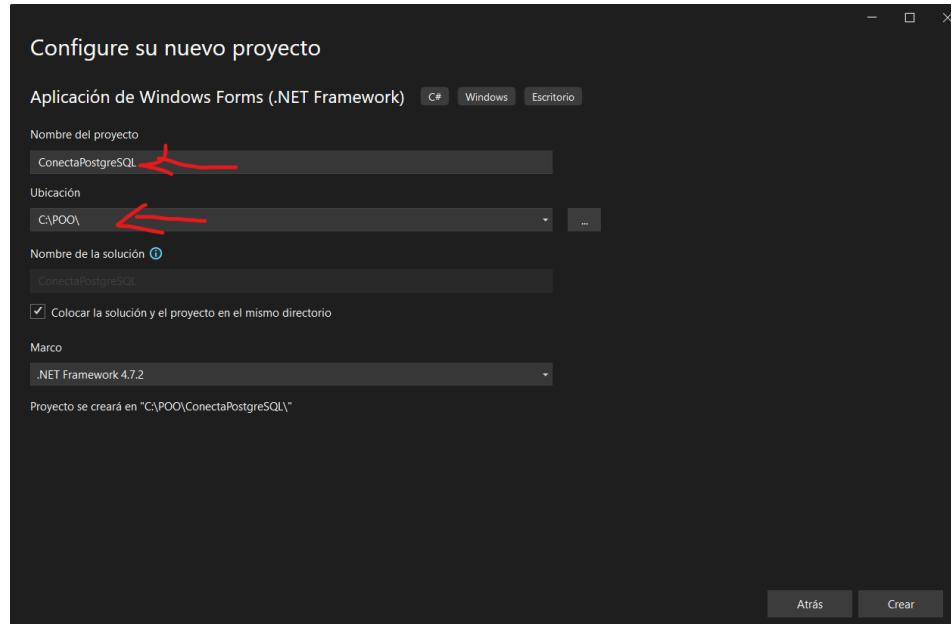
The screenshot shows the pgAdmin 4 interface. In the top bar, the connection is set to "PruebaLap/postgres@PostgreSQL 16". The toolbar includes various icons for database management. Below the toolbar, the "Query" tab is selected, showing the SQL command: `Insert into public."Personas" values (123456789, 'Pepe', 'LePuag');`. The "Data Output" tab is active, displaying the result of the query: `INSERT 0 1`. Below this, a message states: `Query returned successfully in 79 msec.`

Consultando nuevamente:

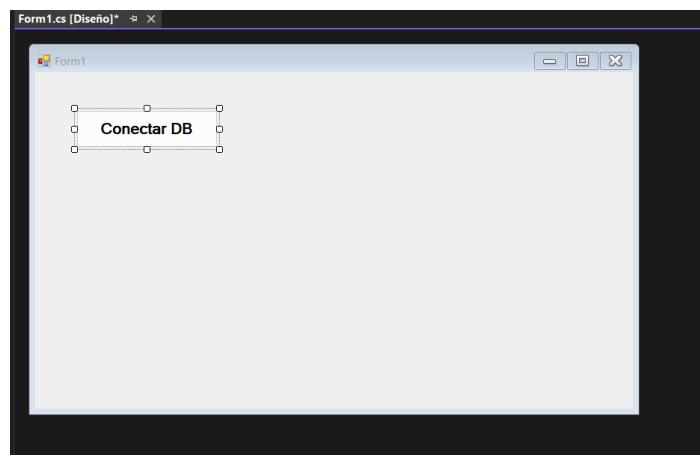
The screenshot shows the pgAdmin 4 interface again. The connection is still "PruebaLap/postgres@PostgreSQL 16". The "Query" tab is selected with the command: `Select * from public."Personas";`. The "Data Output" tab is active, showing a table with three columns: DNI [PK] integer, Nombre text, and Apellido text. The single row contains the values: 123456789, Pepe, and LePuag.

Ahora pasamos a Visual Studio:

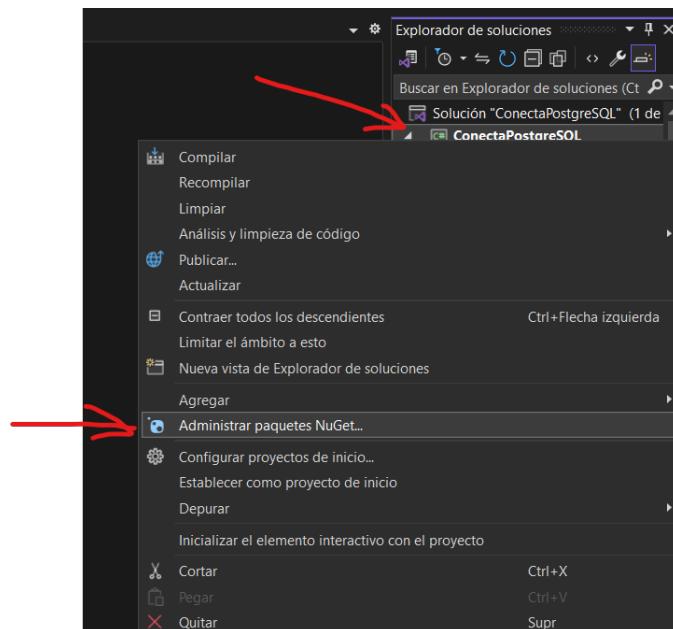




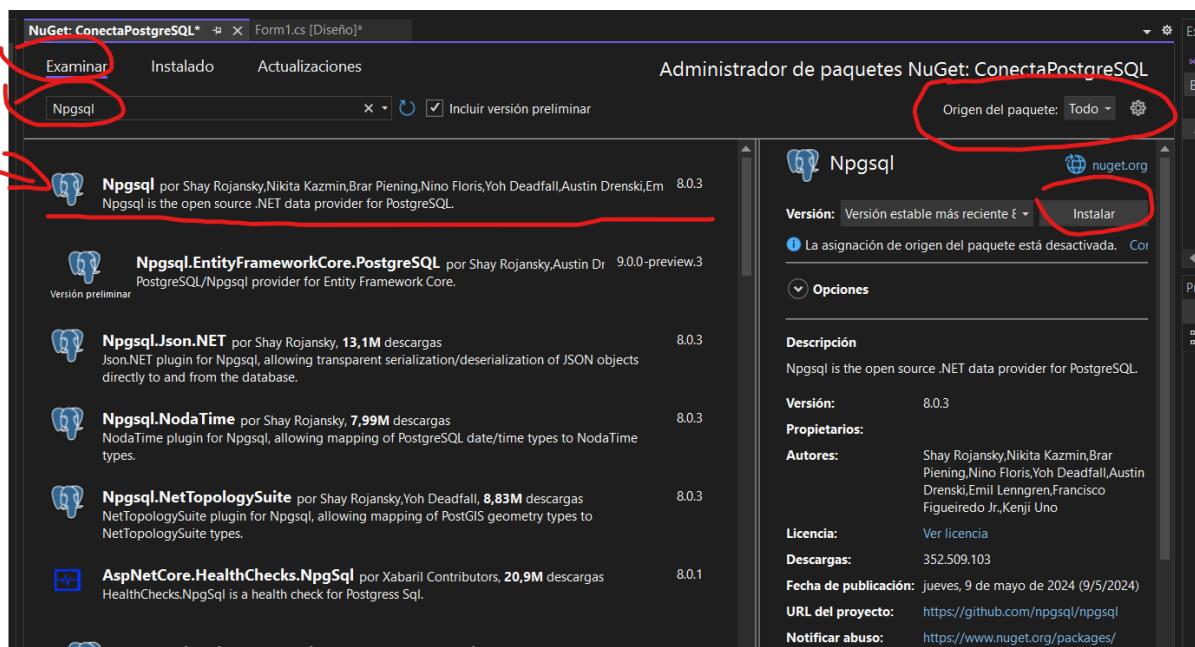
Agregar un Boton, cambiarle el nombre a btn_conectaDB:



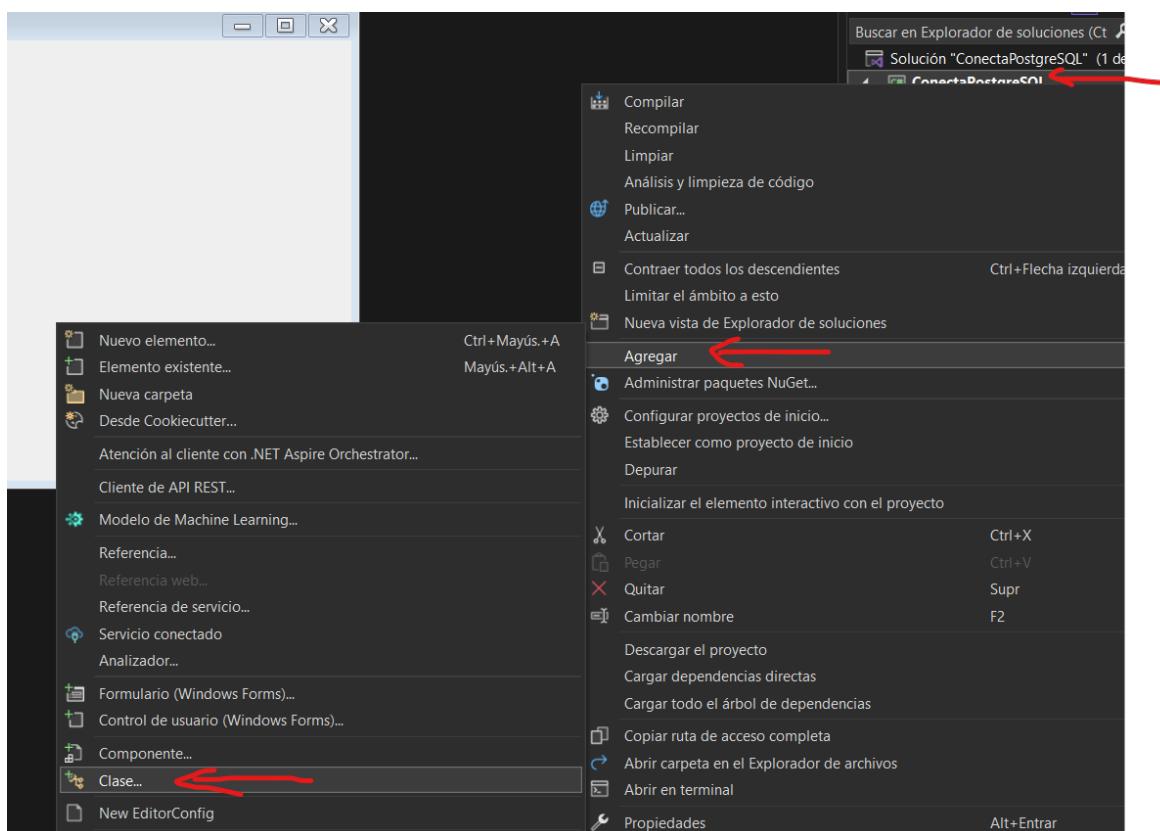
Luego sobre el nombre del proyecto, hacer click con botón derecho del mouse y elegir esta opción:

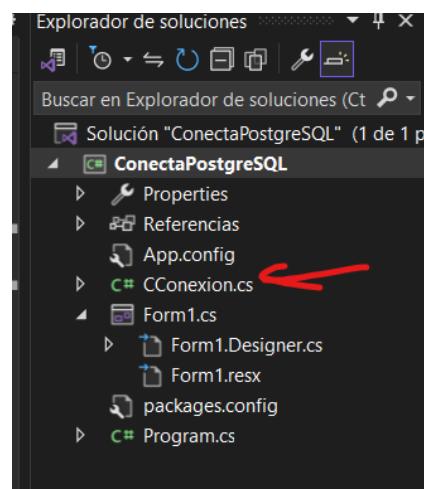


Instalar el NuGet:



Agregar una Clase al proyecto:





Código dentro de la clase:

```
CConexion.cs*  Form1.cs [Diseño]
SQL
using System.Windows.Forms;
using Npgsql;

namespace ConectaPostgreSQL
{
    public class CConexion
    {
        public static NpgsqlConnection conex = new NpgsqlConnection();

        static string servidor = "localhost", DB = "PruebaLap", user = "postgres", pass = "admin", puerto = "5432";

        public static string cadenaConex = "server=" + servidor + ";" + "port=" + puerto + ";" + "user id=" + user + ";" + "password=" + pass + ";" + "database=" + DB + ";";

        public static NpgsqlConnection establerConex()
        {
            try
            {
                conex.ConnectionString = cadenaConex;
                conex.Open();
                MessageBox.Show("DB abierta OK");
            }
            catch(NpgsqlException e)
            {
                MessageBox.Show("ERROR: " + e.ToString());
            }

            return conex;
        }
    }
}
```

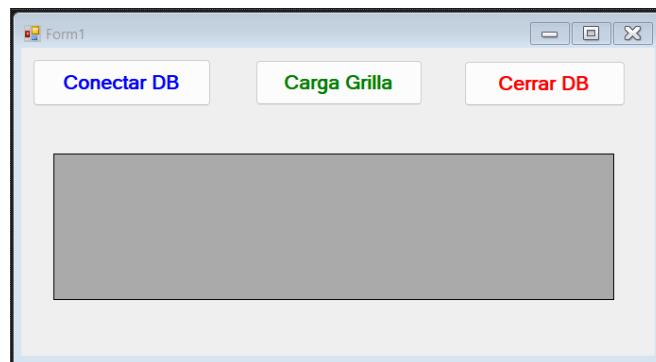
Código dentro del Botón que ConectaDB:

```
Form1.cs  CConexion.cs  Form1.cs [Diseño]
CConexion.cs
  3  using System.Windows.Forms;
  4
  5  namespace ConectaPostgreSQL
  6  {
  7      public partial class Form1 : Form
  8      {
  9          public Form1()
 10          {
 11              InitializeComponent();
 12          }
 13
 14          private void btn_conectaDB_Click(object sender, EventArgs e)
 15          {
 16              CConexion.establerConex(); highlighted with a red arrow
 17          }
 18      }
 19  }
 20
```

Prueba de conexión:



Agregamos un par de botones más y un DataGridView:



Más código dentro de la Clase:

```

using System;
using System.Windows.Forms;
using Npgsql;

namespace ConectaPostgreSQL
{
    public class CConexion
    {
        public static NpgsqlConnection conex = new NpgsqlConnection();

        static string servidor = "localhost", DB = "PruebaLap", user = "postgres", pass = "admin", puerto = "5432";

        public static string cadenaConex = "server=" + servidor + ";" + "port=" + puerto + ";" + "user id=" + user + ";" + "password=" + pass + ";" + "database=" + DB + ";";

        public static NpgsqlConnection establecerConex()
        {
            try
            {
                conex.ConnectionString = cadenaConex;
                conex.Open();
                MessageBox.Show("DB abierta OK");
            }
            catch(NpgsqlException e)
            {
                MessageBox.Show("ERROR: " + e.ToString());
            }

            return conex;
        }

        public static NpgsqlConnection cerrarConex()
        {
            try
            {
                conex.Close();
                MessageBox.Show("DB cerrada OK");
            }
            catch (NpgsqlException e)
            {
                MessageBox.Show("ERROR: " + e.ToString());
            }

            return conex;
        }
    }
}

```

Código del Botón CerrarDB:

```

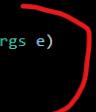
using System;
using System.Collections.Generic;
using System.Windows.Forms;

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

        private void btn_conectaDB_Click(object sender, EventArgs e)
        {
            CConexion.establecerConex();
        }

        private void btn_cierraDB_Click(object sender, EventArgs e)
        {
            CConexion.cerrarConex();
        }
    }
}

```



Código en el botón Carga Grilla:

```

private void btn_cargaDTG_Click(object sender, EventArgs e)
{
    dataGridView.DataSource = CConexion.conusltarDB();
    dataGridView.ClearSelection();
}

```

Nueva función dentro de la Clase:

```

public static DataTable conusltarDB()
{
    NpgsqlCommand conexCommand = new NpgsqlCommand("Select * from public.\"Personas\";", conex);
    DataTable tabla = new DataTable();
    NpgsqlDataReader dr = conexCommand.ExecuteReader();
    tabla.Load(dr);
    return tabla;
}

```

26. VALIDACIÓN DE DATOS INGRESADOS

Validar un campo de texto equivale a restringir su contenido al conjunto de caracteres válidos para dicho campo.

Opciones de validación:

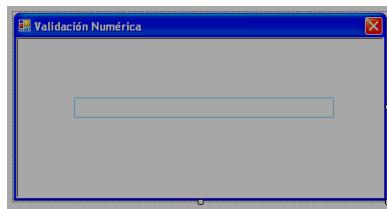
- a- Si la validación de los datos se hace luego de presionar la tecla Enter en la caja de texto, el campo podría contener un dato no válido pero podría ser validado antes de usarlo.
- b- Si la validación se hace verificando por cada tecla que se pulsa (evento KeyPress) el campo contendrá datos ya validados.

Validación usando la opción “a”:

Para validar el contenido de una caja de texto por cada tecla pulsada se debería escribir el método `textBox_KeyPress` quien controle el evento KeyPress ejecutado por el operador. Dicho método, resumiendo, realiza las siguientes operaciones:

- 1- Si pulsa la tecla enter (código ASCII = 13), dá por validada la acción dado que puede deberse al cambio de objeto.
- 2- Se pulsa la tecla retroceso (código ASCII = 8) deja que el sistema controle el evento para que se realice el procedimiento determinado para dicha tecla.
- 3- Si se pulsa la “,” (coma) y ya había una, invalida la acción dando por controlado el evento.
- 4- Si se pulsan los signos + o - , verifica que se tratan del primer carácter ingresado.
- 5- Si se pulso otra tecla que no corresponde a un dígito del 0 al 9 o las ya mencionadas anteriormente se invalida la acción.

En el siguiente ejemplo se muestran estos controles sobre una caja de texto:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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

        private void textBox_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == Convert.ToChar(13))
            {
                // Se pulsó la tecla Enter
                e.Handled = true;
            }
            else if (e.KeyChar == Convert.ToChar(8))
            {
                // Se pulso la tecla retroceso
                e.Handled = false;
            }
            else if (e.KeyChar == ',')
            {
                TextBox ObjtextBox = (TextBox)sender;
                if (ObjtextBox.Text.IndexOf(',') != -1)
                {
                    // Solo puede haber una coma
                    e.Handled = true;
                }
            }
        }
    }
}
```

```

        else if (e.KeyChar == '-' || e.KeyChar == '+')
    {
        TextBox ObjtextBox = (TextBox)sender;
        // Admitir - o + sólo en la primera posición
        if (ObjtextBox.SelectionLength == ObjtextBox.TextLength)
        {
            // Todo el texto seleccionado: se sobrescribe con el signo
            e.Handled = false;
        }
        else if (ObjtextBox.TextLength != 0)
        {
            // La primera posición ya est{a ocupada
            e.Handled = true;
        }
    }
    else if (e.KeyChar < '0' || e.KeyChar > '9')
    {
        // Desechar loscaracteres que no son dígitos
        e.Handled = true;
    }
}
}

```

Validación usando la opción “b”:

Otra forma de validar el contenido de un textBox es añadiendo manejadores para los eventos *Validating* y *Validate*. Estos eventos se producen en el orden descrito cuando el objeto pierde le foco (se cambio de objeto por el usuario asi lo dispuso) siempre y cuando su propiedad *CausesValidation* valga *True*, que es el valor predeterminado.

Para hacer esto, deberán seguir estos pasos atentamente y no saltar ninguno de ellos:

- 1- Arrastren sobre un formulario limpio, desde la caja de herramientas, un componente *TextBox* y denomínelo CajaTexto.
 - 2- Arrastren sobre un formulario limpio, desde la caja de herramientas, un componente *ErrorProvider* y denomínelo ProveedorDeError.
 - 3- Dentro de la clase Form1 agregue un atributo privado *datoCajaTexto*. Este atributo almacenará el valor numérico procedente de la caja de texto.

```
public partial class Form1 : Form
{
    private double datoCajaTexto;

    public Form1()
    {
        InitializeComponent();
    }
}
```

- 4- Para añadir el controlador que validará lo ingresado en la caja de texto diríjase a la ventana de diseño, seleccione la caja de texto, diríjase a la ventana propiedades, seleccione el evento Validating y escriba como nombre para el controlador CajaTexto_Validating. Repitan el mismo procedimiento para añadir al controlador CajaTexto_Validating para el evento Validated quien se encargará luego de de limpiar las notificaciones de error una vez validados los datos.
 - 5- Para el evento Validating, tendrán que ingresar las siguientes líneas de código:

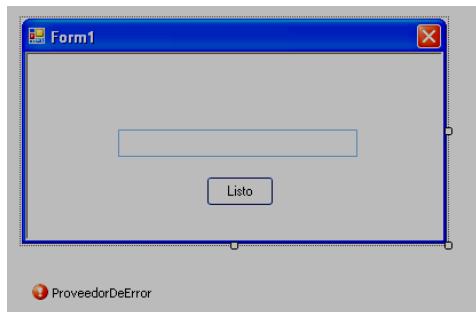
```
private void CajaTexto_Validating(object sender, CancelEventArgs e)
{
    TextBox objTextBox = (TextBox)sender;
    try
    {
        datoCajaTexto = Convert.ToDouble(objTextBox.Text);
    }
    catch (Exception)
    {
        e.Cancel = true;
        objTextBox.SelectAll();
        ProveedorDeError.SetError(objTextBox, "Tiene que ser numerico!!!");
        MessageBox.Show("Tiene que ser numerico!!!");
    }
}
```

- 6- Para el evento Validated, tendrán que ingresar las siguientes líneas de código:

```
private void CajaTexto_Validated(object sender, EventArgs e)
{
    ProveedorDeError.Clear();
}
```

- 7- Agreguen un segundo objeto para poder hacer el cambio de foco y se proceda a validar lo ingresado.

En el siguiente ejemplo se muestran estos controles sobre una caja de texto:



```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Validacion_Numeros_2
{
    public partial class Form1 : Form
    {
        private double datoCajaTexto;

        public Form1()
        {
            InitializeComponent();
        }

        private void CajaTexto_Validating(object sender, CancelEventArgs e)
        {
            TextBox objTextBox = (TextBox)sender;
            try
            {
                datoCajaTexto = Convert.ToDouble(objTextBox.Text);
            }
            catch (Exception)
            {
                e.Cancel = true;
                objTextBox.SelectAll();
                ProveedorDeError.SetError(objTextBox, "Tiene que ser numerico!!!");
                MessageBox.Show("Tiene que ser numerico!!!");
            }
        }

        private void CajaTexto_Validated(object sender, EventArgs e)
        {
            ProveedorDeError.Clear();
        }
    }
}

```

Preguntas (sobre el último ejemplo dado):

- a- ¿Qué objetos forman la interfaz?
- b- ¿Qué eventos hacen que la interfaz responda?
- c- ¿Cuáles son los pasos a seguir para un desarrollo ordenado?

- a- Se incluyen los siguientes objetos:
 - Un formulario.
 - Un textBox.
 - Un botón (Button).
 - Un objeto ErrorProvider.

- b- Eventos que actúan sobre mi interfaz:
 - Eventos KeyPress por cada tecla presionada por el operador.
 - Evento Validating (validación de datos ingresados).
 - Evento Validated (luego de validar los datos borra los mensajes de error).
 - Un posible evento click sobre el botón que en este caso no contiene una acción específica.

- c- Los pasos a seguir son:

- Crear la estructura para una nueva aplicación que utilice un formulario de tipo Form como ventana principal.
- Añadir las propiedades de los componentes.
- Escribir el código para cada uno de los objetos.
- Guardar la aplicación.
- Compilar/ejecutar.
- Corregir los errores si es que los hay luego de la compilación.
- Volver a guardar la aplicación.

Creación de un Menú:

Un menú es una forma de proveer al usuario de un conjunto de órdenes, lógicamente relacionadas, agrupadas bajo un mismo título.

Para crear dicho menú seguir los siguientes pasos:

- 1- Arrastrar desde la caja de herramientas un control MenuStrip sobre el formulario. Dicha acción abrirá de forma automática el editor de menús. Luego agregar cada elemento como se hizo a modo ejemplo en el formulario:



27. UTILIZANDO HILOS (Threads) CON OBJETOS

Muchos lenguajes de programación permiten la creación de hilos o threads en un programa. De forma resumida, los hilos son un mecanismo mediante el cual podemos dividir una aplicación en diferentes partes que se pueden ejecutar de forma paralela, existiendo mecanismos por los que pueden compartir información.

C# ofrece un mecanismo muy sencillo de implementar hilos, basado en la utilización de la clase Thread. El constructor de esta clase recibe como parámetro el método o función que hay que ejecutar en paralelo. Este parámetro se indica mediante la utilización de un delegado, que es el mecanismo que, entre otras cosas, se utiliza en .NET para utilizar punteros a funciones de forma segura. La firma del delegado no incluye ningún parámetro, por lo que únicamente es posible crear hilos de forma directa sobre métodos y funciones que no requieran parámetros de entrada ni de salida. En los siguientes ejemplos muestro un caso sencillo de creación de un hilo y otro en el que explico una forma de poder crear un hilo con entrada y salida de parámetros.

En el siguiente ejemplo se dispone de una clase con dos métodos que muestran mensajes por pantalla. El objetivo es crear dos hilos, uno para cada uno de los métodos y ejecutarlos de forma paralela, de forma que podamos ver como resultado cómo se van intercalando los mensajes escritos por cada método.

```
using System;
using System.IO;
using System.Threading;

public class Mensajes{
    public void Mostrar1()
    {
        for(int i=0;i<10;i++){
            Console.WriteLine("Escribiendo desde ==> 1");
            Thread.Sleep(1000);
        }
    }

    public void Mostrar2()
    {
        for(int i=0;i<10;i++){
            Console.WriteLine("Escribiendo desde ==> 2");
            Thread.Sleep(1000);
        }
    }
}

public class Ejemplo{

    public static void Main()
    {
        Mensajes msg = new Mensajes();

        Thread th1 = new Thread(new ThreadStart(msg.Mostrar1));
        Thread th2 = new Thread(new ThreadStart(msg.Mostrar2));

        th1.Start();
        th2.Start();

        th1.Join();
        th2.Join();
    }
}
```

La creación de cada hilo se realiza mediante las líneas `Thread th1 = new Thread(new ThreadStart(msg.Mostrar1));`. Esta línea indica que se crea una instancia de la clase Thread, con nombre th1, a partir de un delegado de la clase ThreadStart, que apunta al método Mostrar1 del objeto msg creado anteriormente.

Una vez creados los dos hilos hay que activarlos, para lo que se llama al método Start de cada uno de ellos. Tras este punto cada hilo se ejecuta en paralelo entre si, y con el programa principal, por lo que utilizamos el método Join de ambos hilos para esperar a que terminen los hilos antes de finalizar el programa.

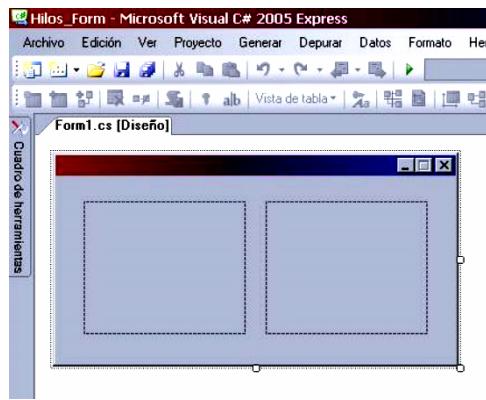
En el siguiente ejemplo, lo que se debe entender primero es que existen dos hilos: "p1" y "p2" (o dos objetos de la clase "Thread", en inglés "Hilo") cada uno asociado a un método: A "p1" le corresponde "Hilo1" y a "p2" le corresponde "Hilo2". Lo que "p1" y "p2" permiten es que ambos métodos (Hilo1 e Hilo2) se ejecuten de manera simultánea.

Todas las otras variables en el programa son de control. Los hilos empiezan a ejecutarse al llamar al método "Start" y se terminan con "Abort". El C# no permite llamar de nuevo a "Start" para un hilo que ya se está ejecutando.

Lo que hace el programa simplemente es variar el color de ambos Picture Boxes en un bucle que jamás termina, para ello se usa el comando `CualquierPictureBox.BackColor = Color.FromArgb(100, g, 80);`

También uso el método `Thread.Sleep` que detiene la ejecución del hilo durante el tiempo en milisegundos que va entre paréntesis. Esto es para que los colores varíen despacio y se puedan apreciar.

Cada vez que dentro de un método se usa la palabra "Thread" ésta se refiere al hilo que está asociado a él.



```

using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        Thread p1;
        Thread p2;

        byte r, g;
        bool b1, b2;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            r = 0; g = 255; b1 = false; b2 = true;

            p1 = new Thread(new ThreadStart(Hilo1));
            p2 = new Thread(new ThreadStart(Hilo2));

            p1.Start();
            p2.Start();
        }

        public void Hilo1()
        {
            while (true)
            {
                Thread.Sleep(10);

                if (r >= 0 && r <= 255 && b1 == false)
                {
                    r++;
                    if (r == 255)
                        b1 = true;
                }

                if (r >= 0 && r <= 255 && b1 == true)
                {
                    r--;
                    if (r == 0)
                        b1 = false;
                }

                pictureBox1.BackColor = Color.FromArgb(r, 80, 100);
            }
        }

        public void Hilo2()
        {
            while (true)
            {
                Thread.Sleep(10);

                if (g >= 0 && g <= 255 && b2 == false)

```

```
{  
    g++;  
  
    if (g == 255)  
        b2 = true;  
}  
  
if (g >= 0 && g <= 255 && b2 == true)  
{  
    g--;  
  
    if (g == 0)  
        b2 = false;  
}  
  
pictureBox2.BackColor = Color.FromArgb(100, g, 80);  
}  
}  
  
private void Form1_FormClosed(object sender, FormClosedEventArgs e)  
{  
    p1.Abort();  
    p2.Abort();  
}  
}  
}
```

Por supuesto todo lo anterior son únicamente ejemplos muy sencillos, ya que la programación de un sistema multihilo suele ser bastante compleja debido al indeterminismo implícito en la ejecución de múltiples hilos de ejecución, que normalmente comparten recursos o dependen unos de otros. En cualquier caso sí que nos permite apreciar la facilidad con la que se pueden crear estas estructuras en C#, olvidándonos de complejas sintaxis y librerías, y centrándonos únicamente en los requisitos de nuestro sistema.

28. MANEJO DE EXCEPCIONES EN C#

C# proporciona soporte integrado para el manejo de excepciones o de una forma más formal situaciones anómalas de funcionamiento, las cuales pueden ocurrir en cualquier momento durante la ejecución del programa y son manejadas por el código que se encuentra fuera del flujo normal de control. Todo esto gracias a las palabras clave try, throw, catch y finally. C# proporciona una solución estructurada tanto a nivel del sistema como de aplicación. A pesar de que es muy similar a C++ en cuanto al manejo de excepciones existen varias diferencias, entre ellas que cada excepción está representada por una instancia de un tipo de clase derivado de System.Exception. En realidad es algo bastante simple:

```
try
{
    has esto...
    si i = 0 throw una excepción
}
catch
{
    si fallo has esto...
}
finally
{
    haya fallado o no, has esto...
}
```

Como se puede apreciar el manejo de excepciones es bastante sencillo y fácil de entender aunque no tengamos mucha experiencia programando, todo aquello que se puso entre las {} del try es un segmento de código en el que puede o no generarse un error en tiempo de ejecución(excepción), en caso de que haya habido un funcionamiento anómalo en el programa(excepción) la ejecución del código entra en el segmento de código catch y ejecuta el bloque de instrucciones que hemos definido para manejar ese error, finalmente el flujo del programa haya o no habido excepción entra en finally aquí podemos poner rutinas para marcar los objetos que ya no se utilizarán de manera que el recolector de basura pueda liberar la memoria que dichos objetos ocupaban, rutinas que guarden un log de la aplicación para llevar un control de ¿cuántas veces ha fallado?, ¿porqué fallo?, etc., todo bloque try puede tener uno o más catch para tratar cada una de las posibles excepciones, pero la flexibilidad de C# va más allá de eso, ya que nos permite lanzar nuestras propias excepciones, por ejemplo si un método no recibe un valor que debe recibir o recibe un valor que no puede procesar podemos lanzar nuestra propia excepción. Ejemplo (para recordar viejos tiempos es un proyecto de tipo consola):

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

namespace Excepciones_1_Console
{
    class Program
    {
        static void Main(string[] args)
        {
            //Declaramos las variables
            int i;
            int j;

            //Iniciamos el Bloque "try"
            try
            {
                //Dentro de este bloque pondremos el código a evaluar
                i = 10;
                j = 0;
                int res;
                res = i / j;
                System.Console.WriteLine("El Resultado es:" + " " + res);
            }
            catch (Exception ex)
            {
                //Este bloque representa la excepción y nos dará el error
                //en caso de que lo haya
                System.Console.WriteLine(ex.Message);

            }
            finally
            {
                //Aquí leemos el resultado escrito en la consola
                //Opcional por si quiere utilizar el resultado para algo más
                System.Console.ReadLine();
            }
        }
    }
}
```

29. PROGRAMACIÓN EN 3 CAPAS CON VISUAL C#

Las aplicaciones han pasado por un proceso evolutivo enorme. Desde su inicio con las aplicaciones monolíticas donde en una aplicación todo estaba ligado o mezclado por decirlo de alguna manera. Es decir, la interfaces de usuario, la lógica de cómo funcionaba la empresa y el manejo de la información almacenada y recuperada estaban juntas.

Luego la industria ha implementado un nuevo modelo de aplicaciones, las aplicaciones distribuidas cliente/servidor, que se convirtió en el estándar por un tiempo. Pero con la llegada de las aplicaciones web se hacía necesario un nuevo estándar para la operaciones de los sistemas, y es por esto que se ha propuesto el modelo de las aplicaciones en n-capas.

Este modelo por lo general está basado en un esquema de tres partes: Acceso, Lógica de negocios e interfaces de usuario. Aunque es posible continuar sub dividiendo este modelo en sub capas para una mayor flexibilidad en la distribución en el equipo de desarrollo y durante el mantenimiento. Veamos en que consiste.

Arquitectura de Aplicaciones en n-capas Se ha convertido en el estándar para el software empresarial. Se caracteriza por la descomposición de las aplicaciones.

- Proporciona una escalabilidad, capacidad de administración y utilización de recursos mejorados.
- Cada capa es un grupo de componentes que realiza una función específica.
- Se puede actualizar una capa sin recompilar otras capas.

Por regla general, la capa de la presentación es una interfaz gráfica que muestra los datos a los usuarios.

La capa de la lógica de negocios es responsable de procesar los datos recuperados y enviarlos a la capa de presentación.

La capa de datos almacena los datos de la aplicación en un almacén persistente, tal como una base de datos relacional o archivos XML.

Se pueden alojar todas las capas en el mismo servidor, pero también es posible alojar cada capa en varios servidores.

Nota: Las aplicaciones distribuidas no necesariamente tienen que estar construidas en base a los web service, estos se utilizan cuando se requiere que la información sea compatible entre plataformas. En caso en que está no sea la prioridad entonces es opcional utilizarla o implementar la distribución de la aplicación por medio de Remoting.Net.

Objetivos:

El objetivo principal de este planteamiento es separar y, por lo tanto independizar, las diversas capas de la Programación Visual.

La capa de aplicación (1ra capa) corresponde a programas interactivos o procesos, que realicen alguna acción sobre los diversos componentes. La capa intermedia (2da capa) estará formada por las clases, quienes interactúan con el medio de almacenamiento que está representado en la capa de datos (3er capa).

Una arquitectura pura de tres capas no permite ninguna forma de comunicación directa entre la aplicación y la capa de datos; esto es, entre los programas de usuario y la base de datos. Los programas de aplicación deben solicitar todos sus requerimientos a la capa de clases, a través de la invocación de servicios, que corresponden a los métodos definidos para cada una de ellas.

Lo más común estaría representado por:

Lo mejor y más correcto sería que la primer capa sea el fuente o ejecutable gráfico y solo presente los datos en un control tipo flexgrid por ejemplo, en la segunda capa la programación requerida en una dll o componente que te permita el interactuar con los controles de la primera y con stored procedures o consultas directas a la base de datos y desconectarse una vez obtenida la información, y la tercer capa serían los stored procedure de explotación de la base de datos misma, selecciones, inserciones, borrados o actualizaciones.

30. ANEXO 7 - Tutoriales de ayuda para 3 capas

Los siguientes tutoriales fueron desarrollados por Mauricio Torres Ramos, ayudante de cátedra de la materia durante los años 2010 y 2011:



Tutorial_3Capas.rar

Los siguientes tutoriales y ejemplos fueron desarrollados por Marcos Prado, ayudante de cátedra de la materia durante el año 2016 y 2017:



Tutorial de
programación en tres



TresCapas con
ErrorProvider.rar

31. ANEXO 8 - QUE ES EL e.Handled

En el caso del evento **KeyPress**, para saber qué tecla se ha pulsado, debemos usar **e.KeyChar**, (que sería el equivalente al parámetro KeyAscii en VB clásico), pero la propiedad KeyChar es de solo lectura, por tanto no se puede asignar un valor a dicha propiedad.

El hecho de asignar un valor cero a KeyAscii es para indicar que no se debe tener en cuenta la pulsación de dicha tecla, esto en .NET se hace asignando un valor verdadero a la propiedad **Handled** del objeto **e**: **e.Handled = True**

Con esto indicamos que esa tecla ha sido "manejada" por nuestro evento y se ignorará.

Pero en otras ocasiones, lo que nos interesa es cambiar la tecla pulsada por otra diferente.

Por ejemplo, si en nuestra configuración regional tenemos que los decimales se representan por la coma, podíamos querer que al pulsar en el punto, ésta pulsación se convierta en una coma.

Pero como acabamos de ver, en .NET no se puede asignar un nuevo valor a la propiedad KeyChar, pero para solucionarlo, podemos echar mano de la clase SendKeys (a diferencia del VB clásico, ahora no es una función de VB sino una clase de .NET Framework) y lanzar la pulsación de la tecla que nos interese: **SendKeys.Send(",")**.

Pero si hacemos sólo esto, nos encontramos con dos pulsaciones: el punto que el usuario ha pulsado y la coma que nosotros hemos "enviado" al teclado.

La solución es la misma que acabamos de ver anteriormente: indicar que el usuario no ha pulsado nada: **e.Handled = True**, con esto sólo quedará la tecla que nosotros hemos enviado con **SendKeys**.

A continuación te muestro el código de un formulario de prueba.

```

private void txtNumero_KeyPress(object sender,
                               System.Windows.Forms.KeyPressEventArgs e)
{
    // Si se pulsa la tecla Intro, pasar al siguiente
    //if( e.KeyChar == Convert.ToChar('\r') ){
    if( e.KeyChar == '\r' ){
        e.Handled = true;
        txtFecha.Focus();
    }else if( e.KeyChar == '.' ){
        // si se pulsa en el punto se convertirá en coma
        e.Handled = true;
        SendKeys.Send(",");
    }
}

private void txtFecha_KeyPress(object sender,
                               System.Windows.Forms.KeyPressEventArgs e)
{
    //
    switch(e.KeyChar){
        case '\r':
            e.Handled = true;
            btnAdd.Focus();
            break;
        case '.':
        case ',':
        case '-':
            // si se pulsa en estos caracteres, se convertirá en /
            e.Handled = true;
            SendKeys.Send("/");
            break;
    }
}

```

En resumen:

e.Handled = true anula la pulsación de la tecla sobre el objeto

e.Handled = false deja que la pulsación se realice

32. ANEXO 9 - EJERCICIOS SIMPLES DE PROGRAMACIÓN EN “Visual C#”

- 1- Desarrollar un programa que evalúe el promedio de un alumno, si el promedio es de 0 – 10 DESAPROBADO, si esta entre 11 – 15 REGULAR, si esta en entre 16 – 18 BUENO y si está entre 19 y 20 EXCELENTE. Caso contrario Valor desconocido.

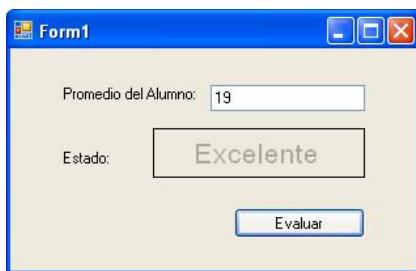
Para ello, crear un proyecto de tipo Aplicación Windows (Windows Form) y agregar los siguientes objetos/controles:

Objeto	Propiedad Name	Propiedad Text	Propiedad Enabled
Label1	Label1	Promedio del Alumno	
Label2	Label2	Estado	
Label3	lblestado		False
TextBox1	Txtprom		
Button1	btnevaluar	Evaluuar	

Para el evento click del botón Evaluuar:

```
private void btnevaluar_Click(object sender, EventArgs e)
{
    int prom = int.Parse(txtprom.Text); // Toma solo el texto ingresado
                                         // y lo convierte en int.
                                         // Ojo, no tiene control de errores

    if ( prom >= 0 && prom <=10 )
    {
        lblestado.Text = "Desaprobado";
    }
    else if ( prom >= 11 && prom <= 15 )
    {
        lblestado.Text = "Regular";
    }
    else if ( prom >= 16 && prom <= 18 )
    {
        lblestado.Text = "Bueno";
    }
    else if ( prom >= 19 && prom <= 20 )
    {
        lblestado.Text = "Excelente";
    }
    else
    {
        lblestado.Text = "Valor desconocido!!";
    }
}
```



Que aprendiste?:

Manejo de evento click sobre un objeto (en este caso un botón), manejo y uso de los valores almacenados en otros objetos (en este caso un Label y un TextBox. Uso y manejo de las propiedades de los abjetos.

- 2-** Desarrollar un programa que cargue un combobox con un listado de artículos varios y que al seleccionar uno de ellos de una breve descripción del mismo.

Para ello, crear un proyecto de tipo Aplicación Windows (Windows Form) y agregar los siguientes objetos/controles:

Objeto	Propiedad Name	Propiedad Text	Propiedad Enabled
Label1	Label1	Elegir Artículo	
Label2	Label2	Estado	
Label3	lblEstado		False
comboBox	comboBox1		

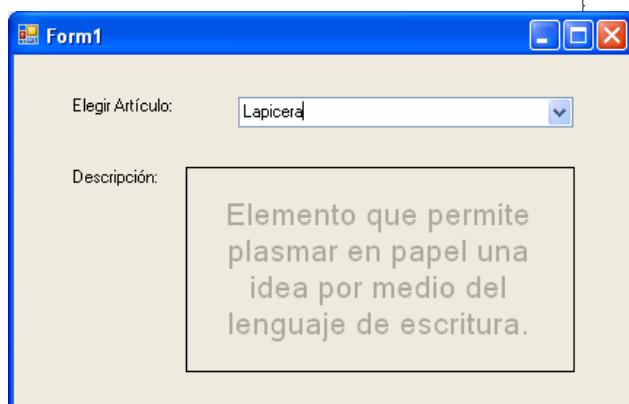
Para el load del formulario y el evento SelectedIndexChanged del ComboBox:

```
public Form1()
{
    InitializeComponent();

    comboBox1.Items.Add("Lapicera");
    comboBox1.Items.Add("Cuaderno");
    comboBox1.Items.Add("Teléfono");

}

private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (comboBox1.SelectedIndex.Equals(0) )
    {
        lbldescripcion.Text = "Elemento que permite plasmar en papel una idea por medio del lenguaje de escritura.";
    }
}
```



```
comboBox1.SelectedIndex.Equals(1))
{
    lbldescripcion.Text = "Determinada cantidad de hojas sujetadas entre otras.";

}
else if (comboBox1.SelectedIndex.Equals(2))
{
    lbldescripcion.Text = "Artefacto que permite comunicarse con otro dispositivo que contenga dicho artefacto.";
}
```

Que aprendiste?:

Manejo de evento Changed sobre un objeto (en este caso un combobox), manejo y uso de los valores almacenados en otros objetos (en este caso un Label). Uso y manejo de las propiedades de los abjetos. Carga de un combobox por código.

3- Desarrollar un programa que cargue una grilla con el resultado de una consulta a una base de datos.

Para ello, crear un proyecto de tipo Aplicación Windows (Windows Form) y agregar los siguientes objetos/controles:

Objeto	Propiedad Name	Propiedad Text	Propiedad Enabled
dataGridView	dataGridView1		

Para el load del formulario:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

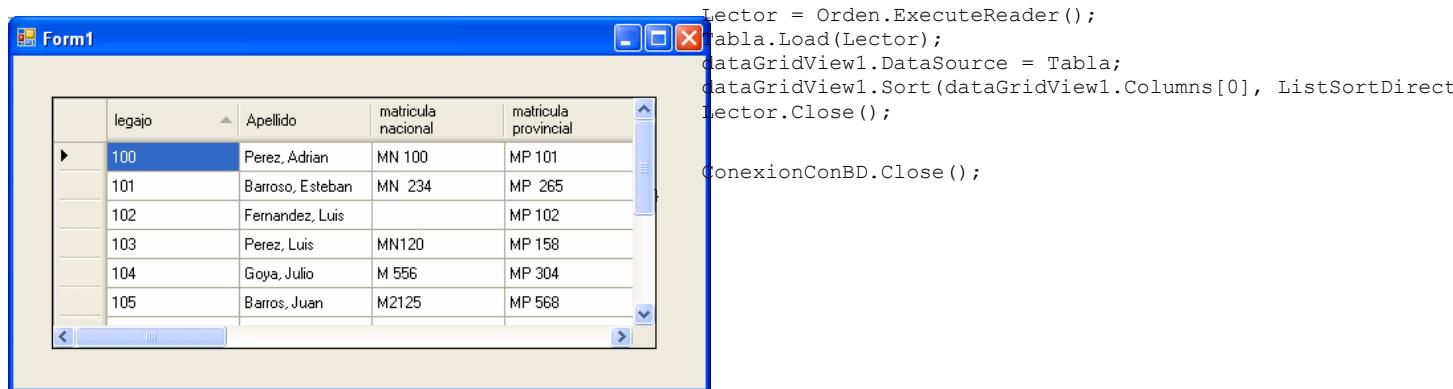
        private OleDbConnection ConexionConBD;
        private OleDbCommand Orden;
        private OleDbDataReader Lector;
        private string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" +
                                      "Data Source=c:\\\\clinica.mdb;";
        DataTable Tabla = new DataTable();
        private string Consulta;

        private void Form1_Load(object sender, EventArgs e)
        {
            ConexionConBD = new OleDbConnection(strConexión);
            Consulta = "SELECT * FROM legajo_profesionales";
            Orden = new OleDbCommand(Consulta, ConexionConBD);
            ConexionConBD.Open();

            Lector = Orden.ExecuteReader();
            Tabla.Load(Lector);
            dataGridView1.DataSource = Tabla;
            dataGridView1.Sort(dataGridView1.Columns[0], ListSortDirection.Ascending);
            Lector.Close();

            ConexionConBD.Close();
        }
    }
}

```



Que aprendiste?:

Carga de una grilla al iniciar el programa con los datos resultantes de una consulta directa a una Base de Datos.

4- Usando el desarrollo anterior agregar que al hacer clic en un registro de la grilla se carguen con los datos del mismo una serie de textbox debajo de dicha grilla.

Para ello, usar el proyecto anterior y agregar los siguientes objetos/controles:

Objeto	Propiedad Name	Propiedad Text	Propiedad Enabled
label1	label1	Legajo	
label2	label2	Apellido	
textBox1	textBox1		
textBox2	textBox2		

Para el evento click sobre la grilla:



```
private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
{
    string Legajo = Convert.ToString(dataGridView1.Rows[dataGridView1.CurrentCellAddress.Y].Cells[0].Value);
    textBox1.Text = Legajo;

    string Apellido = Convert.ToString(dataGridView1.Rows[dataGridView1.CurrentCellAddress.Y].Cells[1].Value);
    textBox2.Text = Apellido;
}
```

Que aprendiste?:

Carga de objetos con los datos cargados en una grilla al hacer clic sobre algún registro de la misma.

33. ANEXO 10 - Programa en C++ que genera Zudokus

```

#include <time.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>

int a[9][9], b[9][9];

bool Verif_cuadros(int grupo){
    int posic, i;
    bool acepto;

    acepto= true;
    for (posic=1; posic<10; posic++) {
        for (i=1; i<posic; i++) {
            if (b[grupo][posic]==b[grupo][i]) {
                acepto=false;
                goto sali;
            }
        }
    }
    sali: return acepto;
}

void Mezclar(){
    int restoX, restoY;
    int subir, t, x, y, i;
    float X, Y;
    int grupo, posic;
    bool repet, acepta;

    srand((unsigned)time(NULL));
    for (y=1; y<10; y++) {
        for (x=1; x<10; x++) {
            a[y][x]= (rand()%9) + 1;

            for (t=1; t<x; t++) {
                if (a[y][x]==a[y][t]) repet=true;
            }
            if (repet==false) {
                for (t=1; t<y; t++) {
                    if (a[y][x]==a[t][x]) repet=true;
                }
            }
            if (repet==true){
                repet=false;
                if (subir> 20){
                    subir= 0;
                    x= 0;
                }
                else {
                    subir++;
                    x--;
                }
            }
            else {
                restoY= y % 3;
                restoX= x % 3;
                if (restoX==0) restoX=3;
                if (restoY==0) restoY=3;
                Y= y; X= x;
                posic= 3 * (restoY-1) + restoX;
                grupo= 3 * (ceil(Y/3)-1) + ceil(X/3);
                b[grupo][posic]= a[y][x];
                subir= 0;
            }
        }
    }

    if (restoY==3){
        for (i=0; i<3; i++){
            grupo= y - i;
            acepta= Verif_cuadros(grupo);
            if (acepta==false) {
                y = y - 3;
                goto sali;
            }
        }
    }
}

```

```
sali:;
}
void Imprimir() {
    int y, x;
    for (y=1; y<10; y++) {
        for (x=1; x<10; x++) {
            cout << a[y][x] << " ";
        }
        cout << endl;
    }
}
void main() {
    Mezclar();
    Imprimir();
}
```

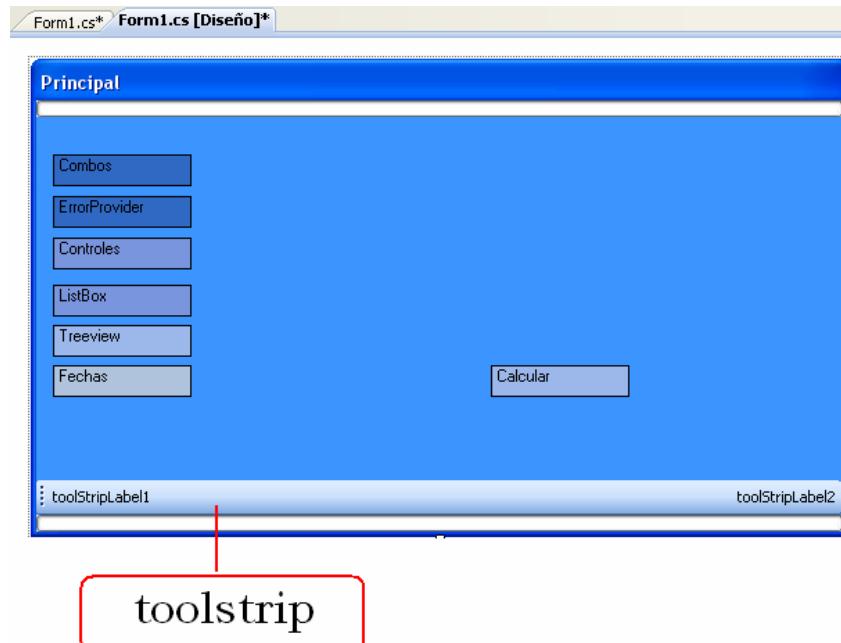
34. Ejemplos Básicos de programación en windows form

Capítulo elaborado por: Ruben Moruzzi

En esta aplicación se encuentran 7 ejemplos con uso de diferentes controles.

ToolStrip:

En el primer formulario se encuentra ubicado en la parte inferior un toolStrip, donde se pueden insertar etiquetas, botones, imágenes, etc.



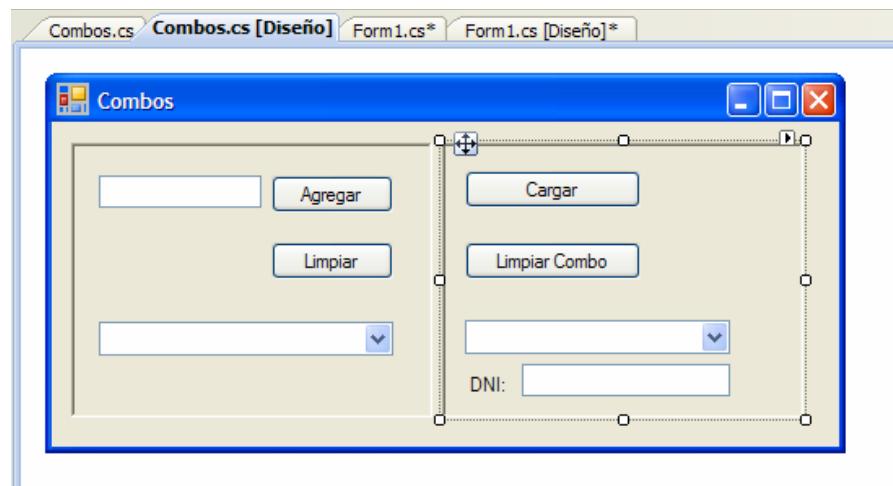
En este caso solo utilice dos etiquetas. Una contiene la fecha actual, y la otra, que actualizándose con un timer, muestra la hora.



- 1- En el load del form modifica el texto del ítem de la posición “0” del toolStrip1, y le asigna la fecha del sistema con el formato “dd/MM/yyyy” (día, mes y año).
En el mismo evento habilita el timer.
- 2- En el evento Tick del timer se le asigna al ítem de la posición “1” del toolStrip la hora del sistema.
2-1 Para que el timer debe tener un tiempo de vuelta de 1 seg o menor para poder ver el cambio cada segundo.

2-2 La hora del sistema llega con horas, minutos, segundos y milésimas de segundo, por lo cual, para tomar solo hasta los segundos, se debe tomar el substring hasta la posición 8.

Combobox:



En el primer panel se encuentran:

- 1 textbox
- 2 botones
- 1 combobox

En el textbox se escribe el elemento a agregar al combo.

Al hacer clic en agregar se ejecuta el siguiente código.

```
private void button1_Click(object sender, EventArgs e)
{
    CboManual.Items.Add(textBox1.Text);
}
```

La función de este es agregar un nuevo ítem al combo.

El segundo botón (“Limpiar”) tiene como función limpiar el combo con el siguiente código.

```
private void button2_Click(object sender, EventArgs e)
{
    CboManual.Items.Clear();
}
```

En el segundo panel:

- 1 textbox
- 2 botones (“Cargar” y “Limpiar Combo”)
- 1 combobox

Al hacer clic en el botón “Cargar” se ejecuta el siguiente código.

```

private void button3_Click(object sender, EventArgs e)
{
    Tabla = new DataTable();
    Conexion = new OleDbConnection(StrConexion);
    Consulta = "Select * From Personas";
    Orden = new OleDbCommand(Consulta, Conexion);
    Conexion.Open();
    Lector= Orden.ExecuteReader();
    Tabla.Load(Lector);
    CboBase.DataSource= Tabla;
    CboBase.DisplayMember = "Apellido";
    CboBase.ValueMember= "Dni";
    Conexion.Close();
}

```

Paso a paso:

- 1- Instancia `DataTable` Tabla.
- 2- Establece la conexión de la base de datos en la variable `Conexion`.
- 3- Se carga en la variable `Consulta` la consulta correspondiente.
- 4- Se establece en `Orden` la consulta a ejecutar y la conexión con la que va a trabajar.
- 5- Se abre la conexión.
- 6- Se carga en `Lector` la ejecución de la Orden.
- 7- Se carga la tabla con el resultado de la lectura a la base.
- 8- La fuente de datos del combo es Tabla.
- 9- Establece que campo va a mostrar en el combo.
- 10- Establece el valor (puede usarse como ID) de cada ítem que se carga en el combo.
- 11- Se cierra la conexión.

Al hacer clic en botón “Limpiar Combo” se ejecuta el siguiente código.

```

private void button4_Click(object sender, EventArgs e)
{
    Tabla.Clear();
    CboBase.DataSource = Tabla;
}

```

- 1- Limpia el contenido de la Tabla.
- 2- Asigna al combo el contenido de tabla.

En el evento `SelectedIndexChanged` del combo se ejecuta el siguiente código.

```

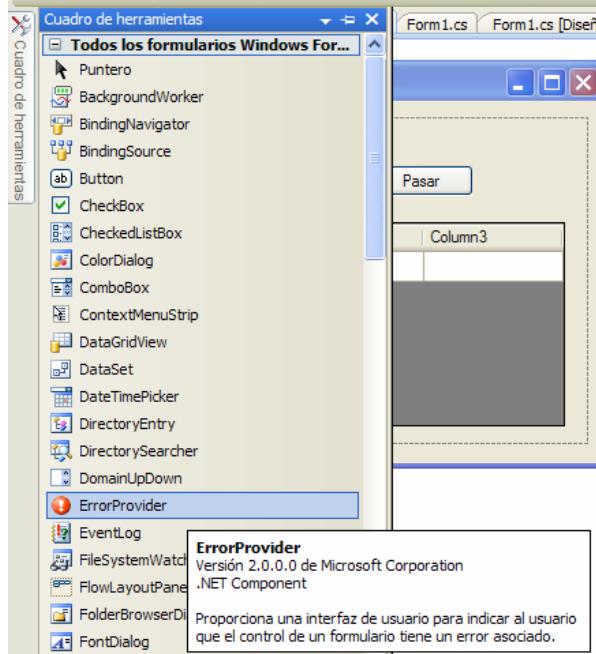
private void CboBase_SelectedIndexChanged(object sender, EventArgs e)
{
    try
    {
        if (CboBase.SelectedValue.ToString() != "System.Data.DataRowView")
            TxtDni.Text = CboBase.SelectedValue.ToString();
    }
    catch
    {
        TxtDni.Text = "";
    }
}

```

Si el texto del valor seleccionado es distinto de “`System.Data.DataRowView`”, le asigna al textbox el valor seleccionado del combo.

Si alguna de las líneas anteriores devuelve un error se vacía el textbox.

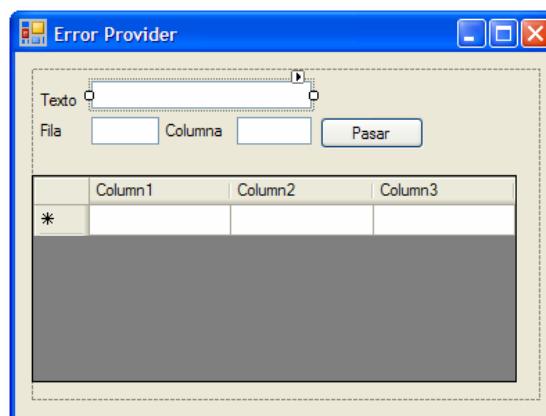
Error provider:



Proporciona una interfaz de usuario para indicar que un control de un formulario tiene un error asociado.

En este ejemplo se utilizan:

- 3 textbox
- 1 botón
- 1 Grilla
- 1 Panel



En el textbox “Texto” se escribe lo que se va a agregar en la celda que se indicará.

En el textbox “Fila” se agrega el número de fila que se desea utilizar.

En el textbox “Columna” se agrega el número de columna que se desea utilizar.

La función del Botón es la siguiente:

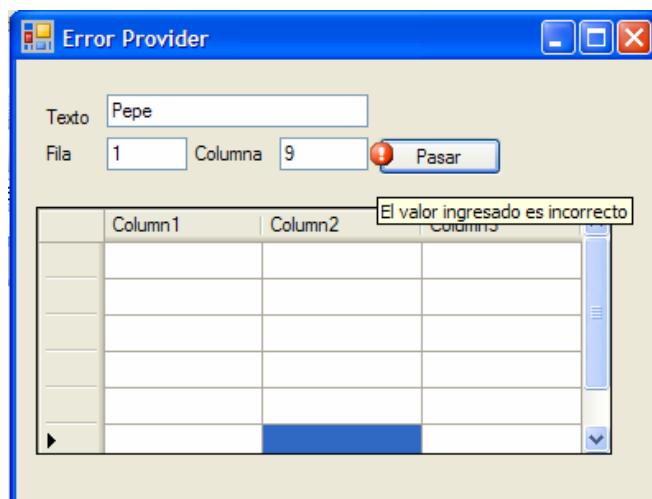
```

private void button1_Click(object sender, EventArgs e)
{
    errorProvider1.Clear();
    try
    {
        dataGridView1[Convert.ToInt32(textBox2.Text),
                     Convert.ToInt32(textBox3.Text)].Value = textBox1.Text;
    }
    catch
    {
        if ((Convert.ToInt32(textBox2.Text) >= dataGridView1.Rows.Count) ||
            (Convert.ToInt32(textBox2.Text) < 0))
            errorProvider1.SetError(textBox2, "El valor ingresado es incorrecto");
        if ((Convert.ToInt32(textBox3.Text) >= dataGridView1.Columns.Count) ||
            (Convert.ToInt32(textBox3.Text) < 0))
            errorProvider1.SetError(textBox3, "El valor ingresado es incorrecto");
    }
}

```

Paso a paso:

- 1- Se limpia el control (“ErrorProvider”).
- 2- Intenta agregar el texto antes mencionado en la fila y columna indicadas.
- 3-Si no se puede ejecutar el código anterior se verifica cual es el textbox que genera el error.
- 4- Se asigna el “ErrorProvider” al textbox correspondiente indicando que el valor ingresado es inválido.



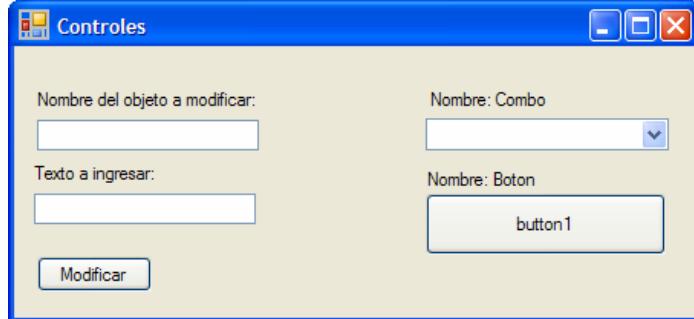
En el caso anterior se presenta una grilla con 6 filas y 3 columnas. Se intenta asignar “Pepe” en la fila 1 y columna 9, por lo tanto se muestra el ícono del ErrorProvider junto al textbox que genera el error.

Controles:

En el siguiente caso se muestra como identificar los controles de un formulario y modificar alguna de sus propiedades.

Está constituido por:

- 1 combobox
- 2 botones
- 2 textbox



Atención: las propiedades que se pueden modificar son limitadas, ya que no se interactúa directamente con el control, sino que se interactúa a través del formulario. Por lo tanto las propiedades que se pueden modificar son las que tienen en común la gran mayoría de los controles del formulario.

Paso a paso:

1- En el textbox “Nombre del objeto a modificar”, como la etiqueta lo dice, se ingresa el nombre del objeto que se desea modificar.

Los nombres de los objetos están en su etiqueta (label).

2- En el textbox “Texto a ingresar” se ingresa el texto que se desea mostrar en el objeto.

3- En el clic del botón “Modificar” se ejecuta el siguiente código:

```
private void BtCargar_Click(object sender, EventArgs e)
{
    for (int i = 0; i < this.Controls.Count; i++)
    {
        if (this.Controls[i].Name == textBox1.Text)
            this.Controls[i].Text = textBox2.Text;
    }
}
```

3.1- Se recorren los controles del formulario con un **for** de $i=0$ hasta la cantidad de objetos que posee el formulario.

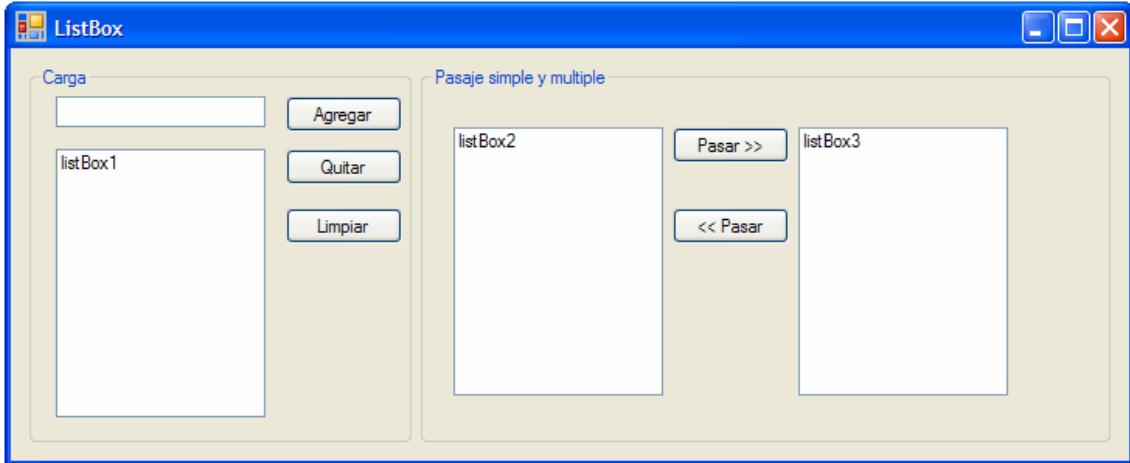
3.2- Si el nombre del objeto seleccionado es igual al textbox “Nombre del objeto a modificar”, le asigna a ese objeto el texto ingresado en el segundo textbox.

Listbox:

En este ejemplo se mostrará como agragar y pasar ítems entre Listbox.

Se utilizan:

- 1 textbox
- 3 listbox
- 5 botones
- 2 panel



En el primer panel se muestra la carga de un listBox:

En el textBox se escribe el nuevo ítem y se hace clic en agregar para insertar ese nuevo ítem al listBox:

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Add(textBox1.Text);
}
```

Una vez agregado el/los ítems se puede seleccionar uno y dar clic al botón “Quitar” para eliminar el ítem seleccionado:

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        listBox1.Items.RemoveAt(listBox1.SelectedIndex);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Debe seleccionar un item", "Error");
    }
}
```

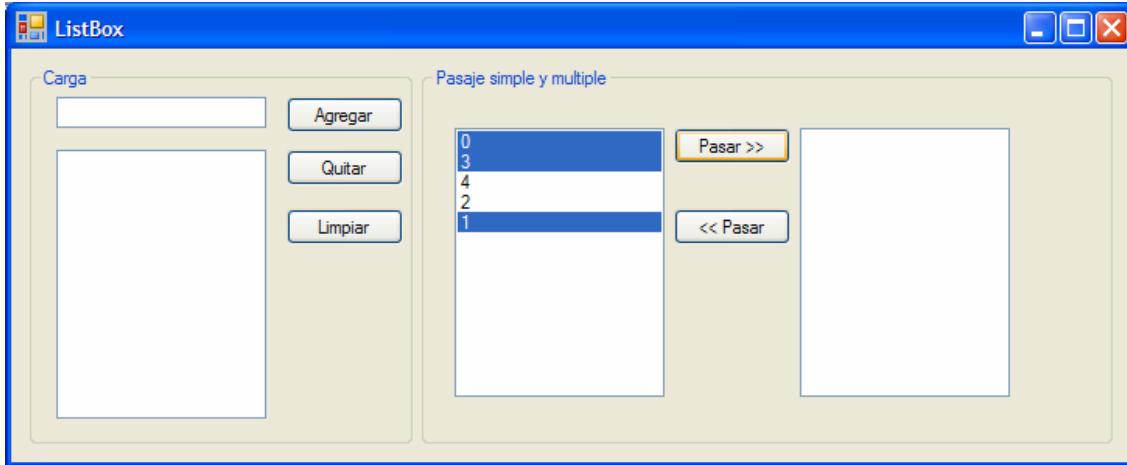
También se puede limpiar el listBox haciendo clic en el Boton “Limpiar”:

```
private void button3_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
}
```

En el segundo panel se muestra un listBox cargado previamente y uno vacío para poder efectuar el traspaso entre listBox.

Del listBox1 al 2 se puede hacer un traspaso múltiple, en cambio del 2º al primero se puede hacer solo el traspaso simple.

Elegir en el primer listBox los ítems a pasar y hacer clic en el botón “Pasar >>”:



```

private void button4_Click(object sender, EventArgs e)
{
    try
    {

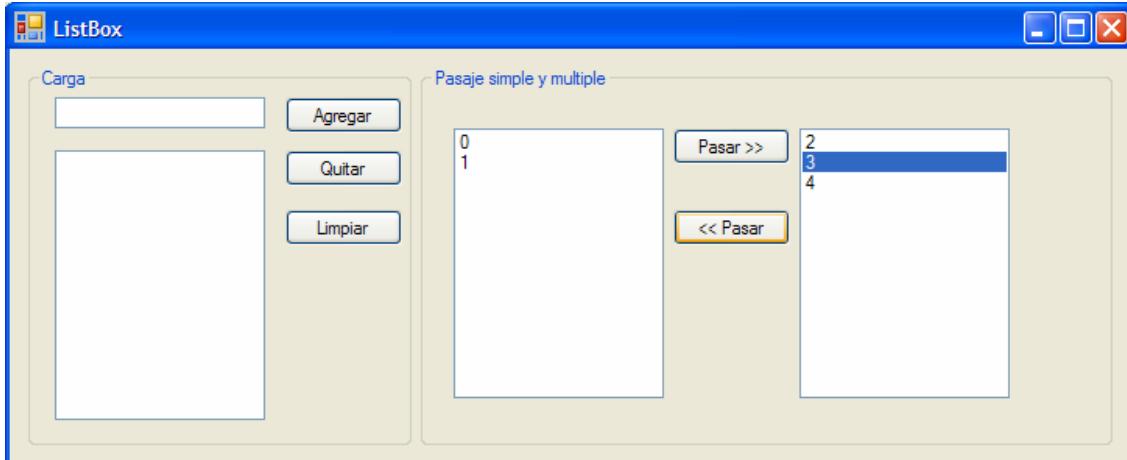
        //listBox3.Items.Add(listBox2.SelectedItem.ToString());
        while (listBox2.SelectedItems.Count > 0)
        {
            listBox3.Items.Add(listBox2.SelectedItem.ToString());
            listBox2.Items.Remove(listBox2.SelectedItem);
        }
    }
    catch
    {
        MessageBox.Show("Debe seleccionar un ítem", "Error");
    }
}

```

Paso a paso:

- 1- Mientras haya seleccionado al menos un ítem.
- 2- Se agrega el otro listbox uno de los ítems seleccionados.
- 3- Se borra el ítem pasado.

Del listbox2 al 1:



Se selecciona el ítem que se desea pasar y se hace clic en el botón "<< Pasar":

```

private void button5_Click(object sender, EventArgs e)
{
    try
    {
        listBox2.Items.Add(listBox3.SelectedItem.ToString());
        listBox3.Items.RemoveAt(listBox3.SelectedIndex);
    }
    catch
    {
        MessageBox.Show("Debe seleccionar un ítem", "Error");
    }
}

```

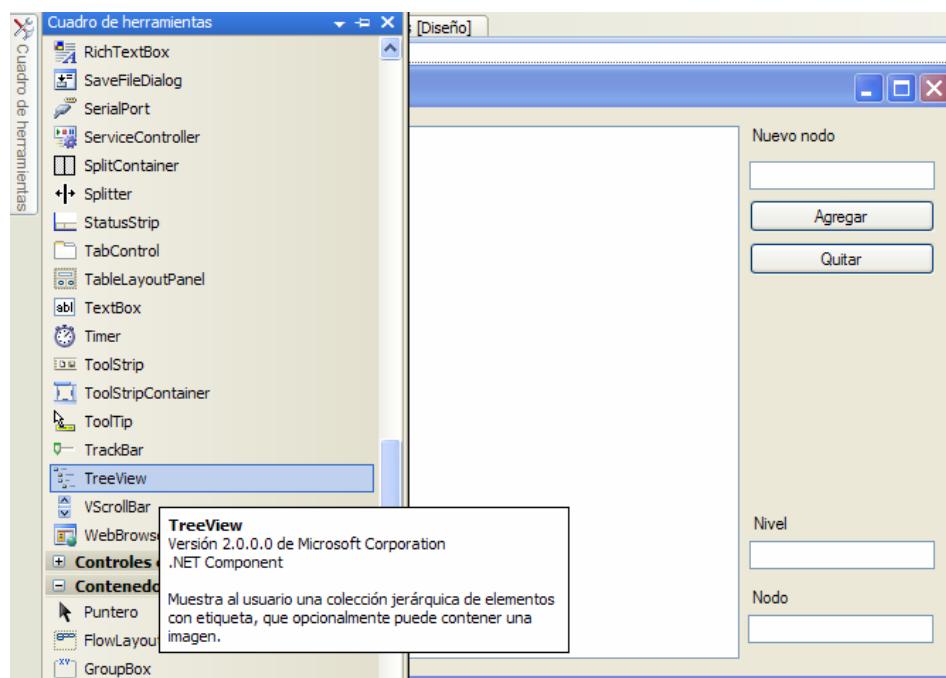
La acción es similar a la del botón anterior pero sin el `while`, porque solo se pasa un ítem.

TreeView:

Tree view (Vista de árbol) es un elemento de interfaz gráfica que presenta una vista jerárquica de información. Cada elemento (a menudo llamado rama o nodo) puede tener una serie de sub elementos. Esta puede ser visualizada como tabulado en una lista. Un elemento puede revelar sub elementos.

Se utilizan:

- 1 TreeView
- 3 textBox
- 2 botones



En este ejemplo vamos a agregar nodo por nodo.

Ingresar el nombre del nodo en el textBox “Nuevo nodo” y hacer clic en agregar:

```

private void button1_Click(object sender, EventArgs e)
{
    try
    {
        treeView1.SelectedNode.Nodes.Add(textBox1.Text);
    }
    catch
    {
        treeView1.Nodes.Add(textBox1.Text);
    }
}

```

Agrega un nodo nuevo en el nodo seleccionado con el texto que se ha ingresado en el textbox "Nuevo nodo".

Si no hay un nodo seleccionado agrega un nodo raíz.

Clic en el botón "Quitar":

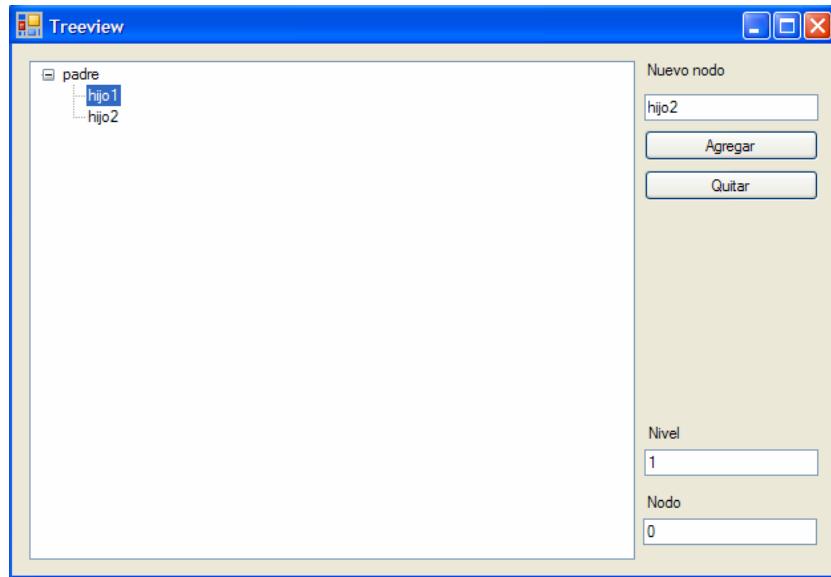
```

private void button2_Click(object sender, EventArgs e)
{
    treeView1.SelectedNode.Remove();
}

```

Quita el nodo seleccionado y todos los subnodos que posee.

Al selecciona un nodo en el treeview:



```

private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    txtnivel.Text = treeView1.SelectedNode.Level.ToString();
    txt nodo.Text = treeView1.SelectedNode.Index.ToString();
}

```

Muestra en el textbox "Nivel" el nivel (profundidad) del nodo seleccionado.

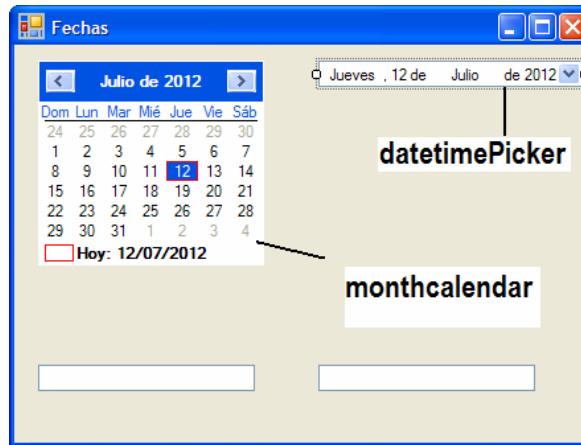
En el textbox "Nodo" muestra el índice entre sus "hermanos"(subnodos del mismo nodo).

Fechas:

En esta sección se mostrara dos controles para interactuar con fechas.

Se utiliza:

- 1 MonthCalendar
- 1 dateTimePicker
- 2 textbox



Lo que se hace en este ejemplo es solo seleccionar una fecha para ver de las dos formas.

En el evento monthCalendar1_DateChanged:

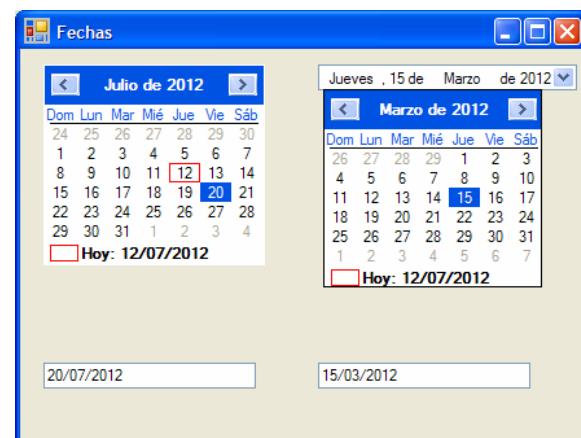
```
private void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
{
    textBox1.Text = monthCalendar1.SelectionStart.ToString("dd/MM/yyyy");
}
```

Toma la selección del calendario y le da formato "dd/MM/yyyy" para asignarla al textbox.

En el evento dateTimePicker1_ValueChanged:

```
private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
{
    textBox2.Text = dateTimePicker1.Value.ToString("dd/MM/yyyy");
}
```

Toma el valor para darle formato y asignarlo al textbox:



35. Ejemplo completo ABMC de una Base de Datos Acces

=====
Preparado por : Ing. Leandro A. Pini
Para: Segundo año de la Carrera de Sistemas del ISFDyT N°93
Año: 2012
Motivo/Finalidad: ejemplificar un ABMC de una Base de Datos Acces
=====

Ayuda a problemas frecuentes:

Tener en cuenta que es solo un ejemplo, no contempla:

- validación de datos ingresados.
- claves primarias, secundarias.
- relaciones entre tablas.
- etc.

PCs con OFFICE 2000/2003:

Si te da error al leer MSysObjects cuando ejecutas una consulta SQL:

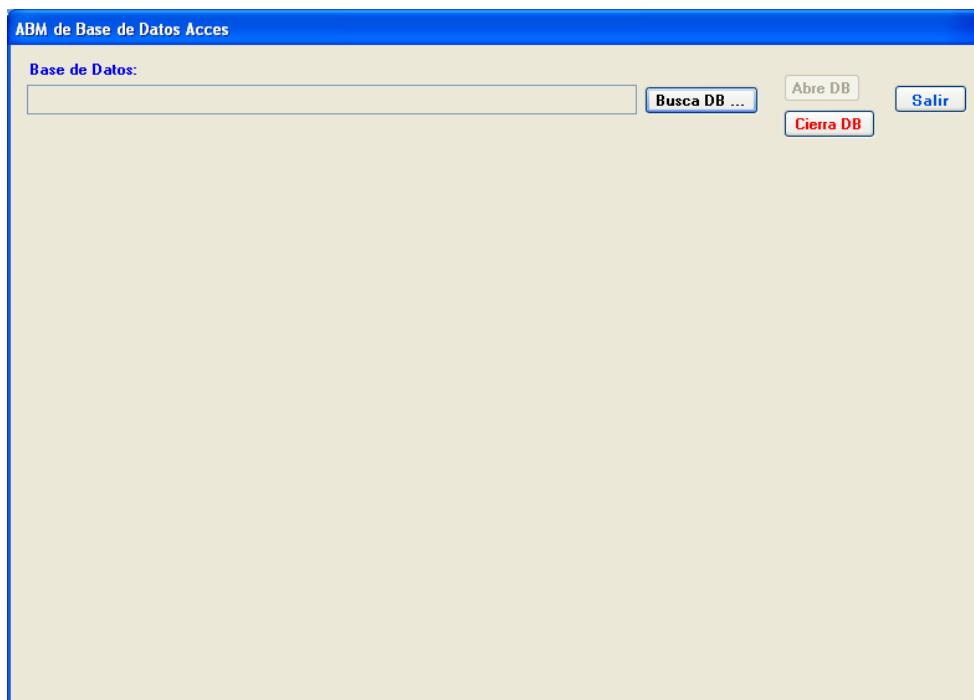
Tenés que cambiarle las propiedades a la tabla. Para hacer eso tenés que abrir la base de datos, ir a Herramientas > Seguridad > Permisos de Usuario y de Grupo. En la solapa Permisos, seleccionar en el combo Tipo de Objeto: Tabla. En la lista de arriba, tenés que seleccionar MSysObjects y tilda la opción Leer Datos.

Si no te aparece la tabla MSysObjects en la lista, previamente deberás ir a Herramientas > Opciones. En la solapa Ver tenés que tildar los checks de Objetos Ocultos y de Objetos de Sistema.

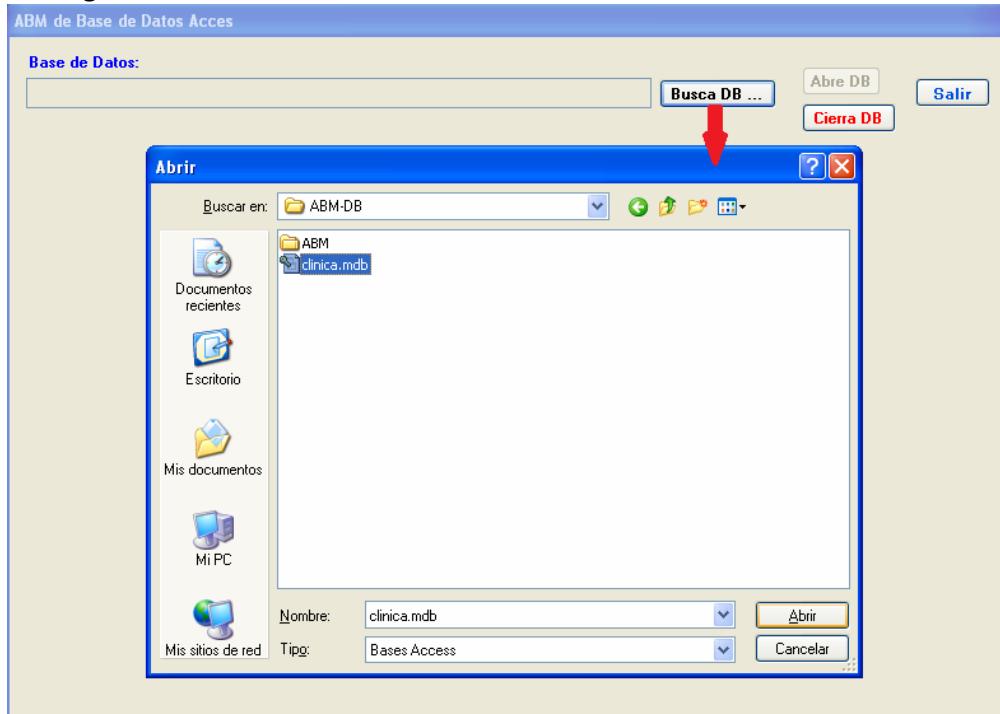
PCs con OFFICE 2007:

- 1- Haciendo click en el botón de Office y luego en el botón de Opciones de Access.
- 2- En las Opciones de Access seleccionamos el Apartado Base de Datos Actual y Buscamos Las opciones de Exploración, allí hacemos click en el botón Opciones de Exploración.
- 3- En la siguiente pantalla debes activar las casillas “Mostrar Objetos Ocultos” y “Mostrar Objetos del Sistema”.

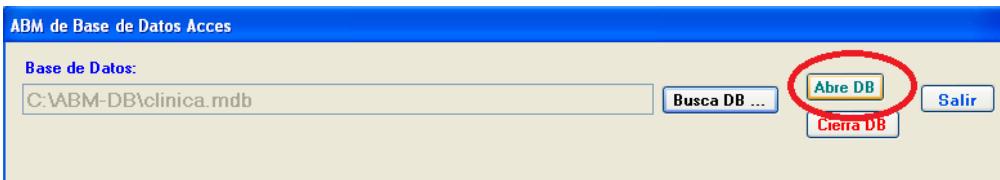
Inicio del Sistema/Ejemplo ABM:



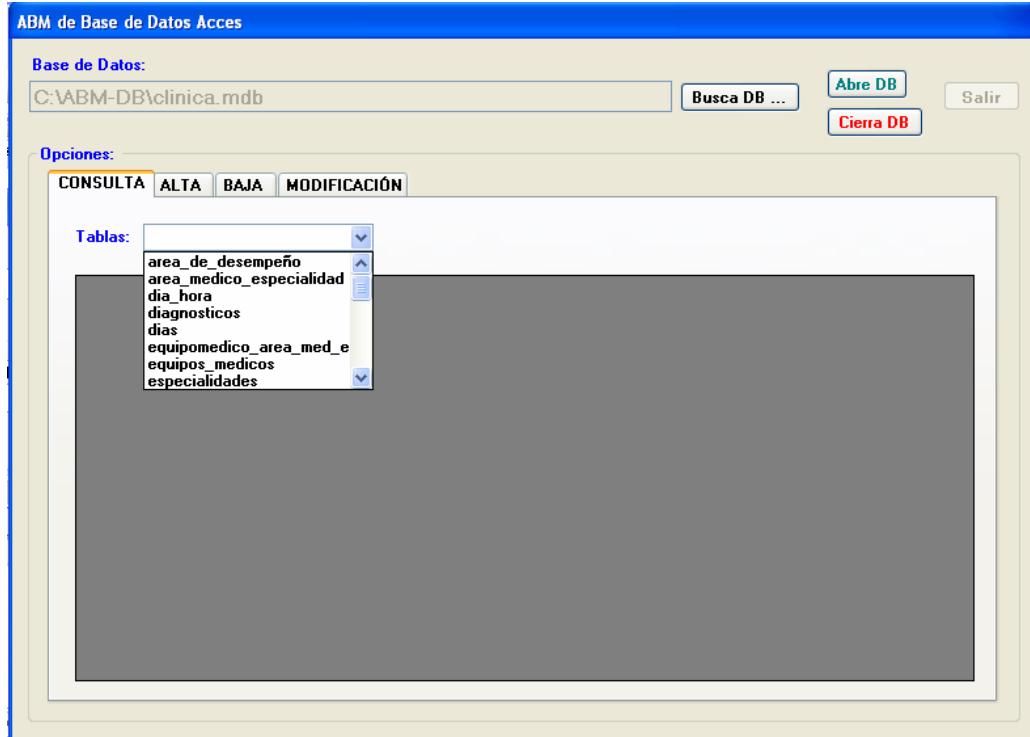
Para poder abrir una Base Datos Acces lo que se debe hacer es click en el botón que dice “Buscar DB ...” tal como se muestra la siguiente imagen:



Una vez seleccionada la DB se hace click en el botón que dice “Abre DB”:



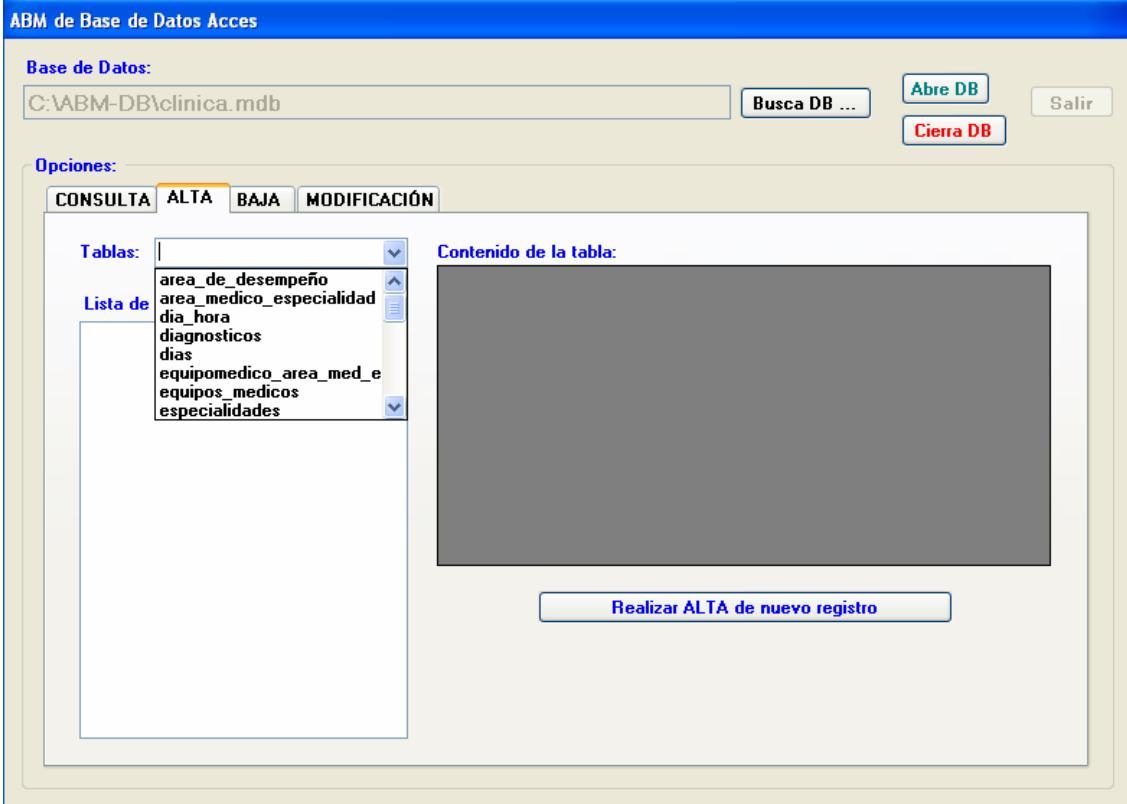
Una vez abierta la DB está listo para usarse, primero tenemos la solapa “CONSULTA”, en donde podemos ver la lista de tablas que hay dentro de la DB:



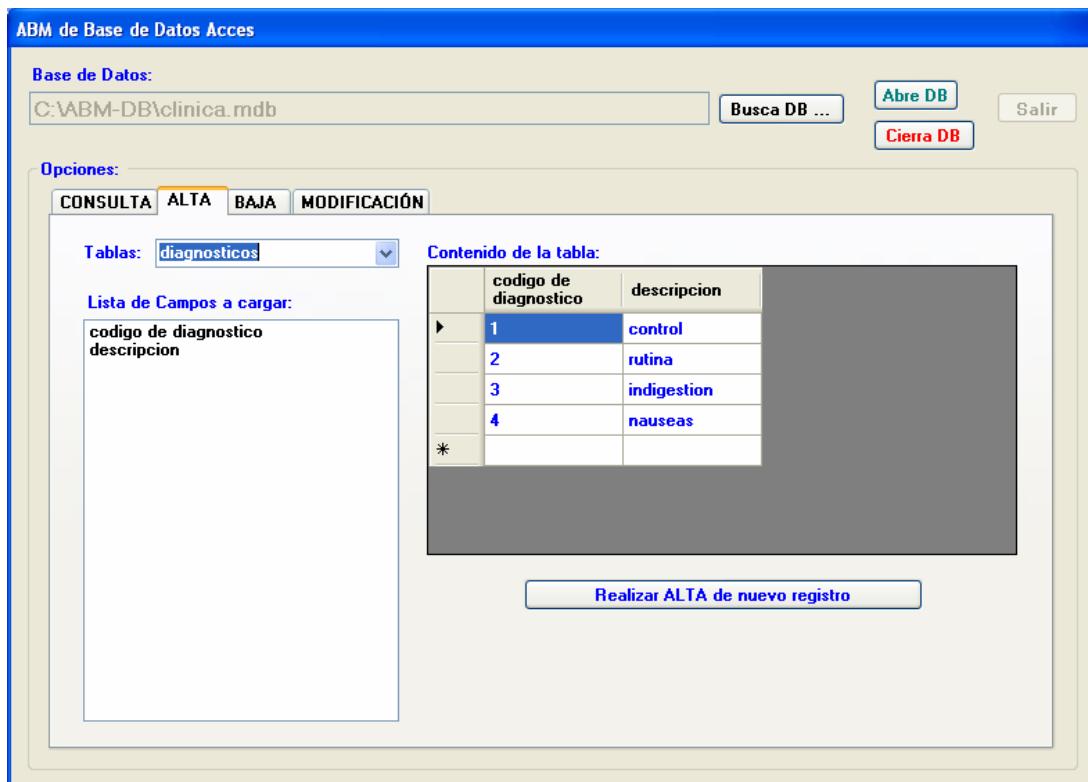
Por cada tabla que seleccionemos en el combo veremos su contenido:

C:\VABM-DB\clinica.mdb																											
Opciones:																											
CONSULTA ALTA BAJA MODIFICACIÓN																											
Tablas: equipos_medicos																											
<table border="1"> <thead> <tr> <th></th> <th>codigo de equipo</th> <th>descripcion</th> <th>diagnosticos en los que acrua</th> </tr> </thead> <tbody> <tr> <td>▶</td> <td>1</td> <td>estandar</td> <td>2</td> </tr> <tr> <td></td> <td>2</td> <td>operaciones rutinarias</td> <td>3</td> </tr> <tr> <td></td> <td>3</td> <td>cirujia mayor</td> <td>4</td> </tr> <tr> <td></td> <td>4</td> <td>cirujia menor</td> <td>1</td> </tr> <tr> <td>*</td> <td>5</td> <td>extraordinario</td> <td>8</td> </tr> </tbody> </table>					codigo de equipo	descripcion	diagnosticos en los que acrua	▶	1	estandar	2		2	operaciones rutinarias	3		3	cirujia mayor	4		4	cirujia menor	1	*	5	extraordinario	8
	codigo de equipo	descripcion	diagnosticos en los que acrua																								
▶	1	estandar	2																								
	2	operaciones rutinarias	3																								
	3	cirujia mayor	4																								
	4	cirujia menor	1																								
*	5	extraordinario	8																								

En la solapa de “ALTA” veran algo similar solo que por cada tabla que eligan veran los campos y podras ingresar un nuevo registro:



Una vez elegida la tabla para poder hacer un alta solo se hace click en el botón “Realizar ALTA de nuevo registro” y verán que debajo del mismo el programa comenzará a pedirles los valores por cada campo de la tabla. Tengan en cuenta los posibles errores o comentarios que se detallaron a principio de este documento:



Primer campo:

3	indigestion
4	nauseas
*	

Realizar ALTA de nuevo registro

Cargando campo:

descripcion **Siguiente**

Segundo campo:

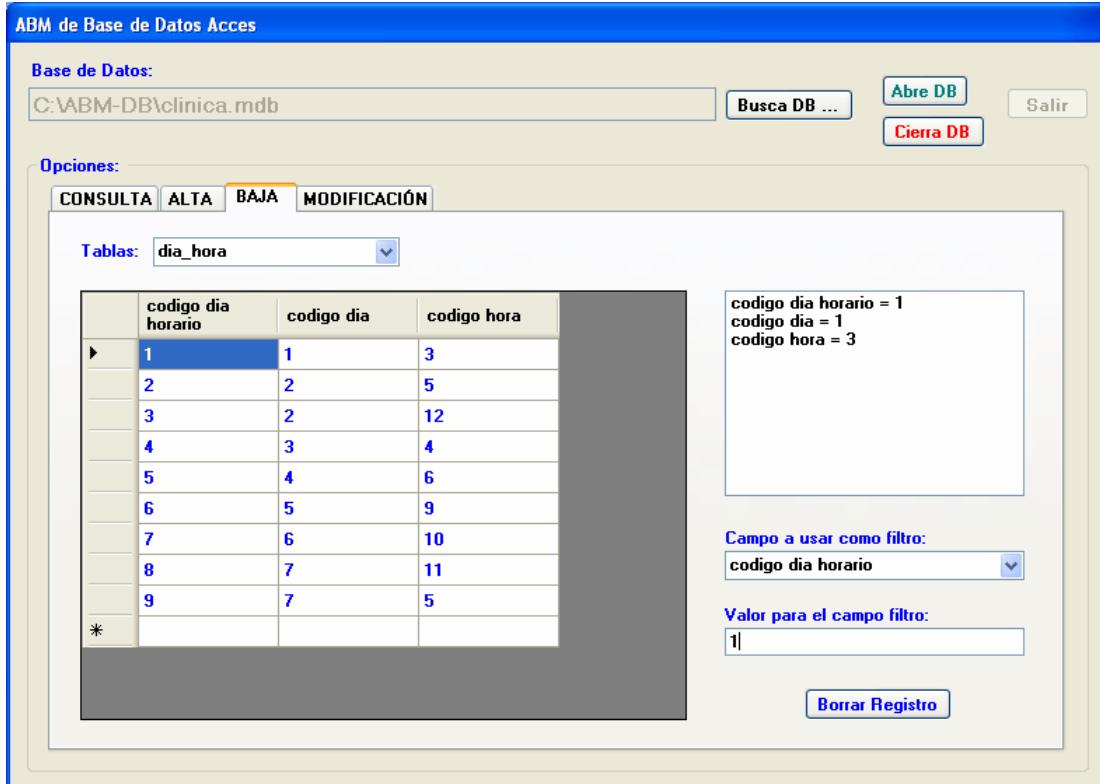
*	
---	--

Realizar ALTA de nuevo registro

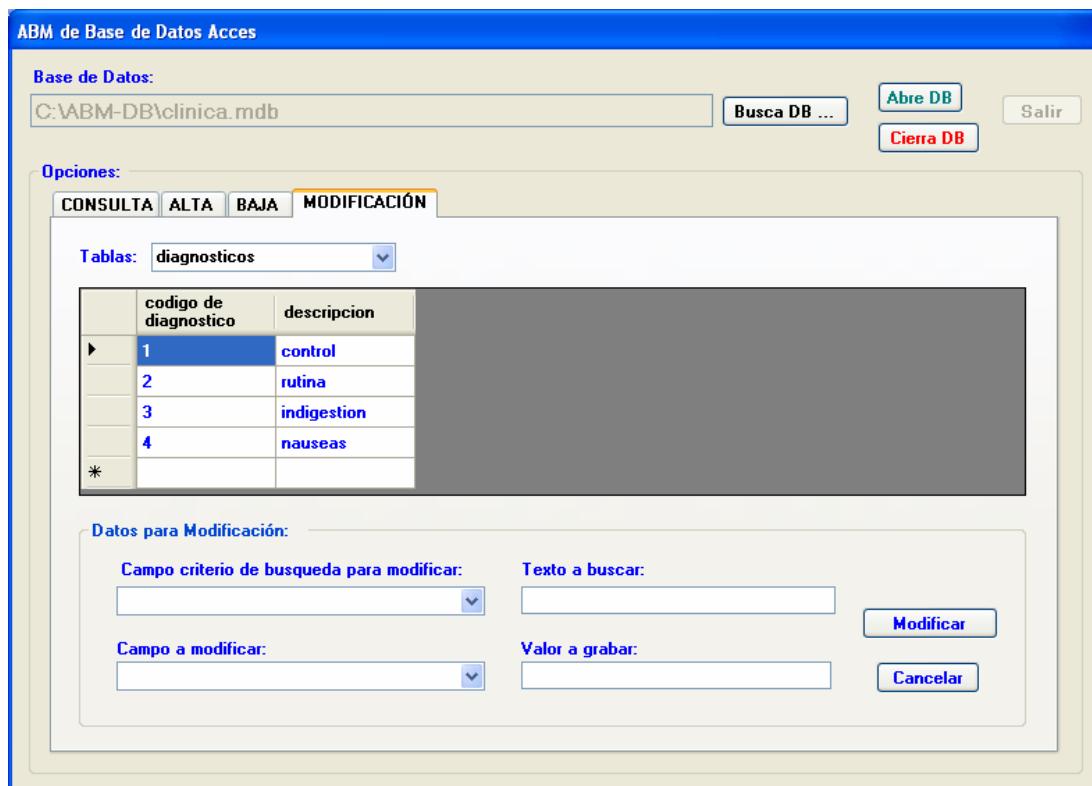
Cargando campo:

codigo de diagnostico **Siguiente**

En la solapa “BAJA” tenemos un esquema en donde vemos las tablas de la DB dentro del ComboBox, por cada tabla que se elige se ve su contenido. Al hacer click en alguno de los registros veremos en un ListBox los campos y sus valores de dicho registro, eso nos sirve como guía para colocar el valor y campo correspondiente del o los registros a ser borrados:



En la solapa “MODIFICACIÓN” veran un esquema similar, pero encontrarán nuevos objetos que servirán para realizar el borrado de los registros deseados según se completen los mismos:



Tareas:

- 1- Antes de comenzar a tocar y/o modificar el programa deberán entender, interpretar y razonar su funcionamiento. Para ello veremos el código que hay detrás de cada objeto en clase y ante cualquier duda levanta tu mano y realiza tu consulta.**

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
using System.Threading;

namespace ABM
{
    public partial class Principal : Form
    {
        private OleDbConnection ConexionConBD;
        private OleDbCommand Orden;
        private OleDbDataReader Lector;
        private OleDbCommand Escribe;
        string SQL_ALTA = "";
        int campo = 0;
        int i = 0;

        public Principal()
        {
            InitializeComponent();
        }

        private void Buscar_Click(object sender, EventArgs e)
        {
            //Este Botón solo es para ubicar la base de datos a abrir y pasar el path y el
            nombre de la misma al
            //al text box cuyo contenido ser{a usado como datos para abrir dicha DB.

            openFileDialog1.Filter = "Bases Access (*.mdb";
            openFileDialog1.InitialDirectory = "C:\\\\";
            openFileDialog1.ShowDialog();
            textBox1.Text = openFileDialog1.FileName.ToString();
            if (openFileDialog1.FileName.ToString() == "openFileDialog1")
                textBox1.Text = "";
            if ((textBox1.Text != null) & (textBox1.Text != ""))
                Abrir.Enabled = true;
        }

        private void Abrir_Click(object sender, EventArgs e)
        {
            // Este botón abre la DB que se seleccionó y precarga los objetos cuyos datos no
            varian.

            string strConexión = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=" +
            textBox1.Text + ";";
            ConexionConBD = new OleDbConnection(strConexión);
            ConexionConBD.Open();
            MessageBox.Show("Base de datos abierta!!!");
            grupo.Visible = true;

            // Carga de datos en los abjetos
            //=====

            // Carga de los combos que contienen la lista de Tablas
            string consulta = "SELECT MSysObjects.Name AS Tabla FROM MsysObjects WHERE
            (Left$([Name],1)<>'~') AND (Left$([Name],4) <> 'Msys') AND (MSysObjects.Type)=1 ORDER BY
            MSysObjects.Name;";
            Orden = new OleDbCommand(consulta, ConexionConBD);
            Lector = Orden.ExecuteReader();
            while (Lector.Read())
            {
                cmb_tablas.Items.Add(Lector["Tabla"]);
                cmb_tablas_alta.Items.Add(Lector["Tabla"]);
                cmb_tabla_baja.Items.Add(Lector["Tabla"]);
                cmb_tabla_modif.Items.Add(Lector["Tabla"]);
            }
            Lector.Close();
            //=====
        }
    }
}

```

```

        button1.Enabled = false;
    }

    private void Cerrar_Click(object sender, EventArgs e)
    {
        // Este botón cierra la conexión con la DB.

        if (Lector != null)
            Lector.Close();

        if (ConexionConBD != null)
        {
            ConexionConBD.Close();
            MessageBox.Show("Base de datos cerrada!!!");
        }

        if (ConexionConBD == null)
        {
            MessageBox.Show("No hay ninguna base abierta!!!");
        }

        grupo.Visible = false;
        textBox1.Text = "";
        button1.Enabled = true;
    }

    private void cmb_tablas_SelectedIndexChanged(object sender, EventArgs e)
    {
        // Según la tabla seleccionada cargo la grilla con todos sus registros
        string consulta = "Select * from " + cmb_tablas.Text;
        Orden = new OleDbCommand(consulta, ConexionConBD);
        Lector = Orden.ExecuteReader();
        DataTable dt = new DataTable();
        dt.Load(Lector);
        registros.DataSource = dt;
        Lector.Close();
    }

    private void cmb_tablas_alta_SelectedIndexChanged(object sender, EventArgs e)
    {
        // Según la tabla seleccionada cargo el listbox con todos sus campos
        string consulta = "Select * From " + cmb_tablas_alta.Text;
        Orden = new OleDbCommand(consulta, ConexionConBD);
        Lector = Orden.ExecuteReader();
        list_campos_alta.Items.Clear();
        DataTable dt = new DataTable();
        dt.Load(Lector);
        dataGridView1.DataSource = dt;
        label7.Text = Convert.ToString(dataGridView1.Columns.Count);
        campo = ((Convert.ToInt32(label7.Text.Trim())) - 1);

        //Armo consulta de ALTA
        SQL_ALTA = "INSERT INTO " + cmb_tablas_alta.Text + "(";
        for (i = ((dataGridView1.Columns.Count) - 1); i >= 0; i--)
        {
            SQL_ALTA = SQL_ALTA + dataGridView1.Columns[i].Name;
            if (i > 0)
                SQL_ALTA = SQL_ALTA + ", ";
            else
                SQL_ALTA = SQL_ALTA + ") VALUES(";
        }

        // Cargo la lista con los nombres de los campos de la tabla
        for (i = 0; i < dataGridView1.Columns.Count; i++)
        {
            list_campos_alta.Items.Add(dataGridView1.Columns[i].Name);
        }
        Lector.Close();
    }

    private void botón_hacer_alta_Click(object sender, EventArgs e)
    {
        groupBox1.Visible = true;
        cmb_tablas_alta.Enabled = false;
        label6.Text = dataGridView1.Columns[campo].Name;
        campo = campo - 1;
        textBox2.Text = "";
    }
}

```

```

private void alta_siguiente_Click(object sender, EventArgs e)
{
    if (campo > 0)
        SQL_ALTA = SQL_ALTA + textBox2.Text + ", ";
    if (campo == 0)
        SQL_ALTA = SQL_ALTA + textBox2.Text;
    if (campo >= 0)
    {
        label6.Text = dataGridView1.Columns[campo].Name;
    }
    else
    {
        groupBox1.Visible = false;
        cmb_tablas_alta.Enabled = true;
        SQL_ALTA = SQL_ALTA + ", " + textBox2.Text + ");";
        MessageBox.Show(SQL_ALTA, "Se ejecuta:");
        label7.Text = Convert.ToString(dataGridView1.Columns.Count);
        campo = ((Convert.ToInt32(label7.Text.Trim()) - 1));
    }
}

// Ejecuto el SQL_ALTA
Orden = new OleDbCommand(SQL_ALTA, ConexionConBD);

try
{
    Orden.ExecuteNonQuery();

    // Recargo la grilla para ver el nuevo registro cargado
    string consulta = "Select * From " + cmb_tablas_alta.Text;
    Orden = new OleDbCommand(consulta, ConexionConBD);
    Lector = Orden.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(Lector);
    dataGridView1.DataSource = dt;
    Lector.Close();
}
catch (Exception ex)
{
    MessageBox.Show("Error: " + ex.Message);
}

}

campo = campo - 1;
textBox2.Text = "";
}

private void button1_Click(object sender, EventArgs e)
{
    Close();
}

private void cmb_tabla_baja_SelectedIndexChanged_1(object sender, EventArgs e)
{
    // Segun la tabla seleccionada cargo la grilla con todos sus registros
    string consulta = "Select * from " + cmb_tabla_baja.Text;
    Orden = new OleDbCommand(consulta, ConexionConBD);
    Lector = Orden.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(Lector);
    grilla_baja.DataSource = dt;
    Lector.Close();
    cmb_filtro_baja.Items.Clear();
    for (i = 0; i < grilla_baja.Columns.Count; i++)
    {
        string col = Convert.ToString(grilla_baja.Columns[i].HeaderText);
        cmb_filtro_baja.Items.Add(col);
    }
}

private void boton_borra_Click(object sender, EventArgs e)
{

    if (((cmb_tabla_baja.Text != null) & (cmb_tabla_baja.Text != "") & ((txt_filtro_baja.Text != null) & (txt_filtro_baja.Text != "")))
    {
        string SQL_baja = "DELETE from " + cmb_tabla_baja.Text + " WHERE [" +
        cmb_filtro_baja.Text + "] = " + txt_filtro_baja.Text;
        MessageBox.Show("Se ejecuta:\n" + SQL_baja, "Se ejecuta:");
    }
}

// Ejecuto el SQL_baja
Orden = new OleDbCommand(SQL_baja, ConexionConBD);

```

```

try
{
    Orden.ExecuteNonQuery();

    // Recargo la grilla
    string consulta = "Select * from " + cmb_tabla_baja.Text;
    Orden = new OleDbCommand(consulta, ConexionConBD);
    Lector = Orden.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(Lector);
    grilla_baja.DataSource = dt;
    Lector.Close();
    cmb_filtro_baja.Items.Clear();
    cmb_filtro_baja.Text = "";

    boton_borra.Enabled = false;
}
catch (Exception ex)
{
    MessageBox.Show("Error: " + ex.Message);
}
}

private void grilla_baja_CellClick(object sender, DataGridViewCellEventArgs e)
{
    lista_campos_baja.Items.Clear();
    string registro;
    for (i = 0; i < grilla_baja.Columns.Count; i++)
    {
        string col = Convert.ToString(grilla_baja.Columns[i].HeaderText);
        string dato =
Convert.ToString(grilla_baja.Rows[grilla_baja.CurrentCellAddress.Y].Cells[i].Value);
        registro = col + " = " + dato;
        lista_campos_baja.Items.Add(registro);
    }
    boton_borra.Enabled = true;
}

private void cmb_tabla_modif_SelectedIndexChanged(object sender, EventArgs e)
{
    // Segun la tabla seleccionada cargo la grilla con todos sus registros
    string consulta = "Select * from " + cmb_tabla_modif.Text;
    Orden = new OleDbCommand(consulta, ConexionConBD);
    Lector = Orden.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(Lector);
    grilla_modif.DataSource = dt;
    Lector.Close();

    // Habilito para modificar
    groupBox2.Enabled = true;
    cmb_criterio_modif.Items.Clear();
    for (i = 0; i < grilla_modif.Columns.Count; i++)
    {
        string col = Convert.ToString(grilla_modif.Columns[i].HeaderText);
        cmb_criterio_modif.Items.Add(col);
        cmb_campo_modif.Items.Add(col);
    }
}

private void boton_cancela_modif_Click(object sender, EventArgs e)
{
    cmb_criterio_modif.Items.Clear();
    cmb_campo_modif.Items.Clear();
    txt_criterio_modif.Text = "";
    txt_valor_modif.Text = "";
    cmb_criterio_modif.Text = "";
    cmb_campo_modif.Text = "";
    groupBox2.Enabled = false;
}

private void button2_Click(object sender, EventArgs e)
{
    string SQL_Modif = "UPDATE " + cmb_tabla_modif.Text + " SET [";
    SQL_Modif = SQL_Modif + cmb_campo_modif.Text + "] = " + txt_valor_modif.Text;
    SQL_Modif = SQL_Modif + " WHERE [" + cmb_criterio_modif.Text + "] = " +
txt_criterio_modif.Text;
    MessageBox.Show(SQL_Modif, "Se ejecuta:");
}

// Ejecuto el SQL_baja

```

- 2- Una vez que entiendan su funcionalidad y el código que hay detrás de cada objeto entonces estarán en condiciones de realizar una prueba del mismo. Verán que el programa no indica cómo hacer las cosas, hay partes que tienen que “adivinar”. Para solucionar esto a futuros usuarios lo que deberán hacer es generar el manual de usuario del sistema en modo “APB” como saben.
 - 3- A medida que lo comiencen a usar/probar verán que hay algunos errores, no a nivel código funcional sino a nivel validaciones de datos por ejemplo. Estos errores y todo los que encuentres tendrás que tratar de solucionarlos sin cambiar la finalidad del proceso o procedimiento que modifiques.
 - 4- Si logras realizar estas tareas no solo sabrás escribir/generar un manual de usuario claro y completo sino que al mismo tiempo sabrás solucionar/arreglar problemas de programas ya hechos y entenderás y sabrás nuevos objetos y métodos de los mismos. De la misma manera aprenderás nuevos temas como por ejemplo “validaciones de datos” o “control de errores” para que el programa no “pinche”, temas que veremos a lo largo de las clases siguientes.



(Para poder bajarlo simplemente doble click sobre el ícono anterior)

36. Cambio de Formulario inicial

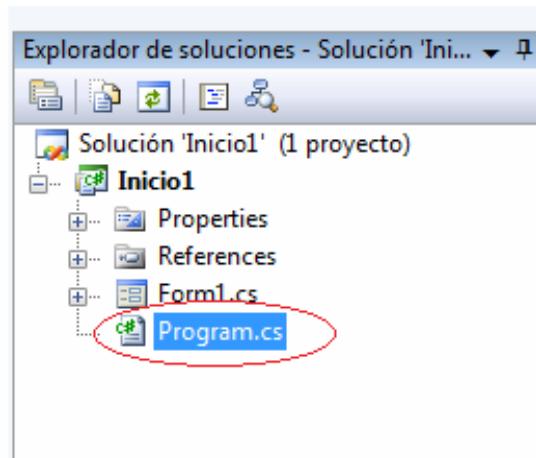
Capítulo desarrollado por: Ruben Moruzzi

Lo que se explica en este documento son dos cosas:

- 1- Iniciar desde un formulario distinto del Form1
- 2- Iniciar desde el proyecto que se elija, cuando hay más de uno en ejecución

Cuando queremos iniciar la aplicación desde un formulario diferente al que nos da por default el visual studio, podemos asignar desde el program.cs el formulario que queremos que comience en nuestra aplicación.

Para hacerlo, vamos a entrar al program.cs, que se encuentra en el explorador de soluciones, y entramos al código haciendo doble clic.



El programa va a abrir una nueva ventana, donde se encuentran las siguientes líneas de código.

```

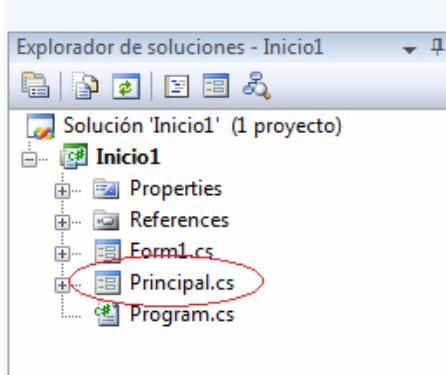
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Inicio1
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

Como verán, en la línea de donde se encuentra el formulario con el que va a arrancar la compilación es Form1.

Para cambiar esto, lo primero que debemos hacer es crear un nuevo formulario.



En este caso, el nuevo formulario, se llama Principal.

Volvemos a las líneas de código del Program.cs, y cambiamos el nombre del formulario anterior por el recién creado.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Inicio1
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Principal());
        }
    }
}

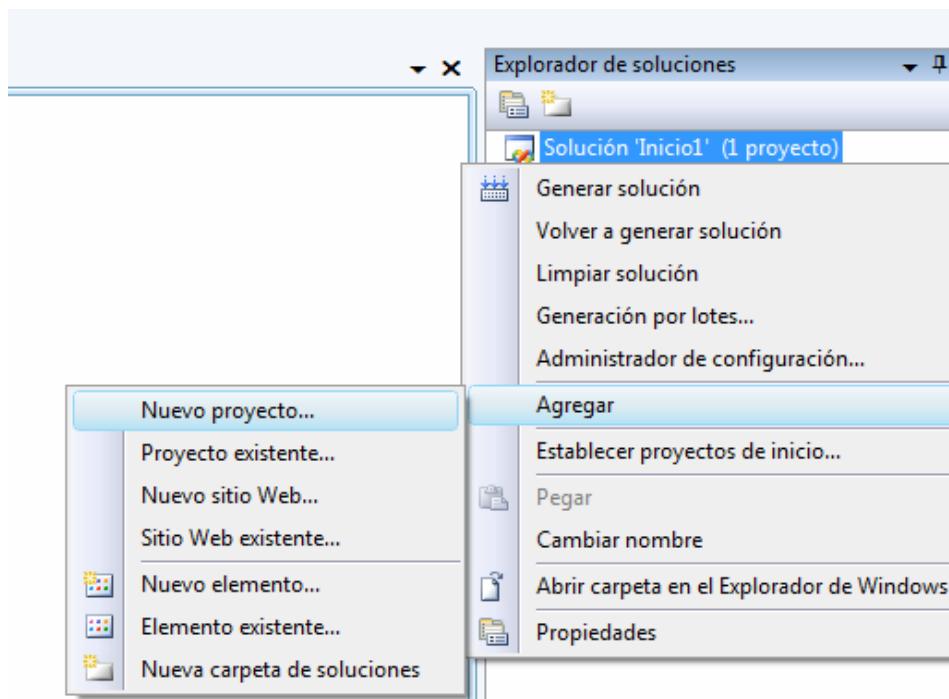
```

Lo único que se necesita modificar es el nombre del formulario.

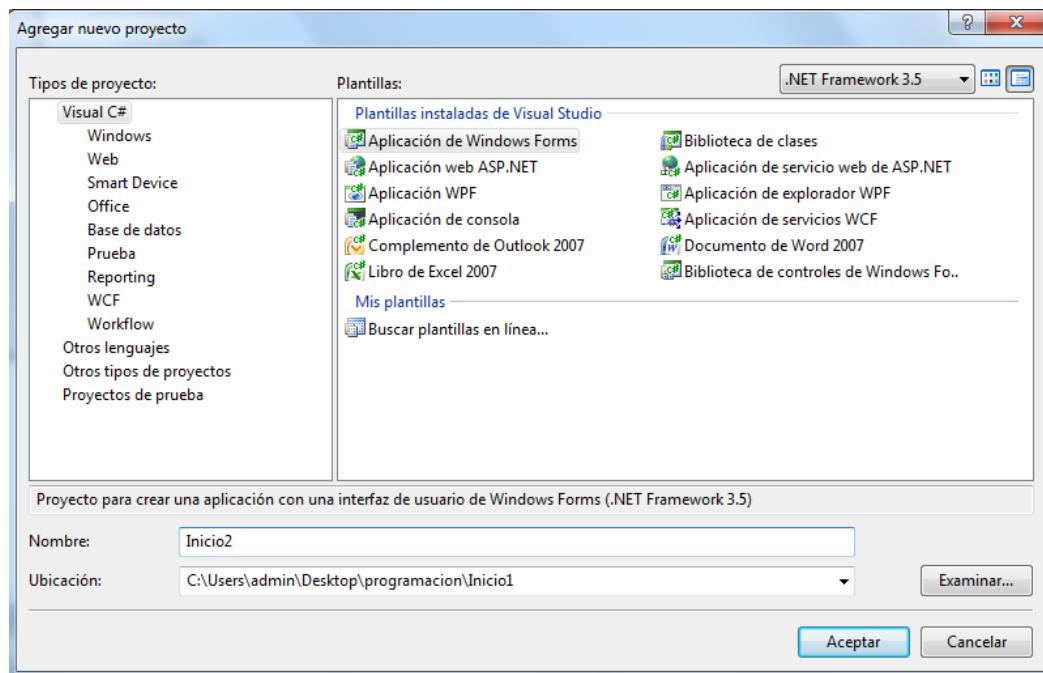
Con esto ya está definido el nuevo formulario para iniciar en la aplicación.

Para iniciar un aplicación diferente

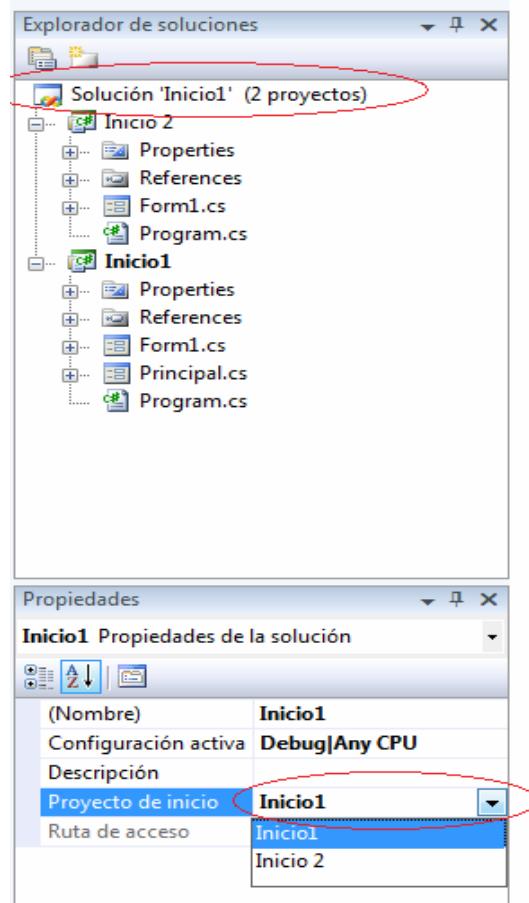
Lo primero que debemos hacer es crear una segunda aplicación dentro de la misma solución.



Haciendo clic derecho en la solución, Agregar y luego Nuevo proyecto.



Aparece la ventana que se muestra anteriormente indicando que es una aplicación de Windows form.



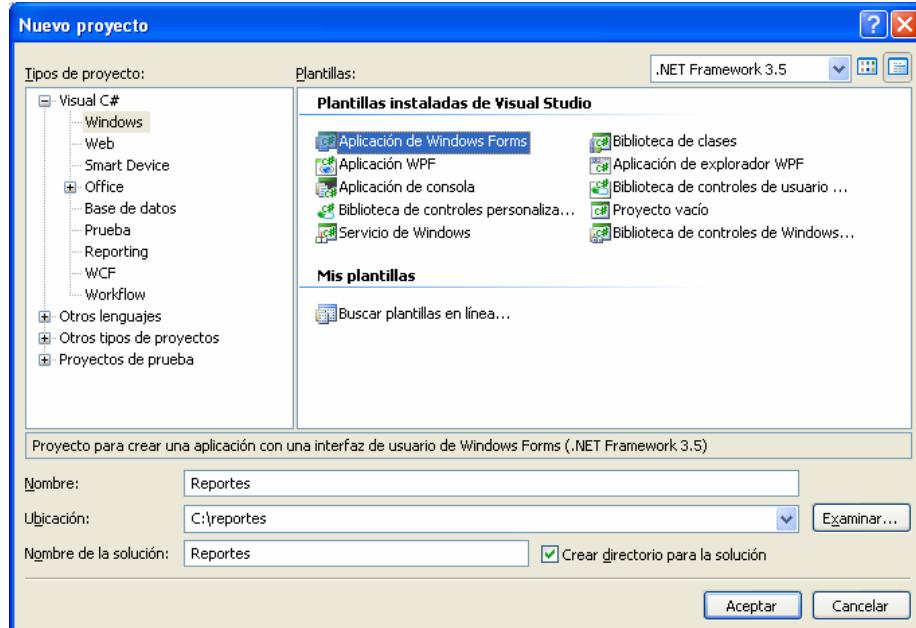
Seleccionamos la solución, y en las propiedades aparece un combo con las soluciones disponibles para comenzar. Para finalizar seleccionamos la que queremos y listo. La solución comenzará desde la nueva aplicación.

37. Generación de reportes de DB con Visual C#

El siguiente paso a paso realizado en C# muestra cómo generar reportes con los datos que están guardados en una tabla en una DB que en este caso es en Access.

Lo que se debe tener es un proyecto de “Aplicación de Windows Forms” y una DB con una tabla con datos sobre la cual se generará el reporte:

- 1- Generamos un proyecto de Windows Forms:

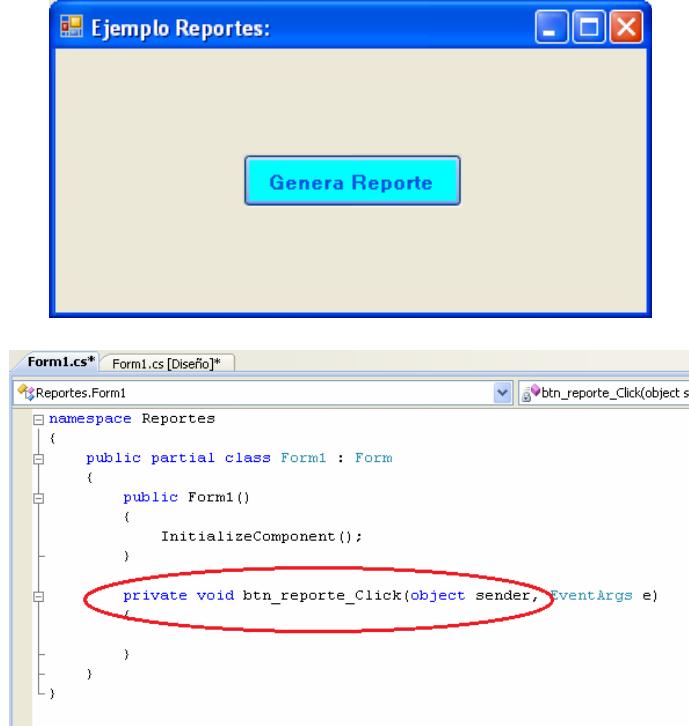


- 2- Generamos una DB con una tabla a modo ejemplo con datos que se mostraran en nuestro reporte:

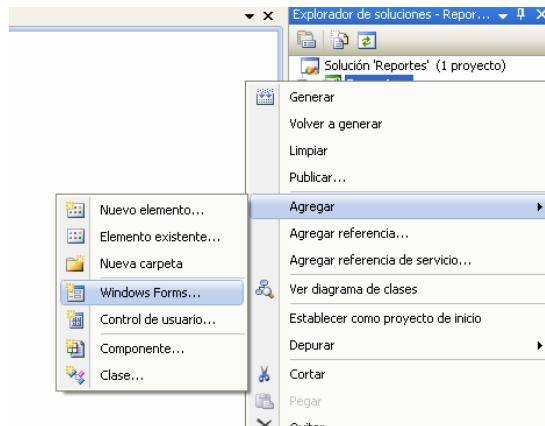
Microsoft Access - [Detalles : Tabla]			
	id	descripcion	precio
	1	Papas	7,5
	2	Tomate	12
	3	Manzana	8,5
	4	Lechuga	15
	5	Pera	9,75
	0		0

Una vez que tengamos el proyecto debidamente generado y la DB con una tabla cargadas de algunos registros para ser tomados por nuestro reporte nos metemos en la parte de la utilización de los objetos requeridos y con la codificación o configuración de los mismos.

- 3- Lo primero que vamos a agregar es un objeto que tenga o genere en evento en el cual podamos escribir o codificar la llamada el reporte:

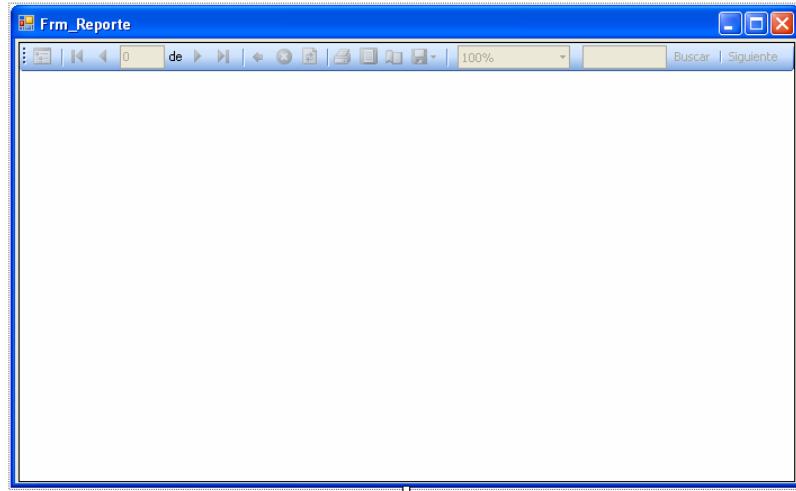


- 4- Para el reporte debemos usar un objeto que permita generar el mismo y dicho objeto debe estar alojado en otro objeto que al llamarlo contenga el evento “Load” para que dentro de ese evento se actualice el reporte de forma automática. Para eso vamos a agregar un nuevo Form de modo tal que usaremos el evento Load del mismo para actualizar el reporte:



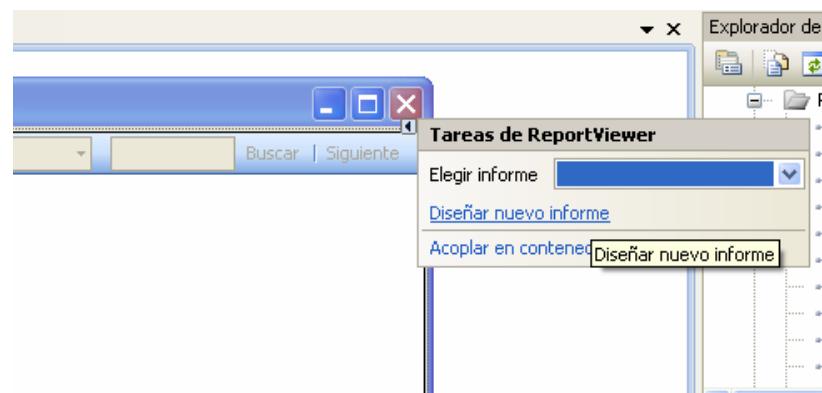
- 5- Dentro del nuevo Form (en este caso de nombre Frm_Reporte) agregamos el nuevo objeto llamado “MicrosoftWindowsViewer”:



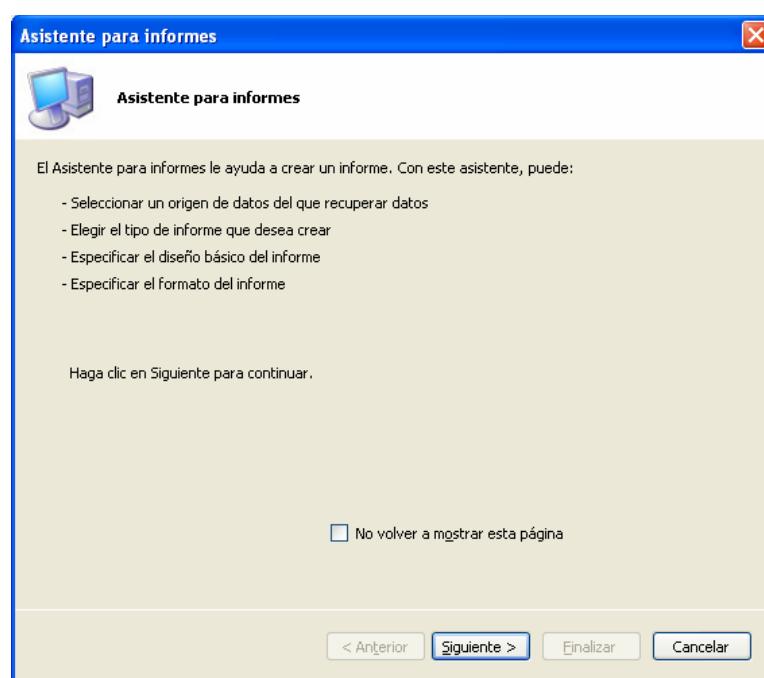


6- Ahora viene la parte de configurar ese nuevo objeto para que tome los datos de esa tabla que creamos en la DB.

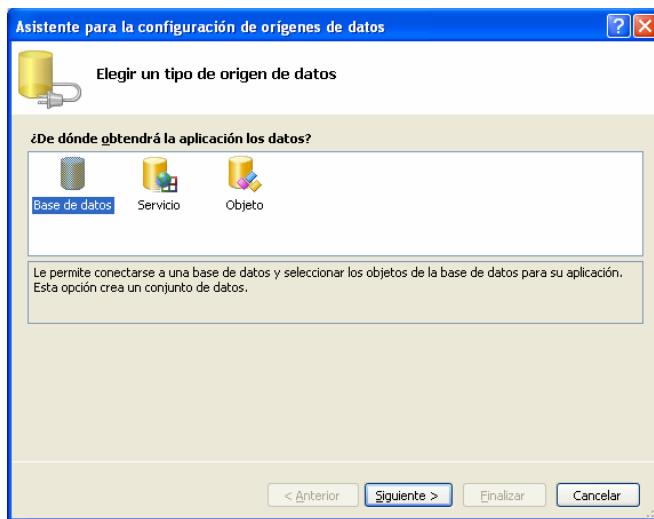
a- Ingresamos a la configuración del objeto y elegimos “Diseñar nuevo Informe”:



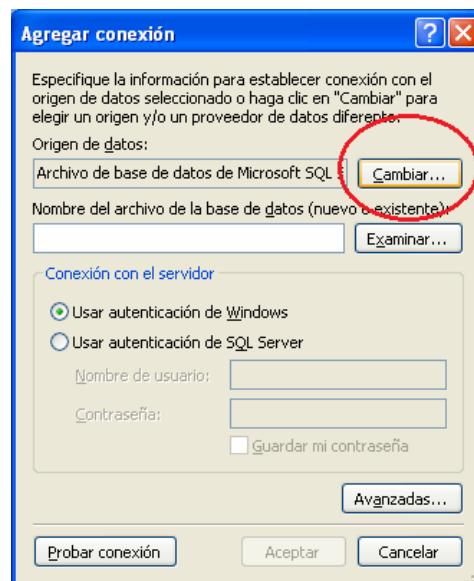
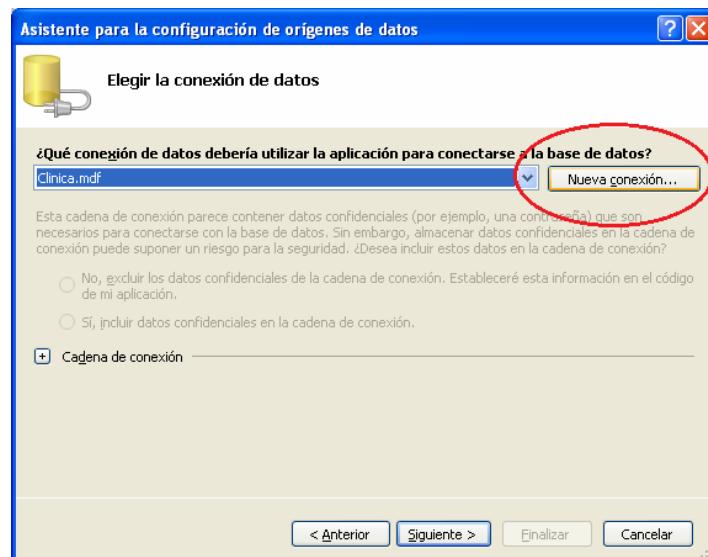
b- Click en “Siguiente”:

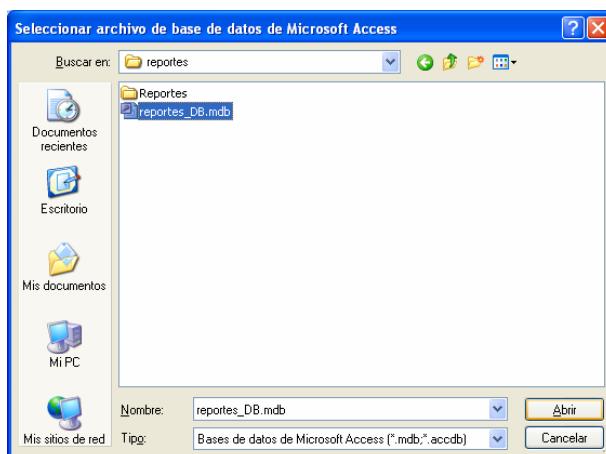
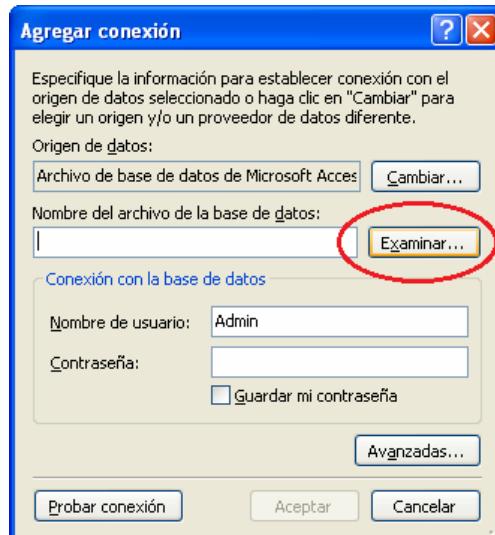
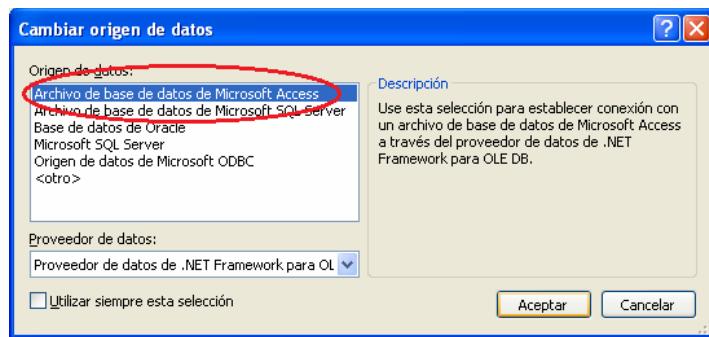


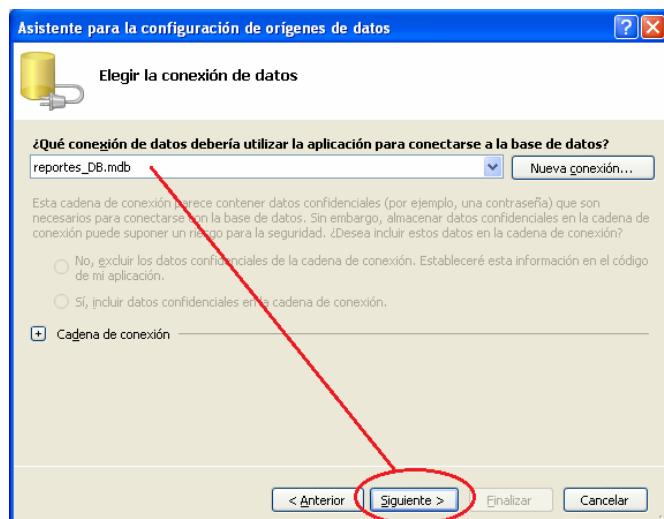
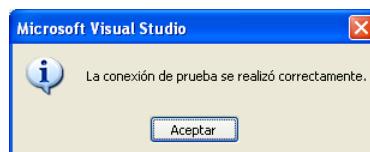
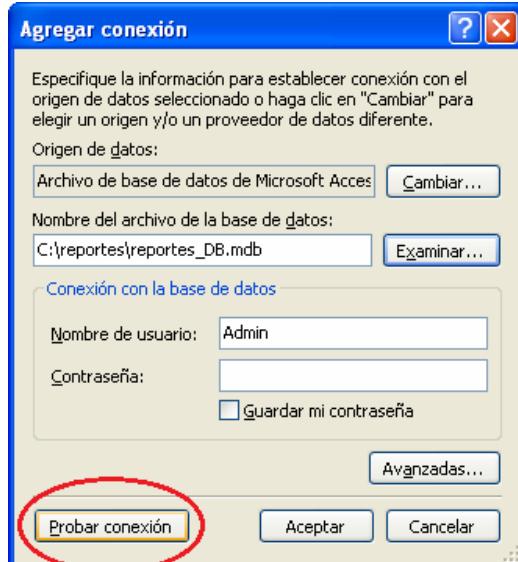
c- Elegimos “Base de Datos”:

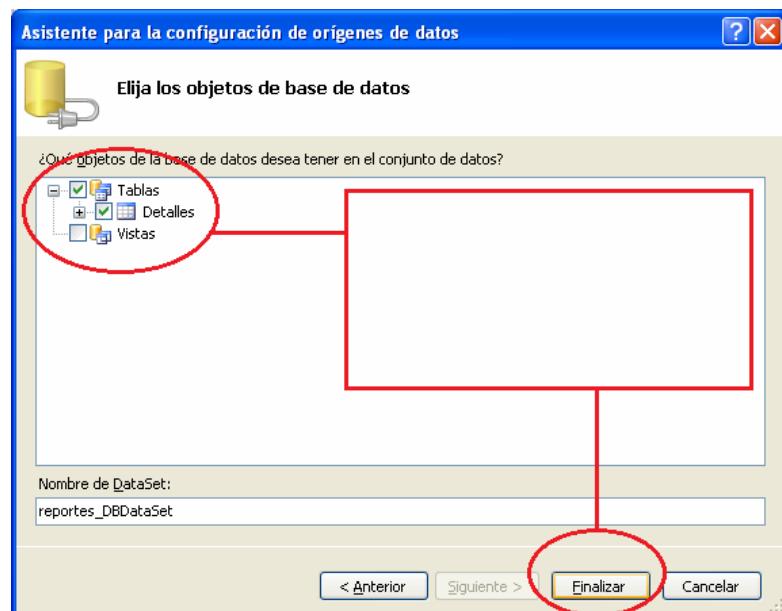
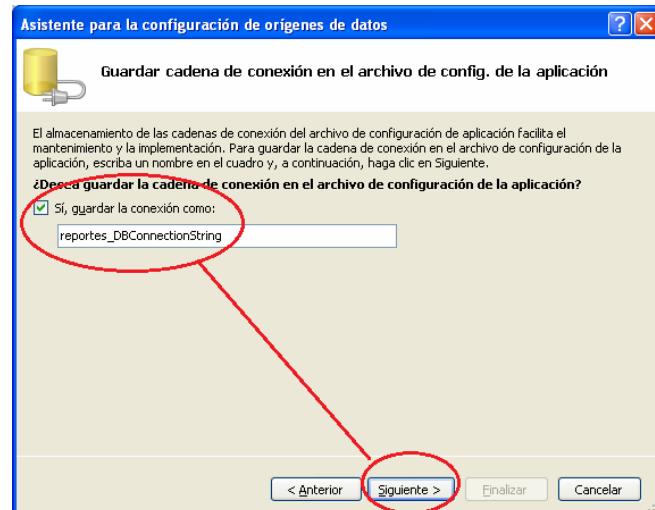
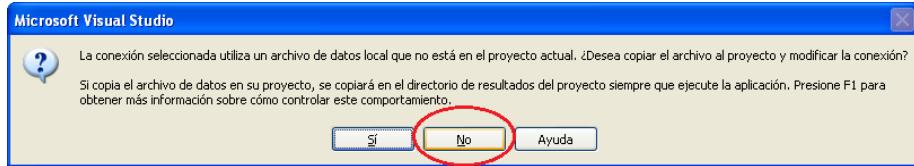


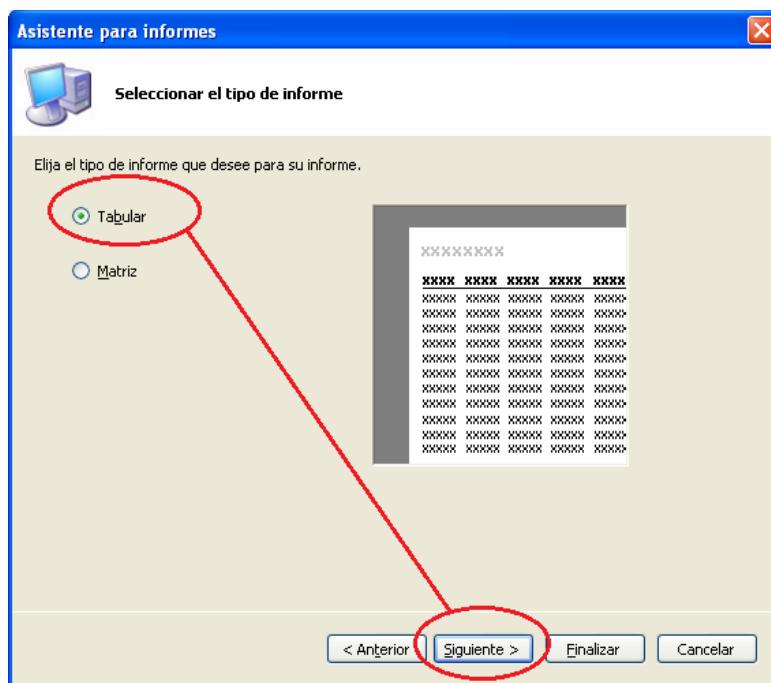
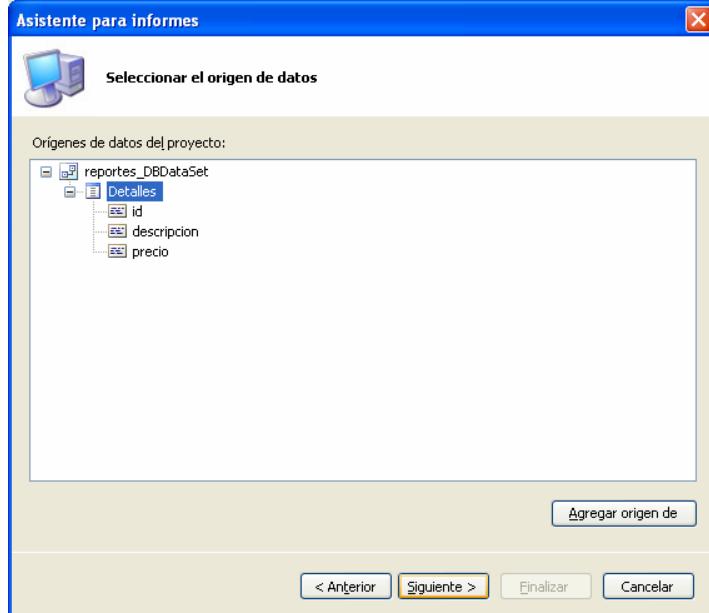
d- Click en “Nueva Conexión” y seguimos el resto de los pasos:

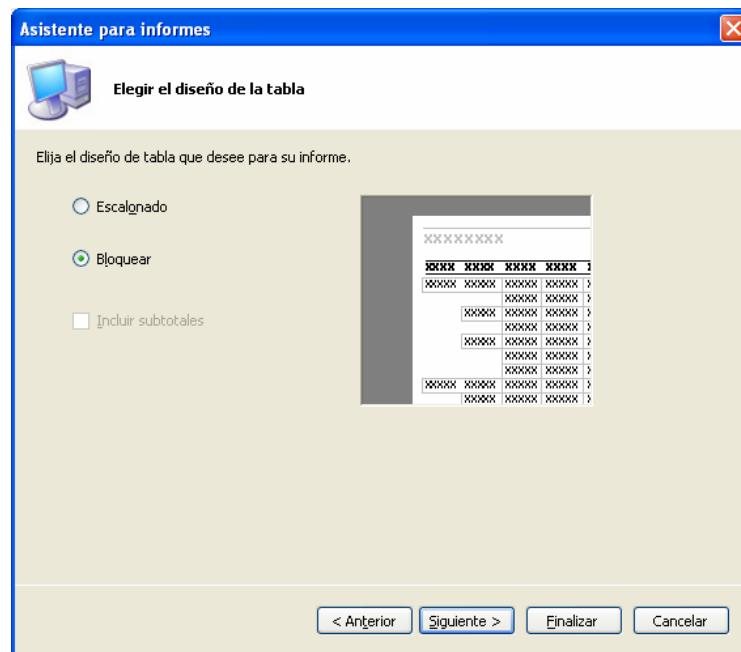
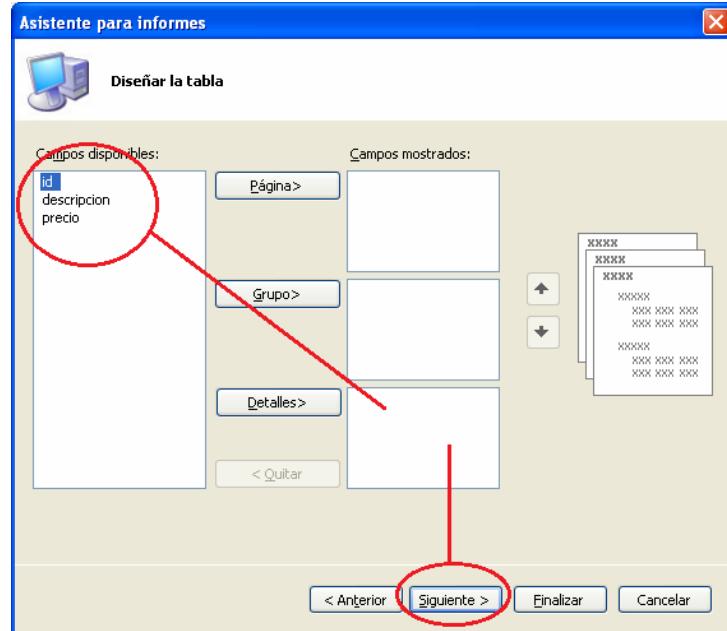


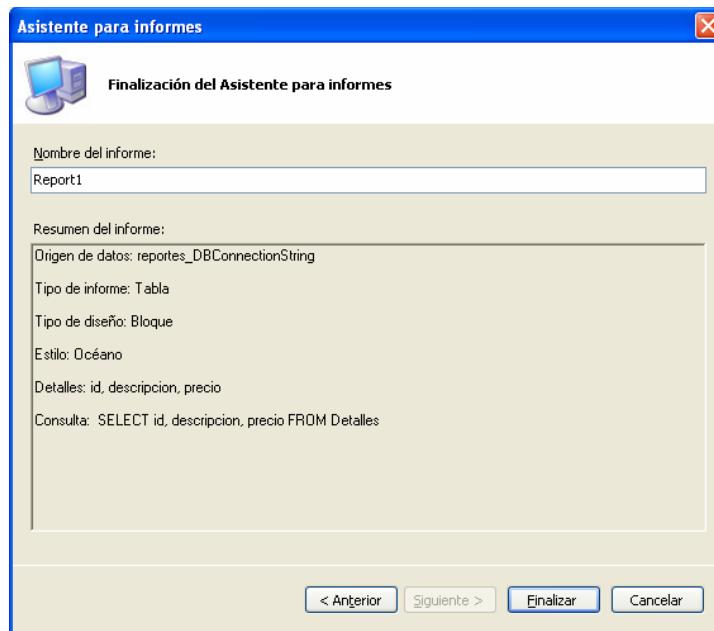
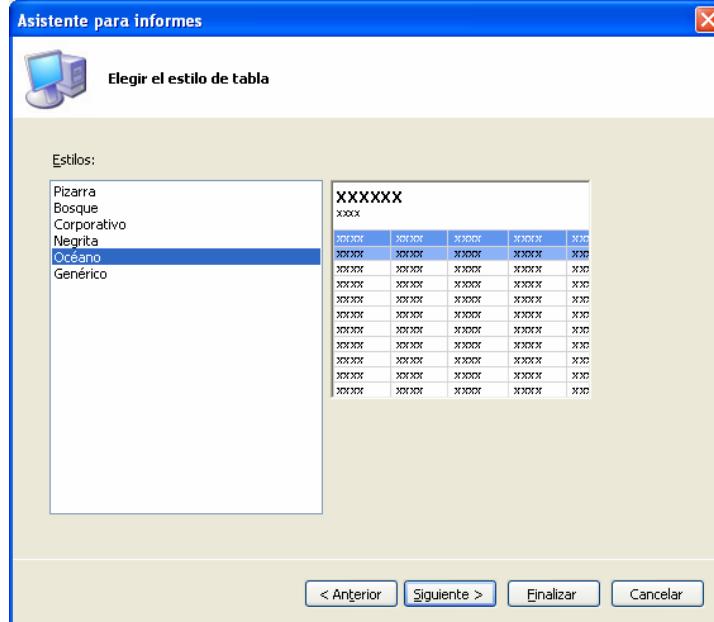




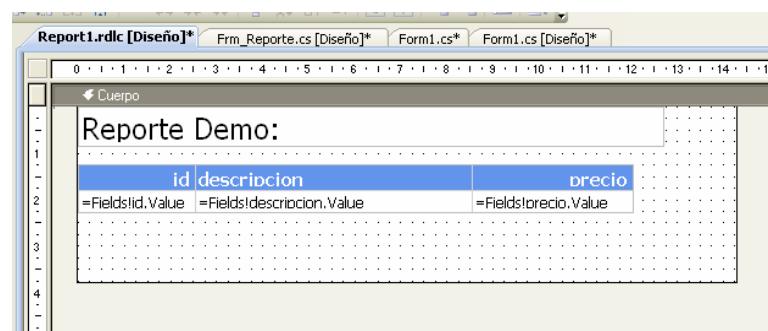




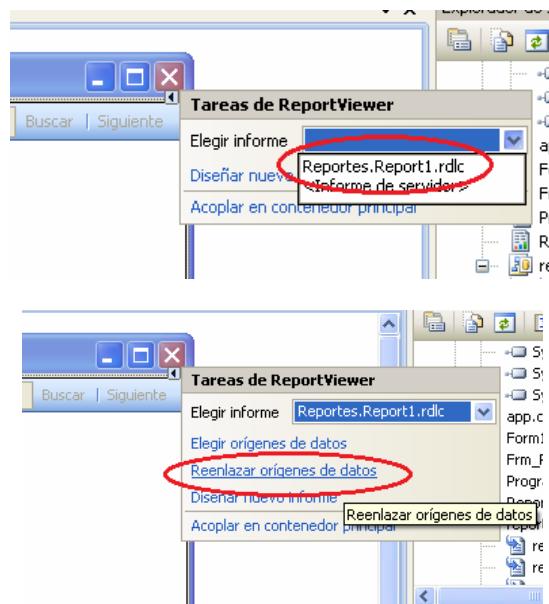




- e- Al hacer click en “Finalizar” de nuestra última ventana de configuraciones nos presentará el “Report1.rdlc [Diseño]”:



- f- Luego de que acomodamos a gusto los campos, tenemos que volver a las propiedades del objeto de reportes y elegir el “Informe” a mostrar que es básicamente el que generemos con todas esas configuraciones anteriores:



- g- Ahora solo queda llamar al nuevo formulario desde el evento Click de nuestro único botón en agregado en un principio al formulario principal:

```

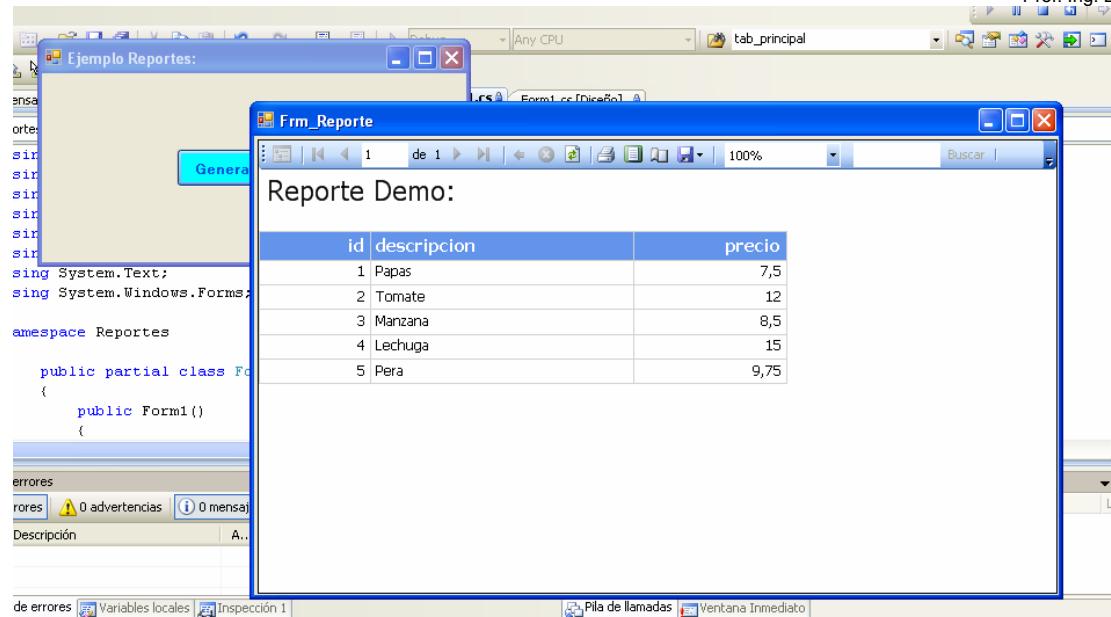
Report1.rdlc [Diseño] Frm_Reporte.cs [Diseño] Form1.cs [Diseño] Form1.cs [Diseño]
Reportes.Form1
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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

        private void btn_reporte_Click(object sender, EventArgs e)
        {
            Form frm_1 = new Frm_Reporte();
            frm_1.Show();
        }
    }
}

```

- h- Si todo salió bien, cuando ejecutemos la aplicación se debería ver lo siguiente:



38. Especial Funciones

Una función es un módulo de un programa separado del cuerpo principal, que realiza una tarea específica y que puede regresar un valor a la parte principal del programa u otra función o procedimiento que la invoque.

¿Para qué nos sirve?

Segmentar el código en funciones permite al programador crear piezas modulares de código que realizan una tarea definida y vuelve a la zona del programa en la que fueron llamadas. El caso típico para crear una función, es cuando uno necesita realizar la misma acción múltiples veces dentro de un mismo programa.

La estandarización de fragmentos de código en funciones tiene diversas ventajas:

- * Las funciones ayudan al programador a ser organizado. Además ayudan a conceptualizar el programa.
- * Las funciones codifican una acción en un lugar, así que sólo deben ser depuradas de errores una vez.
- * Reducen las posibilidades de error en modificaciones, si el código debe ser cambiado.
- * Hacen más fácil la reutilización de código en otros programas por hacerlo más modular y, como efecto paralelo, usando funciones se obtiene un código más legible.

Estructura básica de una función en c#:

```
TipoMetodo1 TipoDeDatoDevuelve2 NombreFuncion(Parametros3)
{
    Codigo;
    Return ()4;
}
```

En donde:

1-Tipo de método:

Private
Public
Protected

2-Tipo de datos que devuelve nuestra función:

Int
string
boolean
OleDbDataReader

Void (Vacío)- En este caso, la función no lleva return, ya que no se espera ninguna devolución.

3-Parametros que recibe la función:

Estos van a ir dentro de los paréntesis después del nombre que le damos a la función y separados por comas uno del otro.

Se deben escribir de esta forma (Tipo_var Nom_Variable, Tipo_Var Nom_Var).

Una función puede, como puede que no, recibir parámetros. En el caso de no recibir, los paréntesis quedarían vacíos ().

4- Return:

En el retorno vamos a especificar qué es lo que se va a devolver.

También puede ser que no devuelva algo a la llamada que la invoca.

Si vemos en la estructura, seguido del tipo de método, está el tipo de dato que va a devolver. En caso de que no queramos devolver nada (VOID) podemos omitir el return, o solo poner return;

Donde ubicar una función:

Dentro de la clase principal, en nuestro caso sería (por ahora) dentro de la clase form, al igual que los demás eventos.

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

        private void MiFuncion()
        {
            //Codigo
        }

        private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

Como llamar (ejecutar) a la función que creamos:

Dependiendo de si devuelve valores, vamos a ejecutarlas de diferentes maneras:

Si hay devolución, debemos guardar el resultado en algún lugar.

Puede ser en variables, vectores, matrices, objetos, etc. O simplemente usarlo.

```
int variable= funcion();
```

Ahora, si no existe devolución por parte de la función, solo la ejecutamos.

```
funcion();
```

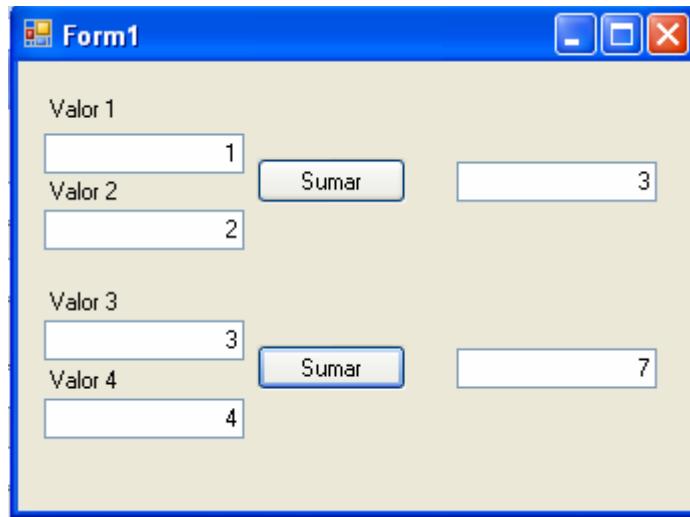
Y si recibe parámetros (supongamos que recibe un valor entero):

```
funcion(Numero);
```

O directamente podemos escribir el número sin necesidad de tenerlo en una variable

```
funcion(5);
```

Ejemplo:



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

        private int Suma(int Valor1, int Valor2)
        {
            int Resultado = Valor1 + Valor2;
            return (Resultado);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int val1 = Convert.ToInt32(txtvalor1.Text);
            int val2 = Convert.ToInt32(txtvalor2.Text);
            txtresultado1.Text = Suma(val1, val2).ToString();
        }

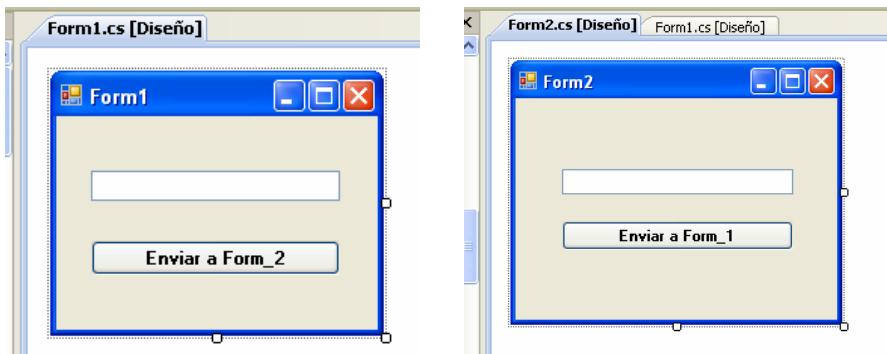
        private void button2_Click(object sender, EventArgs e)
        {
            int val1 = Convert.ToInt32(txtvalor3.Text);
            int val2 = Convert.ToInt32(txtvalor4.Text);
            txtresultado2.Text = Suma(val1, val2).ToString();
        }
    }
}
```

39. Manejo de objetos de un formulario a otro

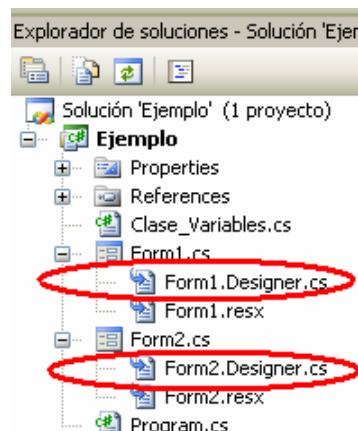
El siguiente ejemplo es simple, demuestra cómo realizar el manejo de objetos entre dos formularios poniendo las propiedades del objeto como públicas.

Con esto podemos lograr interactuar con los eventos y propiedades de un objeto (Name, text, BackColor, etc) desde otro formulario.

1-En un nuevo proyecto creo dos formularios con un TextBox y un Boton cada uno:



2-Entro al Designer* de cada formulario



3-Modificar los objetos con los que se trabajara desde otro formulario de private a public para que el acceso al mismo no este restringido a acciones externas.

```

namespace Ejemplo
{
    partial class Form2
    {
        /// <summary>
        /// Variable del diseñador requerida.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se están utilizando.
        /// </summary>
        /// <param name="disposing">true si los recursos administrados
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        // Código generado por el Diseñador de Windows Forms

        public System.Windows.Forms.TextBox Txt2;
        public System.Windows.Forms.Button Bt2;
    }
}

```

4- Declarar una nueva instancia de cada formulario como pública y estática*, y modificar las propiedades de los objetos:

-En el Form1 crear una nueva instancia del Form2.

-En el Form2 crear una nueva instancia del Form1.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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

        public static Form2 fr = new Form2();

        private void Bt1_Click(object sender, EventArgs e)
        {
            fr.Txt2.Text = Txt1.Text;
            fr.Show();
        }
    }
}

```

Atención:

No se puede modificar el primer formulario, ya que cuando se crea una nueva instancia para el Form1 lo toma como un nuevo formulario y no como el que se encuentra abierto. Si se cierra un formulario se pierde la instancia (fr), ya que deja de existir el objeto al que se hace referencia.

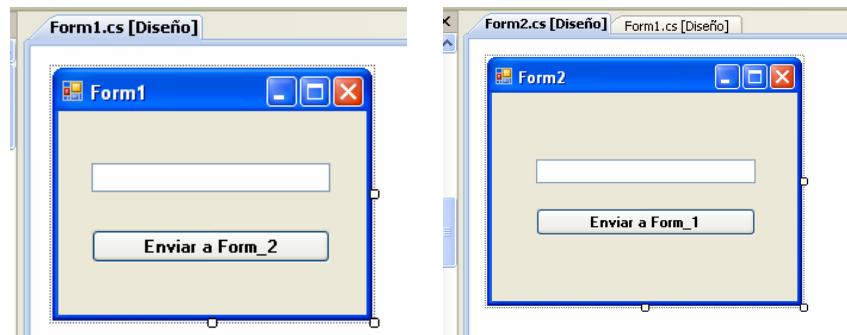
*Designer: Diseñador del formulario, aquí se encuentra el código que crea a cada objeto y sus características.

*Una variable estática es aquella que mantiene su estructura como tal durante toda la ejecución del programa por lo que se dice que la vida de la misma se extiende durante todo el uso de dicho programa.

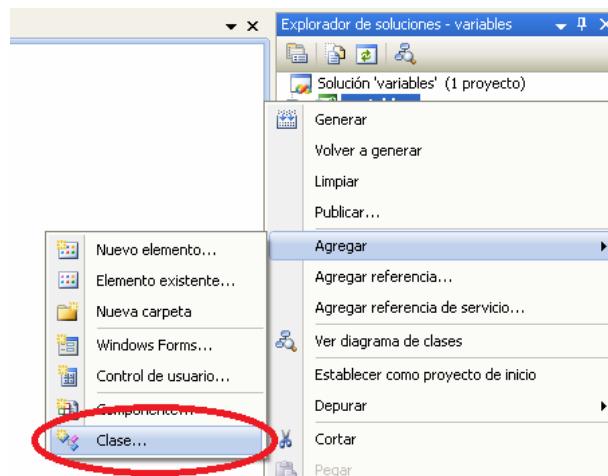
40. Pasaje de valores (datos en variables) de un formulario a otro

El siguiente ejemplo es simple, pero demuestra cómo realizar un pasaje de valores entre dos formularios usando variables declaradas en una clase pública:

- 1- En un nuevo proyecto creo dos formularios con un TextBox y un Botón cada uno:



- 2- Agrego una clase a mi solución en la cual voy a declarar las variables públicas a usar:



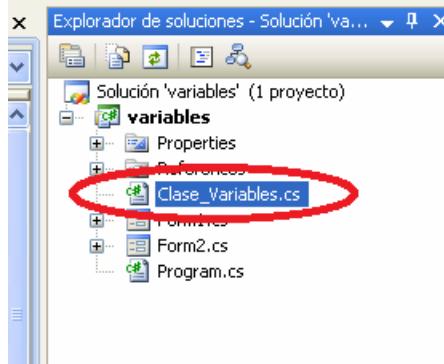
- 3- La misma la llamé Clase_Variables:

```

Clase_Variables.cs*  Form1.Designer.cs*  Form2.cs [Diseño]*  For
Ejemplo.Clase_Variables
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Ejemplo
{
    class Clase_Variables
    {
        public static string Texto = "";
    }
}

```



Dentro de la clase establecemos una variable estática y pública, para que pueda ser utilizada desde cualquier lugar que se la invoque y que su contenido permanezca estático.

- 4- Una vez declarada la variable, solo queda pasar el contenido que le asignemos de un Textbox a otro.

Lo hacemos de la siguiente manera.

Bt1 y Txt1 pertenecen al Form1.

Bt2 y Txt2 al Form2.

-En el evento clic de Bt1 asignamos el contenido del Txt1 a la variable de Clase_Variables.

-Luego recorremos con un ciclo for los formularios que están abiertos para darle el foco al que queremos ir.

Para eso usamos Application.OpenForms.Count.

-Comparamos si el nombre del formulario para verificar si está abierto.

-Si está abierto solo le damos el foco y rompemos el bucle, en caso contrario abrimos uno nuevo (código dentro del catch).

```

private void Bt1_Click(object sender, EventArgs e)
{
    Clase_Variables.Texto=Txt1.Text;
    try
    {
        for (int i = 0; i <= Application.OpenForms.Count; i++)
        {
            if (Application.OpenForms[i].Name == "Form2")
            {
                Application.OpenForms[i].Focus();
                break;
            }
        }
    }
    catch
    {
        Form2 fr = new Form2();
        fr.Show();
    }
}

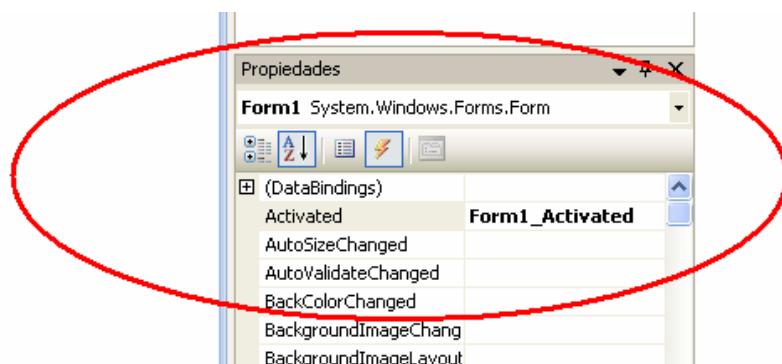
```

- 5- Lo mismo hacemos con el Bt2, pero con una pequeña diferencia. Ya que el Form1 ya está abierto, no necesitamos abrir uno nuevo, así que solo debemos buscar su posición entre los forms y darle el foco de esta manera.

```
private void Bt2_Click(object sender, EventArgs e)
{
    Clase_Variables.Texto = Txt2.Text;
    for (int i = 0; i <= Application.OpenForms.Count; i++)
    {
        if (Application.OpenForms[i].Name == "Form1")
        {
            Application.OpenForms[i].Focus();
            break;
        }
    }
}
```

- 6- Por último agregarle a cada formulario el evento Activated haciéndole doble clic

y
de la



```
private void Form1_Activated(object sender, EventArgs e)
{
    Txt1.Text = Clase_Variables.Texto;
}
```

asignar al Textbox el contenido
variable Texto de
Clase_Variables.

Form1:

41. EXTRA 1 - Apuntes de SQL

Índice:

- 1 - Introducción
 - 2 - Consultas de selección
 - 3 - Criterios de selección
 - 4 - Agrupamiento de registros
 - 5 - Consultas de acción
 - 6 - Tipos de datos
 - 7 - Subconsultas
 - 8 - Consultas de referencias cruzadas
 - 9 - Consultas de unión internas
 - 10 - Consultas de unión externas
 - 11 - Estructuras de las tablas
 - 12 - Consultas con parámetros
 - 13 - Bases de datos externas
 - 14 - Omitir los permisos de ejecución
 - 15 - La cláusula Procedure
 - 16 - Anexos
-

1.- INTRODUCCION

1.1. Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

1.2 Comandos

Existen dos tipos de comandos SQL:

los DLL que permiten crear y definir nuevas bases de datos, campos e índices.

los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL

CREATE: Utilizado para crear nuevas tablas, campos e índices

DROP: Empleado para eliminar tablas e índices

ALTER: Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML

SELECT: Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado

INSERT: Utilizado para cargar lotes de datos en la base de datos en una única operación.

UPDATE: Utilizado para modificar los valores de los campos y registros especificados

DELETE: Utilizado para eliminar registros de una tabla de una base de datos

1.3 Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

FROM: Utilizada para especificar la tabla de la cual se van a seleccionar los registros

WHERE: Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.

GROUP BY: Utilizada para separar los registros seleccionados en grupos específicos

HAVING: Utilizada para expresar la condición que debe satisfacer cada grupo

ORDER BY: Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

1.4 Operadores Lógicos

AND: Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.

OR: Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.

NOT: Negación lógica. Devuelve el valor contrario de la expresión.

1.5 Operadores de Comparación

<: Menor que

>: Mayor que

<>: Distinto de

<=: Menor ó Igual que

>=: Mayor ó Igual que

=: Igual que

BETWEEN: Utilizado para especificar un intervalo de valores.

LIKE: Utilizado en la comparación de un modelo

In: Utilizado para especificar registros de una base de datos

1.6 Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

AVG: Utilizada para calcular el promedio de los valores de un campo determinado

COUNT: Utilizada para devolver el número de registros de la selección

SUM: Utilizada para devolver la suma de todos los valores de un campo determinado

MAX: Utilizada para devolver el valor más alto de un campo especificado

MIN: Utilizada para devolver el valor más bajo de un campo especificado

2.- CONSULTAS DE SELECCIÓN

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

2.1 Consultas básicas

La sintaxis básica de una consulta de selección es la siguiente:

`SELECT Campos FROM Tabla;`

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

`SELECT Nombre, Telefono FROM Clientes;`

Esta consulta devuelve un recordset con el campo nombre y teléfono de la tabla clientes.

2.2 Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

`SELECTCodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;`

Esta consulta devuelve los campos CódigoPostal, Nombre, Teléfono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por más de un campo, como por ejemplo:

`SELECT CódigoPostal, Nombre, Teléfono FROM Clientes ORDER BY CódigoPostal, Nombre;`

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (ASC –se toma este valor por defecto) ó descendente (DESC)

`SELECT CódigoPostal, Nombre, Teléfono FROM Clientes ORDER BY CódigoPostal DESC , Nombre ASC;`

2.3 Consultas con Predicado

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar, los posibles predicados son:

ALL: Devuelve todos los campos de la tabla

TOP: Devuelve un determinado número de registros de la tabla

DISTINCT: Omite los registros cuyos campos seleccionados coincidan totalmente

DISTINCTROW: Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados.

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No se conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

`SELECT ALL FROM Empleados;`

`SELECT * FROM Empleados;`

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula

ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

`SELECT TOP 25 Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;`

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes .El predicado TOP no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

`SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;`

El valor que va a continuación de TOP debe ser un Integer sin signo. TOP no afecta a la posible actualización de la consulta.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos. Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT Apellido FROM Empleados;
```

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

DISTINCTROW

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campo indicados en la cláusula SELECT.

Manual de SQL

```
SELECT DISTINCTROW Apellido FROM Empleados;
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López el ejemplo del predicado DISTINCT devuleve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

2.4 Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados;
```

3.- CRITERIOS DE SELECCIÓN

En el capítulo anterior se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de este capítulo se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan una condiciones preestablecidas. Antes de comenzar el desarrollo de este capítulo hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda es que no se posible establecer condiciones de búsqueda en los campos memo y; la tercera y última hacereferencia a las fechas. Las fechas se deben escribir siempre en formato mm-dd-aa en donde mm representa el mes, dd el día y aa el año, hay que prestar atención a los separadores -no sirve la separación habitual de la barra (/), hay que utilizar el guión (-) y además la fecha debe ir encerrada entre almohadillas (#). Por ejemplo si deseamos referirnos al día 3 de Septiembre de 1995 deberemos hacerlo de la siguiente forma: #09-03-95# ó #9-3-95#.

3.1 Operadores Lógicos

Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not. A excepción de los dos últimos todos poseen la siguiente sintaxis: <expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;
SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR (Provincia = 'Buenos Aires' AND Estado = 'Casado');
```

3.2 Intervalos de Valores

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es: campo [Not] Between valor1 And valor2 (la condición Not es opcional) En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;
(Devuelve los pedidos realizados en la provincia de Madrid)
```

```
SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial', 'Nacional') FROM Editores;
(Devuelve el valor 'Provincial' si el código postal se encuentra en el intervalo, 'Nacional' en caso contrario)
```

3.3 El Operador Like

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

expresión Like modelo En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador Like para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (Like An*). El operador Like se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce Like C* en una consulta SQL, la consulta devuelve todos los valores de campo que comienzan por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar. El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

Like 'P[A-F]###'

Este ejemplo devuelve los campos cuyo contenido empieza con una letra de la A a la D seguidas de cualquier cadena.

Like '[A-D]*'

3.4 El Operador In

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es: expresión [Not] In(valor1, valor2, ...)

`SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');`

3.5 La cláusula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Después de escribir esta cláusula se deben especificar las condiciones expuestas en los apartados 3.1 y 3.2. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM.

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario > 21000;
SELECT Id_Producto, Existencias FROM Productos WHERE Existencias <= Nuevo_Pedido;
SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;
SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos = 'King';
SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos Like 'S*';
SELECT Apellidos, Salario FROM Empleados WHERE Salario Between 200 And 300;
SELECT Apellidos, Salario FROM Empl WHERE Apellidos Between 'Lon' And 'Tol';
SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE Fecha_Pedido Between #1-1-94# And #30-6-94#;
SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE Ciudad In ('Sevilla', 'Los Angeles', 'Barcelona');
```

Manual de SQL

4.- AGRUPAMIENTO DE REGISTROS

4.1 GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

`SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo`

GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción

SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas. Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados. A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no está incluido en la instrucción SELECT, siempre y cuando la instrucción SELECT incluya al menos una función SQL agregada. Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

`SELECT Id_Familia, Sum(Stock) FROM Productos GROUP BY Id_Familia;`

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuáles de ellos se van a mostrar.

`SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia HAVING Sum(Stock) > 100 AND NombreProducto Like BOS*;`

4.2 AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente Avg(expr) En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

`SELECT Avg(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;`

4.3 Count

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente Count(expr) En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto. Aunque expr puede

realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ("").

Manual de SQL

`SELECT Count(*) AS Total FROM Pedidos;`

Si expr identifica a múltiples campos, la función Count cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

`SELECT Count(FechaEnvío & Transporte) AS Total FROM Pedidos;`

4.4 Max, Min

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico de una consulta. Su sintaxis es: Min(expr) o Max(expr) En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

`SELECT Min(Gastos) AS EIMin FROM Pedidos WHERE País = 'España';
SELECT Max(Gastos) AS EIMax FROM Pedidos WHERE País = 'España';`

4.5 StDev, StDevP

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es: StDev(expr) o StDevP(expr) En donde expr representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL) StDevP evalúa una población, y StDev evalúa una muestra de la población. Si la consulta contiene menos de dos registros (o ningún registro para StDevP), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

`SELECT StDev(Gastos) AS Desviación FROM Pedidos WHERE País = 'España';
SELECT StDevP(Gastos) AS Desviación FROM Pedidos WHERE País = 'España';`

4.6 Sum

Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es: Sum(expr) En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

`SELECT Sum(PrecioUnidad * Cantidad) AS Total FROM DetallePedido;`

4.7 Var, VarP

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

`Var(expr)
VarP(expr)`

VarP evalúa una población, y Var evalúa una muestra de la población. Expr el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL) Si la consulta contiene menos de dos registros, Var y VarP devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar Var y VarP en una expresión de consulta o en una Instrucción SQL.

`SELECT Var(Gastos) AS Varianza FROM Pedidos WHERE País = 'España';
SELECT VarP(Gastos) AS Varianza FROM Pedidos WHERE País = 'España';`

5.- CONSULTAS DE ACCIÓN

Las consultas de acción son aquellas que no devuelven ningún registro, son las encargadas de acciones como añadir y borrar y modificar registros.

5.1 DELETE

Crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula FROM que satisfagan la cláusula WHERE. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Su sintaxis es:

`DELETE Tabla.* FROM Tabla WHERE criterio`

DELETE es especialmente útil cuando se desea eliminar varios registros. En una instrucción **DELETE** con múltiples tablas, debe incluir el nombre de tabla (Tabla.*). Si especifica más de una tabla desde la que eliminar registros, todas deben ser tablas de muchos a uno. Si desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Se puede utilizar **DELETE** para eliminar registros de una única tabla o desde varios lados de una relación uno a muchos.

Las operaciones de eliminación en cascada en una consulta únicamente eliminan desde varios lados de una relación. Por ejemplo, en la relación entre las tablas Clientes y Pedidos, la tabla Pedidos es la parte de muchos por lo que las operaciones en cascada solo afectarán a la tabla Pedidos. Una consulta de borrado elimina los registros completos, no únicamente los datos en campos específicos. Si desea eliminar valores en un campo especificado, crear una consulta de actualización que cambie los valores a Null.

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado.

Mantenga copias de seguridad de sus datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

```
DELETE * FROM Empleados WHERE Cargo = 'Vendedor';
```

5.2 INSERT INTO

Agrega un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipo: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla.

5.2.1 Para insertar un único Registro:

En este caso la sintaxis es la siguiente:

```
INSERT INTO Tabla (campo1, campo2, ..., campoN) VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente. Hay que prestar especial atención a acotar entre comillas simples ('') los valores literales (cadenas de caracteres) y las fechas indicarlas en formato mm-dd-aa y entre caracteres de almohadillas (#).

5.2.2 Para insertar Registros de otra Tabla:

En este caso la sintaxis es:

Manual de SQL

```
INSERT INTO Tabla [IN base_externa] (campo1, campo2, ..., campoN) SELECT TablaOrigen.campo1,  
TablaOrigen.campo2, ..., TablaOrigen.campoN FROM TablaOrigen
```

En este caso se seleccionarán los campos 1,2, ..., n de la tabla origen y se grabarán en los campos 1,2,..., n de la Tabla.

La condición **SELECT** puede incluir la cláusula **WHERE** para filtrar los registros a copiar. Si Tabla y TablaOrigen poseen la misma estructura podemos simplificar la sintaxis a:

```
INSERT INTO Tabla SELECT TablaOrigen.* FROM TablaOrigen
```

De esta forma los campos de TablaOrigen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de TablaOrigen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de TablaOrigen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementándose como le corresponde.

Se puede utilizar la instrucción **INSERT INTO** para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código especifica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar **INSERT INTO** para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula **SELECT ... FROM** como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula **SELECT** especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta.

Si la tabla destino contiene una clave principal, hay que asegurarse que es única, y con valores no-Null ; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador , no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula **IN** para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula **VALUES**. Si se omite la lista de campos, la cláusula **VALUES** debe incluir un valor para cada campo de la tabla, de otra forma fallará **INSERT**.

```
INSERT INTO Clientes SELECT Clientes_Viejos.* FROM Clientes_Nuevos;  
INSERT INTO Empleados (Nombre, Apellido, Cargo) VALUES ('Luis', 'Sánchez', 'Becario');  
INSERT INTO Empleados SELECT Vendedores.* FROM Vendedores WHERE Fecha_Contratacion < Now() - 30;
```

5.3 UPDATE

Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico. Su sintaxis es:

```
UPDATE Tabla SET Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN WHERE Criterio;
```

UPDATE es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez. El ejemplo siguiente incrementa los valores Cantidad pedidos en un 10 por ciento y los valores Transporte en un 3 por ciento para aquellos que se hayan enviado al Reino Unido.:

```
UPDATE Pedidos SET Pedido = Pedidos * 1.1, Transporte = Transporte * 1.03 WHERE PaisEnvío = 'ES';
```

UPDATE no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

```
UPDATE Empleados SET Grado = 5 WHERE Grado = 2;
```

```
UPDATE Productos SET Precio = Precio * 1.1 WHERE Proveedor = 8 AND Familia = 3;
```

Si en una consulta de actualización suprimimos la cláusula WHERE todos los registros de la tabla señalada serán actualizados.

```
UPDATE Empleados SET Salario = Salario * 1.1
```

6.- TIPOS DE DATOS

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos.

Tipos de datos primarios:

Tipo de Datos Longitud Descripción

BINARY 1 byte Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.

BIT 1 byte Valores Si/No ó True/False

BYTE 1 byte Un valor entero entre 0 y 255.

COUNTER 4 bytes Un número incrementado automáticamente (de tipo Long)

CURRENCY 8 bytes Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.

DATETIME 8 bytes Un valor de fecha u hora entre los años 100 y 9999.

SINGLE 4 bytes Un valor en punto flotante de precisión simple con un rango de -3.402823*1038 a -1.401298*10-45 para valores negativos, 1.401298*10-45 a 3.402823*1038 para valores positivos, y 0.

DOUBLE 8 bytes Un valor en punto flotante de doble precisión con un rango de -1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos, 4.94065645841247*10-324 a 1.79769313486232*10308 para valores positivos, y 0.

SHORT 2 bytes Un entero corto entre -32,768 y 32,767.

LONG 4 bytes Un entero largo entre -2,147,483,648 y 2,147,483,647.

LONGTEXT 1 byte por carácter De cero a un máximo de 1.2 gigabytes.

LONGBINARY Según se necesite De cero 1 gigabyte. Utilizado para objetos OLE.

TEXT 1 byte por carácter De cero a 255 caracteres.

La siguiente tabla recoge los sinónimos de los tipos de datos definidos:

Tipo de Dato Sinónimos

BINARY VARBINARY

BIT BOOLEAN

LOGICAL

LOGICAL1

YESNO

BYTE INTEGER1

COUNTER AUTOINCREMENT

CURRENCY MONEY

DATETIME DATE

TIME

TIMESTAMP

SINGLE FLOAT4

IEEEFLOAT4

REAL

DOUBLE FLOAT

```

FLOAT8
IEEEDOUBLE
NUMBER
NUMERIC
SHORT INTEGER2
SMALLINT
LONG INT
INTEGER
INTEGER4
LONGBINARY GENERAL
OLEOBJECT
LONGTEXT LONGCHAR
MEMO
NOTE
TEXT ALPHANUMERIC
CHAR
CHARACTER
STRING
VARCHAR
VARIANT (No Admitido) VALUE

```

7.- SUBCONSULTAS

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta. Puede utilizar tres formas de sintaxis para crear una subconsulta: comparación [ANY | ALL | SOME] (instrucción sql) expresión [NOT] IN (instrucción sql) [NOT] EXISTS (instrucción sql)

En donde:

comparación

Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

expresión

Es una expresión por la que se busca el conjunto resultante de la subconsulta.

instrucción sql

Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre

paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING. Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento.:

```

SELECT * FROM Productos WHERE PrecioUnidad > ANY
(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);

```

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo. El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento.:

```

SELECT * FROM Productos WHERE IDProducto IN
(SELECT IDProducto FROM DetallePedido WHERE Descuento >= 0.25);

```

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título. A la tabla Empleados se le ha dado el alias T1:

```

SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1 WHERE Salario >= (SELECT Avg(Salario) FROM Empleados WHERE T1.Titulo = Empleados.Titulo) ORDER BY Titulo;

```

En el ejemplo anterior , la palabra reservada AS es opcional.

```

SELECT Apellidos, Nombre, Cargo, Salario FROM Empleados WHERE Cargo LIKE "Agente Ven*" AND Salario > ALL (SELECT Salario FROM Empleados WHERE (Cargo LIKE "*Jefe*") OR (Cargo LIKE "*Director*"));

```

Obtiene una lista con el nombre, cargo y salario de todos los agentes de ventas cuyo salario es mayor que el de todos los jefes y directores.

SELECT DISTINCTROW NombreProducto, Precio_Unidad FROM Productos WHERE (Precio_Unidad = (SELECT Precio_Unidad FROM Productos WHERE Nombre_Producto = "Almíbar anisado"));

Manual de SQL

Obtiene una lista con el nombre y el precio unitario de todos los productos con el mismo precio que el almíbar anisado.

SELECT DISTINCTROW Nombre_Contacto, Nombre_Compañía, Cargo_Contacto, Telefono FROM Clientes WHERE (ID_Cliente IN (SELECT DISTINCTROW ID_Cliente FROM Pedidos WHERE Fecha_Pedido >= #04/1/93# <#07/1/93#));

Obtiene una lista de las compañías y los contactos de todos los clientes que han realizado un pedido en el segundo trimestre de 1993.

SELECT Nombre, Apellidos FROM Empleados AS E WHERE EXISTS (SELECT * FROM Pedidos AS O WHERE O.ID_Empresa = E.ID_Empresa);

Selecciona el nombre de todos los empleados que han reservado al menos un pedido.

SELECT DISTINCTROW Pedidos.Id_Producto, Pedidos.Cantidad, (SELECT DISTINCTROW Productos.Nombre FROM Productos WHERE Productos.Id_Producto = Pedidos.Id_Producto) AS ElProducto FROM Pedidos WHERE Pedidos.Cantidad > 150 ORDER BY Pedidos.Id_Producto;

Recupera el Código del Producto y la Cantidad pedida de la tabla pedidos, extrayendo el nombre del producto de la tabla de productos.

8 - CONSULTAS DE REFERENCIAS CRUZADAS

Una consulta de referencias cruzadas es aquella que nos permite visualizar los datos en filas y en columnas, estilo tabla, por ejemplo:

Producto / Año 1996 1997

Pantalones	1.250	3.000
Camisas	8.560	1.253
Zapatos	4.369	2.563

Si tenemos una tabla de productos y otra tabla de pedidos, podemos visualizar en total de productos pedidos por año para un artículo determinado, tal y como se visualiza en la tabla anterior. La sintaxis para este tipo de consulta es la siguiente:

TRANSFORM función agregada instrucción select PIVOT campo pivot [IN (valor1[, valor2[, ...]])]

En donde:

función agregada

Es una función SQL agregada que opera sobre los datos seleccionados.

instrucción select

Es una instrucción SELECT.

campo pivot

Es el campo o expresión que desea utilizar para crear las cabeceras de la columna en el resultado de la consulta.

valor1, valor2

Son valores fijos utilizados para crear las cabeceras de la columna.

Para resumir datos utilizando una consulta de referencia cruzada, se seleccionan los valores de los campos o expresiones especificadas como cabeceras de columnas de tal forma que pueden verse los datos en un formato más compacto que con una consulta de selección.

TRANSFORM es opcional pero si se incluye es la primera instrucción de una cadena SQL. Precede a la instrucción SELECT que especifica los campos utilizados como encabezados de fila y una cláusula GROUP BY que especifica el agrupamiento de las filas. Opcionalmente puede incluir otras cláusulas como por ejemplo WHERE, que especifica una selección adicional o un criterio de ordenación .

Los valores devueltos en campo pivot se utilizan como encabezados de columna en el resultado de la consulta. Por ejemplo, al utilizar las cifras de ventas en el mes de la venta como pivot en una consulta de referencia cruzada se crearán 12 columnas. Puede restringir el campo pivot para crear encabezados a partir de los valores fijos (valor1, valor2) listados en la cláusula opcional IN. También puede incluir valores fijos, para los que no existen datos, para crear columnas adicionales.

Ejemplos

TRANSFORM Sum(Cantidad) AS Ventas SELECT Producto, Cantidad FROM Pedidos WHERE Fecha Between #01-01-98# And #12-31-98# GROUP BY Producto ORDER BY Producto PIVOT DatePart("m", Fecha);

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por mes para un año específico. Los meses aparecen de izquierda a derecha como columnas y los nombres de los productos aparecen de arriba hacia abajo como filas.

TRANSFORM Sum(Cantidad) AS Ventas SELECT Compañía FROM Pedidos WHERE Fecha Between #01-01-98# And #12-31-98# GROUP BY Compañía ORDER BY Compañía PIVOT "Trimestre" & DatePart("q", Fecha) In ('Trimestre1', 'Trimestre2', 'Trimestre 3', 'Trimestre 4');

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por trimestre de cada proveedor en el año indicado. Los trimestres aparecen de izquierda a derecha como columnas y los nombres de los proveedores aparecen de arriba hacia abajo como filas.

Un caso práctico: Se trata de resolver el siguiente problema: tenemos una tabla de productos con dos campos, el código y el nombre del producto, tenemos otra tabla de pedidos en la que anotamos el código del producto, la fecha del pedido y la cantidad pedida. Deseamos consultar los totales de producto por año, calculando la media anual de ventas.

Estructura y datos de las tablas:

1. Artículos:

ID	Nombre
1	Zapatos
2	Pantalones
3	Blusas
2.	Pedidos:

ID	Fecha	Cantidad
1	11/11/1996	250
2	11/11/1996	125
3	11/11/1996	520
1	12/10/1996	50
2	04/05/1996	250
3	05/08/1996	100
1	01/01/1997	40
2	02/08/1997	60
3	05/10/1997	70
1	12/12/1997	8
2	15/12/1997	520
3	17/10/1997	1250

Para resolver la consulta planteamos la siguiente consulta:

```
TRANSFORM Sum(Pedidos.Cantidad) AS Resultado
SELECT Nombre AS Producto, Pedidos.Id AS Código, Sum(Pedidos.Cantidad) AS TOTAL,
Avg(Pedidos.Cantidad) AS Media
FROM Pedidos
INNER JOIN Artículos ON Pedidos.Id = Artículos.Id
GROUP BY Pedidos.Id, Artículos.Nombre
PIVOT Year(Fecha);
```

y obtenemos el siguiente resultado:

Producto	Código	TOTAL	Media	1996	1997
Zapatatos	1	348		87	300
Pantalones	2	955		238,75	375
Blusas	3	1940		485	620

Comentarios a la consulta:

La cláusula TRANSFORM indica el valor que deseamos visualizar en las columnas que realmente pertenecen a la consulta, en este caso 1996 y 1997, puesto que las demás columnas son opcionales.

SELECT especifica el nombre de las columnasopcionales que deseamos visualizar, en este caso Producto, Código, Total y Media, indicando el nombre del campo que deseamos mostrar en cada columna o el valor de la misma. Si incluimos una función de cálculo el resultado se hará en base a los datos de la fila actual y no al total de los datos.

FROM especifica el origen de los datos. La primera tabla que debe figurar es aquella de donde deseamos extraer los datos, esta tabla debe contener al menos tres campos, uno para los títulos de la fila, otros para los títulos de la columna y otro para calcular el valor de las celdas. En este caso en concreto se deseaba visualizar el nombre del producto, como el tabla de pedidos sólo figuraba el código del mismo se añadió una nueva columna en la cláusula select llamada Producto que se corresponda con el campo Nombre de la tabla de artículos. Para vincular el código del artículo de la tabla de pedidos con el nombre del mismo de la tabla artículos se insertó la cláusula INNER JOIN.

La cláusula GROUP BY especifica el agrupamiento de los registros, contrariamente a los manuales de instrucción esta cláusula no es opcional ya que debe figurar siempre y debemos agrupar los registros por el campo del cual extraemos la información. En este caso existen dos campos del cual extraemos la información: pedidos.cantidad y artículos.nombre, por ellos agrupamos por los campos.

Para finalizar la cláusula PIVOT indica el nombre de las columnas no opcionales, en este caso 1996 y 1997 y como vamos a el dato que aparecerá en las columnas, en este caso empleamos el año en que se produjo el pedido, extrayéndolo del campo pedidos.fecha.

Otras posibilidades de fecha de la cláusula pivot son las siguientes:

1. Para agrupamiento por Trimestres
PIVOT "Tri" & DatePart("q", [Fecha]);
Manual de SQL
2. Para agrupamiento por meses (sin tener en cuenta el año)
PIVOT Format([Fecha], "mmm") In ("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic");
3. Para agrupar por días
PIVOT Format([Fecha], "Short Date");

9.- CONSULTAS DE UNIÓN INTERNAS

Las vinculaciones entre tablas se realiza mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2
```

En donde:

tb1, tb2 Son los nombres de las tablas desde las que se combinan los registros.

campo1, campo2 Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

Comp. Es cualquier operador de comparación relacional : =, <, >, <=, >=, o <>.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoría:

```
SELECT Nombre_Categoría, NombreProducto FROM Categorías INNER JOIN Productos ON Categorías.IDCategoría = Productos.IDCategoría;
```

En el ejemplo anterior, IDCategoría es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción SELECT. Para incluir el campo combinado, incluir el nombre del campo en la instrucción SELECT, en este caso, Categorías.IDCategoría. También se pueden enlazar varias cláusulas ON en una instrucción JOIN, utilizando la sintaxis siguiente:

```
SELECT campos FROM tabla1 INNER JOIN tabla2 ON tb1.campo1 comp tb2.campo1 AND ON tb1.campo2 comp tb2.campo2) OR ON tb1.campo3 comp tb2.campo3];
```

También puede anidar instrucciones JOIN utilizando la siguiente sintaxis:

```
SELECT campos FROM tb1 INNER JOIN (tb2 INNER JOIN [( ]tb3 [INNER JOIN [( ]tablaX [INNER JOIN ...]) ON tb3.campo3 comp tbx.campox]) ON tb2.campo2 comp tb3.campo3) ON tb1.campo1 comp tb2.campo2;
```

Manual de SQL

Un LEFT JOIN o un RIGHT JOIN puede anidarse dentro de un INNER JOIN, pero un INNER JOIN no puede anidarse dentro de un LEFT JOIN o un RIGHT JOIN.

Ejemplo

```
SELECT DISTINCTROW Sum([Precio unidad] * [Cantidad]) AS [Ventas], [Nombre] & " " & [Apellidos] AS [Nombre completo] FROM [Detalles de pedidos], Pedidos, Empleados, Pedidos INNER JOIN [Detalles de pedidos] ON Pedidos. [ID de pedido] = [Detalles de pedidos].[ID de pedido], Empleados INNER JOIN Pedidos ON Empleados.[ID de empleado] = Pedidos.[ID de empleado] GROUP BY [Nombre] & " " & [Apellidos];
```

Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y Empleados. Esto es necesario ya que la tabla Empleados no contiene datos de ventas y la tabla Detalles de pedidos no contiene datos de los empleados. La consulta produce una lista de empleados y sus ventas totales. Si empleamos la cláusula INNER en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de INNER JOIN que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave INNER, estas cláusulas son LEFT y RIGHT.

LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la derecha.

RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

10.- CONSULTAS DE UNIÓN EXTERNAS

Se utiliza la operación UNION para crear una consulta de unión, combinando los resultados de dos o más consultas o tablas independientes. Su sintaxis es:

```
[TABLE] consulta1 UNION [ALL] [TABLE] consulta2 [UNION [ALL] [TABLE] consultan [ ... ]]
```

En donde:

consulta1, consulta2, consultan

Son instrucciones SELECT, el nombre de una consulta almacenada o el nombre de una tabla almacenada precedido por la palabra clave TABLE.

Puede combinar los resultados de dos o más consultas, tablas e instrucciones SELECT, en cualquier orden, en una única operación UNION. El ejemplo siguiente combina una tabla existente llamada Nuevas Cuentas y una instrucción SELECT: TABLE [Nuevas Cuentas] UNION ALL SELECT * FROM Clientes WHERE [Cantidad pedidos] > 1000;

Si no se indica lo contrario, no se devuelven registros duplicados cuando se utiliza la operación UNION, no obstante puede incluir el predicado ALL para asegurar que se devuelven todos los registros. Esto hace que la consulta se ejecute más rápidamente. Todas las consultas en una operación UNION deben pedir el mismo número de campos, no obstante los campos no tienen porqué tener el mismo tamaño o el mismo tipo de datos.

Se puede utilizar una cláusula GROUP BY y/o HAVING en cada argumento consulta para agrupar los datos devueltos.

Puede utilizar una cláusula ORDER BY al final del último argumento consulta para visualizar los datos devueltos en un orden específico.

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE País = 'Brasil' UNION SELECT [Nombre de compañía], Ciudad FROM Clientes WHERE País = "Brasil"
```

Recupera los nombres y las ciudades de todos proveedores y clientes de Brasil

SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE País = 'Brasil' UNION SELECT [Nombre de compañía], Ciudad FROM Clientes WHERE País = 'Brasil' ORDER BY Ciudad
 Recupera los nombres y las ciudades de todos proveedores y clientes radicados en Brasil, ordenados por el nombre de la ciudad

SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE País = 'Brasil' UNION SELECT [Nombre de compañía], Ciudad FROM Clientes WHERE País = 'Brasil' UNION SELECT [Apellidos], Ciudad FROM Empleados WHERE Región = 'América del Sur'
 Recupera los nombres y las ciudades de todos los proveedores y clientes de brasil y los apellidos y las ciudades de todos los empleados de América del Sur

TABLE [Lista de clientes] UNION TABLE [Lista de proveedores]

Recupera los nombres y códigos de todos los proveedores y clientes

11.- ESTRUCTURAS DE LAS TABLAS

11.1 Creación de Tablas Nuevas

Si se está utilizando el motor de datos de Microsoft para acceder a bases de datos access, sólo se puede emplear esta instrucción para crear bases de datos propias de access. Su sintaxis es:

`CREATE TABLE tabla (campo1 tipo (tamaño) índice1 , campo2 tipo (tamaño) índice2 , ..., índice multicampo , ...)`

En donde:

Parte Descripción

tabla Es el nombre de la tabla que se va a crear.

campo1

campo2

Es el nombre del campo o de los campos que se van a crear en la nueva tabla. La nueva tabla debe contener, al menos, un campo. tipo Es el tipo de datos de campo en la nueva tabla. (Ver Tipos de Datos)

tamaño

Es el tamaño del campo sólo se aplica para campos de tipo texto.

índice1

índice2

Es una cláusula CONSTRAINT que define el tipo de índice a crear. Esta cláusula es opcional.

índice multicampos Es una cláusula CONSTRAINT que define el tipo de

índice multicampos a crear. Un índice multi campo es aquel que está indexado por el contenido de varios campos. Esta cláusula es opcional.

`CREATE TABLE Empleados (Nombre TEXT (25) , Apellidos TEXT (50));`

Crea una nueva tabla llamada Empleados con dos campos, uno llamado Nombre de tipo texto y longitud 25 y otro llamado apellidos con longitud 50.

`CREATE TABLE Empleados (Nombre TEXT (10), Apellidos TEXT, Fecha_Nacimiento DATETIME) CONSTRAINT IndiceGeneral UNIQUE ([Nombre], [Apellidos], [Fecha_Nacimiento]);`

Crea una nueva tabla llamada Empleados con un campo Nombre de tipo texto y longitud 10, otro con llamado Apellidos de tipo texto y longitud predeterminada (50) y uno más llamado Fecha_Nacimiento de tipo Fecha/Hora. También crea un índice único (no permite valores repetidos) formado por los tres campos.

`CREATE TABLE Empleados (ID INTEGER CONSTRAINT IndicePrimario PRIMARY, Nombre TEXT, Apellidos TEXT, Fecha_Nacimiento DATETIME);`

Crea una tabla llamada Empleados con un campo Texto de longitud predeterminada (50) llamado Nombre y otro igual llamado Apellidos, crea otro campo llamado Fecha_Nacimiento de tipo Fecha/Hora y el campo ID de tipo entero el que establece como clave principal.

11.2 La cláusula CONSTRAINT

Se utiliza la cláusula CONSTRAINT en las instrucciones ALTER TABLE y CREATE TABLE para crear o eliminar índices.

Existen dos sintaxis para esta cláusula dependiendo si desea Crear ó Eliminar un índice de un único campo o si se trata de un campo multiíndice. Si se utiliza el motor de datos de Microsoft, sólo podrá utilizar esta cláusula con las bases de datos propias de dicho motor.

Para los índices de campos únicos:

`CONSTRAINT nombre {PRIMARY KEY | UNIQUE | REFERENCES tabla externa [(campo externo1, campo externo2)]}`

Para los índices de campos múltiples:

`CONSTRAINT nombre {PRIMARY KEY (primario1[, primario2 [, ...]]) | UNIQUE (único1[, único2 [, ...]]) | FOREIGN KEY (ref1[, ref2 [, ...]]) REFERENCES tabla externa [(campo externo1 [,campo externo2 [, ...]])]}`

Parte Descripción

nombre Es el nombre del índice que se va a crear.

primarioN Es el nombre del campo o de los campos que forman el índice primario.

únicoN Es el nombre del campo o de los campos que forman el índice de clave única.

refN Es el nombre del campo o de los campos que forman el índice externo (hacen referencia a campos de otra tabla).

tabla externa Es el nombre de la tabla que contiene el campo o los campos referenciados en refN

campos externos

Es el nombre del campo o de los campos de la tabla externa especificados por ref1, ref2, ..., refN

Si se desea crear un índice para un campo cuando se está utilizando las instrucciones ALTER TABLE o CREATE TABLE

la cláusula CONSTRAINT debe aparecer inmediatamente después de la especificación del campo indexado.

Si se desea crear un índice con múltiples campos cuando se está utilizando las instrucciones ALTER TABLE o CREATE

TABLE la cláusula CONSTRAINT debe aparecer fuera de la cláusula de creación de tabla.

Tipo de Índice Descripción

Manual de SQL

UNIQUE Genera un índice de clave única. Lo que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados.

PRIMARY KEY Genera un índice primario el campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede contener una única clave principal.

FOREIGN KEY Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa fueran los campos referenciados.

11.3 Creación de Índices

Si se utiliza el motor de datos Jet de Microsoft sólo se pueden crear índices en bases de datos del mismo motor. La sintaxis para crear un índice es la siguiente:

`CREATE [UNIQUE] INDEX índice ON tabla (campo [ASC|DESC][, campo [ASC|DESC], ...]) [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]`

En donde:

Parte Descripción

índice Es el nombre del índice a crear.

tabla Es el nombre de una tabla existente en la que se creará el índice.

campo Es el nombre del campo o lista de campos que constituyen el índice.

ASC|DESC Indica el orden de los valores de los campos

ASC indica un orden ascendente (valor predeterminado) y DESC un orden descendente.

UNIQUE Indica que el índice no puede contener valores duplicados.

DISALLOW NULL Prohibe valores nulos en el índice

IGNORE NULL Excluye del índice los valores nulos incluidos en los campos que lo componen.

PRIMARY Asigna al índice la categoría de clave principal, en cada tabla sólo puede existir un único índice que sea "Clave Principal". Si un índice es clave principal implica que no puede contener valores nulos ni duplicados.

Se puede utilizar CREATE INDEX para crear un pseudo índice sobre una tabla adjunta en una fuente de datos ODBC tal como SQL Server que no tenga todavía un índice. No necesita permiso o tener acceso a un servidor remoto para crear un pseudo índice, además la base de datos remota no es consciente y no es afectada por el pseudo índice. Se utiliza la misma sintaxis para las tablas adjuntas que para las originales. Esto es especialmente útil para crear un índice en una tabla que sería de sólo lectura debido a la falta de un índice.

`CREATE INDEX Milindice ON Empleados (Prefijo, Telefono);`

Crea un índice llamado Milindice en la tabla empleados con los campos Prefijo y Telefono.

`CREATE UNIQUE INDEX Milindice ON Empleados (ID) WITH DISALLOW NULL;`

Crea un índice en la tabla Empleados utilizando el campo ID, obligando que el campo ID no contenga valores nulos ni repetidos.

11.4 Modificar el Diseño de una Tabla

Modifica el diseño de una tabla ya existente, se pueden modificar los campos o los índices existentes. Su sintaxis es:

`ALTER TABLE tabla {ADD {COLUMN tipo de campo[(tamaño)] [CONSTRAINT índice] CONSTRAINT índice multicampo} | DROP {COLUMN campo | CONSTRAINT nombre del índice}}`

En donde:

Parte Descripción

tabla Es el nombre de la tabla que se desea modificar.

campo Es el nombre del campo que se va a añadir o eliminar.

tipo Es el tipo de campo que se va a añadir.

tamaño El tamaño del campo que se va a añadir (sólo para campos de texto).

índice Es el nombre del índice del campo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.

índice multicampo Es el nombre del índice del campo multicampo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.

Operación Descripción

ADD COLUMN Se utiliza para añadir un nuevo campo a la tabla, indicando el nombre, el tipo de campo y opcionalmente el tamaño (para campos de tipo texto).

ADD Se utiliza para agregar un índice de multicampos o de un único campo.

DROP COLUMN Se utiliza para borrar un campo. Se especifica únicamente el nombre del campo.

DROP Se utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada CONSTRAINT.

ALTER TABLE Empleados ADD COLUMN Salario CURRENCY;

Agrega un campo Salario de tipo Moneda a la tabla Empleados.

ALTER TABLE Empleados DROP COLUMN Salario; Elimina el campo Salario de la tabla Empleados.

ALTER TABLE Pedidos ADD CONSTRAINT RelacionPedidos FOREIGN KEY (ID_Empleado) REFERENCES Empleados (ID_Empleado);

Agrega un índice externo a la tabla Pedidos. El índice externo se basa en el campo ID_Empleado y se refiere al campo ID_Empleado de la tabla Empleados. En este ejemplo no es necesario indicar el campo junto al nombre de la tabla en la cláusula REFERENCES, pues ID_Empleado es la clave principal de la tabla Empleados.

ALTER TABLE Pedidos DROP CONSTRAINT RelacionPedidos;

Elimina el índice de la tabla Pedidos.

12.- CONSULTAS CON PARÁMETROS

Las consultas con parámetros son aquellas cuyas condiciones de búsqueda se definen mediante parámetros. Si se ejecutan directamente desde la base de datos donde han sido definidas aparecerá un mensaje solicitando el valor de cada uno de los parámetros. Si deseamos ejecutarlas desde una aplicación hay que asignar primero el valor de los parámetros y después ejecutarlas. Su sintaxis es la siguiente:

PARAMETERS nombre1 tipo1, nombre2 tipo2, ... , nombreN tipoN Consulta

En donde:

<PRIVATE>Parte

Descripción

nombre Es el nombre del parámetro

tipo Es el tipo de datos del parámetro

consulta Una consulta SQL

Puede utilizar nombre pero no tipo de datos en una cláusula WHERE o HAVING.

PARAMETERS Precio_Minimo Currency, Fecha_Inicio DateTime; SELECT IDPedido, Cantidad FROM Pedidos WHERE Precio > Precio_Minimo AND FechaPedido >= Fecha_Inicio;

Ejemplo:

PARAMETERS [Escriba los Apellidos:] Text; SELECT * FROM Empleados WHERE [Escriba los Apellidos:] = [Apellidos];

La ejecución desde la base de datos solicita al usuario los apellidos del empleado y después muestra los resultados.

13.- BASES DE DATOS EXTERNAS

Para el acceso a bases de datos externas se utiliza la cláusula IN. Se puede acceder a base de datos dBase, Paradox o Btrieve. Esta cláusula sólo permite la conexión de una base de datos externa a la vez. Una base de datos externa es una base de datos que no sea la activa. Aunque para mejorar los rendimientos es mejor adjuntarlas a la base de datos actual y trabajar con ellas.

Para especificar una base de datos que no pertenece a Access Basic, se agrega un punto y coma (;) al nombre y se encierra entre comillas simples. También puede utilizar la palabra reservada DATABASE para especificar la base de datos externa. Por ejemplo, las líneas siguientes especifican la misma tabla:

FROM Tabla IN '[dBASE IV; DATABASE=C:\DBASE\DATOS\VENTAS;]';
FROM Tabla IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV,';

Acceso a una base de datos externa de Microsoft Access:

SELECT IDCliente FROM Clientes IN MISDATOS.MDB WHERE IDCliente Like 'A*';

En donde MISDATOS.MDB es el nombre de una base de datos de Microsoft Access que contiene la tabla Clientes.

Acceso a una base de datos externa de dBASE III o IV:

```
SELECT IDCliente FROM Clientes IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV'; WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de dBASE III+ hay que utilizar 'dBASE III+' en lugar de 'dBASE IV':

Acceso a una base de datos de Paradox 3.x o 4.x:

```
SELECT IDCliente FROM Clientes IN 'C:\PARADOX\DATOS\VENTAS"Paradox 4.x;" WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de Paradox versión 3.x, hay que sustituir 'Paradox 4.x;' por 'Paradox 3.x;'.

Acceso a una base de datos de Btrieve:

```
SELECT IDCliente FROM Clientes IN 'C:\BTRIEVE\DATOS\VENTAS\FILE.DDF"Btrieve;" WHERE IDCliente Like 'A*';
```

C:\BTRIEVE\DATOS\VENTAS\FILE.DDF es la ruta de acceso y nombre de archivo del archivo de definición de datos de Btrieve.

Manual de SQL

14.- OMITIR LOS PERMISOS DE EJECUCIÓN

En entornos de bases de datos con permisos de seguridad para grupos de trabajo se puede utilizar la cláusula WITH OWNERACCESS OPTION para que el usuario actual adquiera los derechos de propietario a la hora de ejecutar la consulta. Su sintaxis es:

instrucción sql WITH OWNERACCESS OPTION SELECT Apellido, Nombre, Salario FROM Empleados ORDER BY Apellido WITH OWNERACCESS OPTION;

Esta opción requiere que esté declarado el acceso al fichero de grupo de trabajo (generalmente system.mda ó system .mdw) de la base de datos actual.

15.- LA CLÁUSULA PROCEDURE

Esta cláusula es poco usual y se utiliza para crear una consulta a la misma vez que se ejecuta, opcionalmente define los parámetros de la misma. Su sintaxis es la siguiente:

```
PROCEDURE NombreConsulta Parámetro1 tipo1, .... , ParámetroN tipon ConsultaSQL
```

En donde:

Parte Descripción

NombreConsulta Es el nombre con se guardará la consulta en la base de datos.

Parámetro Es el nombre de parámetro o de los parámetros de dicha consulta.

tipo Es el tipo de datos del parámetro

ConsultaSQL Es la consulta que se desea grabar y ejecutar.

```
PROCEDURE Lista_Categorias; SELECT DISTINCTROW Nombre_Categoría, ID_Categoría FROM Categorías ORDER BY Nombre_Categoría;
```

Asigna el nombre Lista_de_categorías a la consulta y la ejecuta.

```
PROCEDURE Resumen Fecha_Inicio DateTime, Fecha_Final DateTime; SELECT DISTINCTROW Fecha_Envio, ID_Pedido, Importe_Pedido, Format(Fecha_Envio, "yyyy") AS Año FROM Pedidos WHERE Fecha_Envio Between Fecha_Inicio And Fecha_Final;
```

Asigna el nombre Resumen a la consulta e incluye dos parámetros.

16.- ANEXOS

16.1 Recuperar Registros de una tabla que no contengan registros relacionados en otra.

Este tipo de consulta se emplea en situaciones tales como saber que productos no se han vendido en un determinado periodo de tiempo, SELECT DISTINCTROW Productos.IdProducto, Productos.Nombre FROM Productos LEFT JOIN Pedidos ON Productos.IdProducto = Pedidos.IdProduct WHERE (Pedidos.IdProducto Is Null) AND (Pedidos.Fecha Between #01-01-98# And #01-30-98#);

La sintaxis es sencilla, se trata de realizar una unión interna entre dos tablas seleccionadas mediante un LEFT JOIN, estableciendo como condición que el campo relacionado de la segunda sea Null.

16.2 Evaluar valores antes de ejecutar la Consulta.

Dentro de una sentencia SQL podemos emplear la función iif para indicar las condiciones de búsqueda. La sintaxis de la función iif es la siguiente:

iif(Expression,Valor1,Valor2)

En donde Expresión es la sentencia que evaluamos; si Expresión es verdadera entonces se devuelve Valor1, si Expresión es falsa se devuelve Valor2.

```
SELECT * Total FROM Empleados WHERE Apellido = iff(TX_Apellido.Text <> "", TX_Apellido.Text, *) ;
```

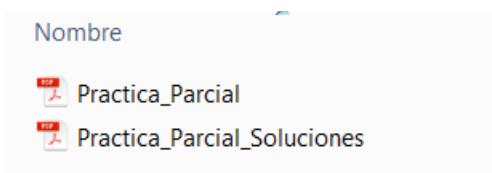
Supongamos que en un formulario tenemos una casilla de texto llamada TX_Apellido. Si cuando ejecutamos esta consulta la casilla contiene algún valor se devuelven todos los empleados cuyo apellido coincide con el texto de la casilla, en caso contrario se devuelven todos los empleados.

```
SELECT Fecha, Producto, Cantidad, (iif(CodigoPostal>=28000A nd CodigoPostal <=28999,'Madrid','Nacional'))A S Destino FROM Pedidos;
```

Esta consulta devuelve los campos Fecha, Nombre del Producto y Cantidad de la tabla pedidos, añadiendo un campo al final con el valor Madrid si el código postal está dentro el intervalo, en caso contrario devuelve Nacional.

42. EXTRA 2 - PRACTICA PRIMER PARCIAL

El siguiente .rar contiene dos archivos pdf, que son los siguientes:



Practica_Parcial.rar

43. EXTRA 3 – PORTAFOLIO EN GIT-HUB

El siguiente PDF contiene un paso a paso de como crear un Portafolio en Git-Hub.



Portafolio_git.pdf

44. EXTRA 4 - PROYECTOS CON EJEMPLOS ÚTILES

Uso de objeto charts	 Charts.rar
InputBox	 InputBox.rar
Reportes a PDF	 PDF_Reports.rar
Struct y Records v2	 Structuras_Recors_v2.rar
TabControl con acceso DB	 TabControl.zip
Validaciones usando máscaras y mucho más	 Validaciones.zip