

## Unidad 03 - Laboratorio DataGridView

### 1. Creación Formulario de Listado

#### Objetivos

Crear el formulario de listado y todos sus controles

#### Aviso

Para realizar este laboratorio necesita utilizar la base de datos academia con la tabla de usuarios. Si se encuentra en el laboratorio del Departamento de Sistemas la misma se encuentra disponible en el servidor del departamento de sistemas serverisi. Si realiza este laboratorio desde otra ubicación antes de realizar los pasos indicados a continuación realice los pasos indicados en el Anexo - Creación de la base de datos

#### Sugerencia

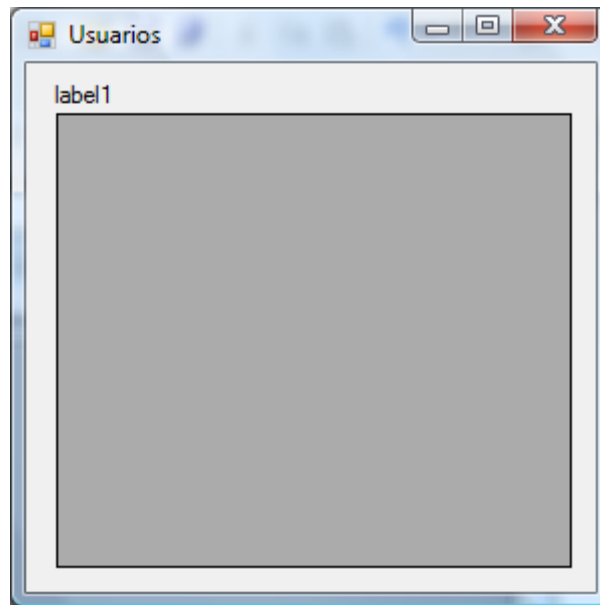
Puede utilizar directamente el proyecto de la capa de presentación del TP2 para realizar este laboratorio.

#### Duración Aproximada

5 minutos

#### Pasos

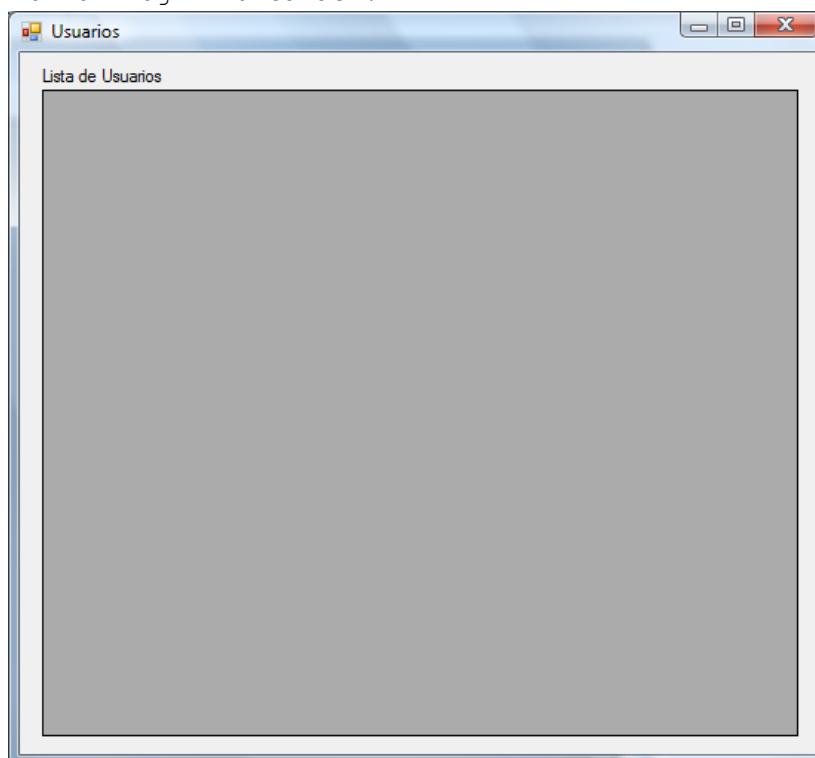
- 1) Crear un nuevo proyecto de Aplicación Windows llamado LabGrilla
- 2) Renombramos el formulario a formListaUsuarios desde el explorador de soluciones.
- 3) Hacemos clic sobre el formulario entonces editamos las siguientes propiedades:
  - (Name): formListaUsuarios
  - Text: Usuarios
- 4) Luego del Cuadro de herramientas arrastramos 1 Label y una DataGridView y los ubicamos de la siguiente forma:



5) Ahora editamos cada control y sus propiedades de la siguiente forma

Control	Propiedad	Valor Modificado
label1	Text	Lista de Usuarios:
DataGridView1	(Name)	dgvUsuarios
	Anchor	Top, Bottom, Left, Right

6) Arrastramos el borde del formulario para agrandar el formulario para que el formulario formLogin luzca así:



## 2. Crear Origen de Datos de la Grilla

### Objetivos

Definir las clases de origen de datos de la Grilla

### Sugerencia

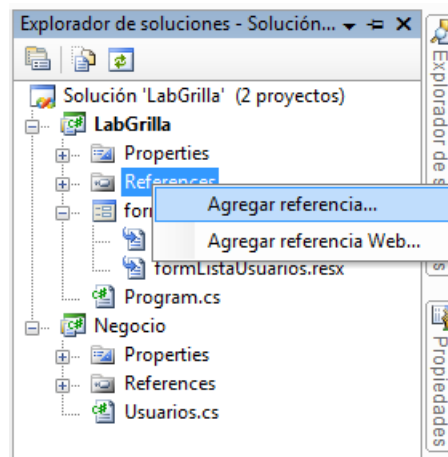
Puede utilizar directamente el proyecto de la capa de negocios del TP2 para realizar este laboratorio.

### Duración Aproximada

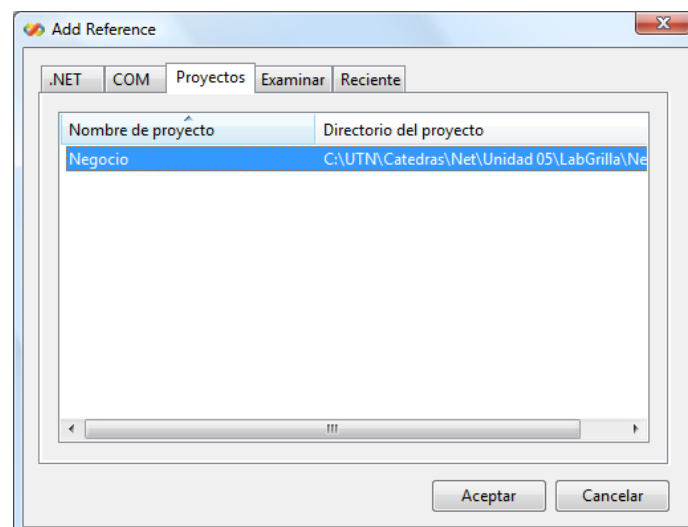
15 minutos

### Pasos

- 1) Crear un nuevo proyecto en la solución de tipo Class Library llamado Negocio (Aquí puede agregar el proyecto de TP2 si prefiere).
- 2) En el explorador de soluciones cambiamos el nombre de Class1 a Usuarios.
- 3) En el proyecto LabGrilla Agregamos una referencia al proyecto Negocio. Para ello hacemos clic con el botón derecho sobre el nodo References del Proyecto LabGrilla y hacemos clic en Agregar Referencia...



Allí vamos a la pestaña Proyectos, seleccionamos el proyecto Negocio y hacemos clic en Aceptar



**Aclaración:** Para simplificar el laboratorio en la clase Usuarios haremos el acceso a datos directamente en el método GetAll pero esto debería invocar a un método de la clase Usuarios del proyecto responsable de acceso a datos.

- 4) En la Clase Usuarios agregamos un using System.Data; y using System.Data.SqlClient;
- 5) Luego agregamos las propiedades daUsuarios y Conn como sigue:

```
protected SqlDataAdapter _daUsuarios;

public SqlDataAdapter daUsuarios
{
    get { return _daUsuarios; }
    set { _daUsuarios = value; }
}

protected SqlConnection _conn;

public SqlConnection Conn
{
    get { return _conn; }
    set { _conn = value; }
}
```

Estas propiedades sólo son necesarias porque no implementamos la capa de acceso a datos por separado.

- 6) Creamos entonces el constructor de la clase Usuarios:

```
public Usuarios()
{
    this.Conn = new SqlConnection(
        "Data Source=serverisi;Initial Catalog=academia;Integrated Security=false;user=net;password=net;");
    /*
     * Este connection string es para conectarse con la base de datos academia en el servidor
     * del departamento sistemas desde una PC de los laboratorios de sistemas,
     * si realiza este Laboratorio desde su PC puede probar el siguiente connection string
     *
     * "Data Source=localhost;Initial Catalog=academia;Integrated Security=true;"
     *
     * Si realiza esta práctica sobre el MS SQL SERVER 2005 Express Edition entonces debe
     * utilizar el siguiente connection string
     *
     * "Data Source=localhost\\SQLEXPRESS;Initial Catalog=academia;Integrated Security=true;"
     */

    this.daUsuarios = new SqlDataAdapter("select * from usuarios", this.Conn);

    this.daUsuarios.UpdateCommand =
        new SqlCommand(" UPDATE usuarios " +
            " SET tipo_doc = @tipo_doc, nro_doc = @nro_doc, fecha_nac = @fecha_nac, " +
            " apellido = @apellido, nombre = @nombre, direccion = @direccion, " +
            " telefono = @telefono, email = @email, celular = @celular, usuario = @usuario, " +
            " clave = @clave WHERE id=@id ", this.Conn);

    this.daUsuarios.UpdateCommand.Parameters.Add("@tipo_doc", SqlDbType.Int, 1, "tipo_doc");
    this.daUsuarios.UpdateCommand.Parameters.Add("@nro_doc", SqlDbType.Int, 1, "nro_doc");
    this.daUsuarios.UpdateCommand.Parameters.Add("@fecha_nac", SqlDbType.DateTime, 1, "fecha_nac");
    this.daUsuarios.UpdateCommand.Parameters.Add("@apellido", SqlDbType.VarChar, 50, "apellido");
    this.daUsuarios.UpdateCommand.Parameters.Add("@nombre", SqlDbType.VarChar, 50, "nombre");
    this.daUsuarios.UpdateCommand.Parameters.Add("@direccion", SqlDbType.VarChar, 50, "direccion");
    this.daUsuarios.UpdateCommand.Parameters.Add("@telefono", SqlDbType.VarChar, 50, "telefono");
    this.daUsuarios.UpdateCommand.Parameters.Add("@email", SqlDbType.VarChar, 50, "email");
    this.daUsuarios.UpdateCommand.Parameters.Add("@celular", SqlDbType.VarChar, 50, "celular");
    this.daUsuarios.UpdateCommand.Parameters.Add("@usuario", SqlDbType.VarChar, 50, "usuario");
    this.daUsuarios.UpdateCommand.Parameters.Add("@clave", SqlDbType.VarChar, 50, "clave");
    this.daUsuarios.UpdateCommand.Parameters.Add("@id", SqlDbType.Int, 1, "id");
}
```

```

this.daUsuarios.InsertCommand =
new SqlCommand(" INSERT INTO usuarios(tipo_doc,nro_doc,fecha_nac,apellido, "+
" nombre,direccion,telefono,email,celular,usuario,clave) "+
" VALUES (@tipo_doc,@nro_doc,@fecha_nac,@apellido,@nombre,@direccion, " +
" @telefono,@email,@celular, @usuario, @clave )", this.Conn);
this.daUsuarios.InsertCommand.Parameters.Add("@tipo_doc", SqlDbType.Int, 1, "tipo_doc");
this.daUsuarios.InsertCommand.Parameters.Add("@nro_doc", SqlDbType.Int, 1, "nro_doc");
this.daUsuarios.InsertCommand.Parameters.Add("@fecha_nac", SqlDbType.DateTime, 1, "fecha_nac");
this.daUsuarios.InsertCommand.Parameters.Add("@apellido", SqlDbType.VarChar, 50, "apellido");
this.daUsuarios.InsertCommand.Parameters.Add("@nombre", SqlDbType.VarChar, 50, "nombre");
this.daUsuarios.InsertCommand.Parameters.Add("@direccion", SqlDbType.VarChar, 50, "direccion");
this.daUsuarios.InsertCommand.Parameters.Add("@telefono", SqlDbType.VarChar, 50, "telefono");
this.daUsuarios.InsertCommand.Parameters.Add("@email", SqlDbType.VarChar, 50, "email");
this.daUsuarios.InsertCommand.Parameters.Add("@celular", SqlDbType.VarChar, 50, "celular");
this.daUsuarios.InsertCommand.Parameters.Add("@usuario", SqlDbType.VarChar, 50, "usuario");
this.daUsuarios.InsertCommand.Parameters.Add("@clave", SqlDbType.VarChar, 50, "clave");

this.daUsuarios.DeleteCommand =
new SqlCommand(" DELETE FROM usuarios WHERE id=@id ", this.Conn);
this.daUsuarios.DeleteCommand.Parameters.Add("@id", SqlDbType.Int, 1, "id");
}

```

- 7) Luego en la clase Usuarios del proyecto Negocio agregamos el Método GetAll() como sigue para obtener la lista de Usuarios con sus datos

```

public DataTable GetAll()
{
    DataTable dtUsuarios = new DataTable();
    this.daUsuarios.Fill(dtUsuarios);
    return dtUsuarios;
}

```

Este método devuelve un DataTable con el resultado de la consulta del SelectCommand.

- 8) Finalmente creamos el método GuardarCambios para aplicar los cambios que realicemos.

```

public void GuardarCambios(DataTable dtUsuarios)
{
    this.daUsuarios.Update(dtUsuarios);
    dtUsuarios.AcceptChanges();
}

```

Este método aplicará los cambios en la base de datos con el Update y con el AcceptChanges le hará que en la DataTable sepa que los cambios que tenía pendientes ya fueron aplicados

### 3. Mostrar datos en grilla y guardar cambios

#### Objetivos

Mostrar los datos en la grilla.

#### Duración Aproximada

5 minutos

#### Pasos

- 1) En la clase formListaUsuarios agregamos una nueva propiedad oUsuarios de tipo Negocio.Usuarios que servirá para recuperar los datos y visualizarlos.

```

public Negocio.Usuarios oUsuarios
{
    get { return _usuarios; }
    set { _usuarios = value; }
}

```

- 2) Luego modificamos el constructor para instanciar oUsuarios y para obtener la DataTable del método GetAll de oUsuarios y mostrarla en la grilla. El constructor entonces queda así:

```

public formListaUsuarios()
{
    InitializeComponent();
    this.oUsuarios = new Negocio.Usuarios();
    this.dgvUsuarios.DataSource = this.oUsuarios.GetAll();
}

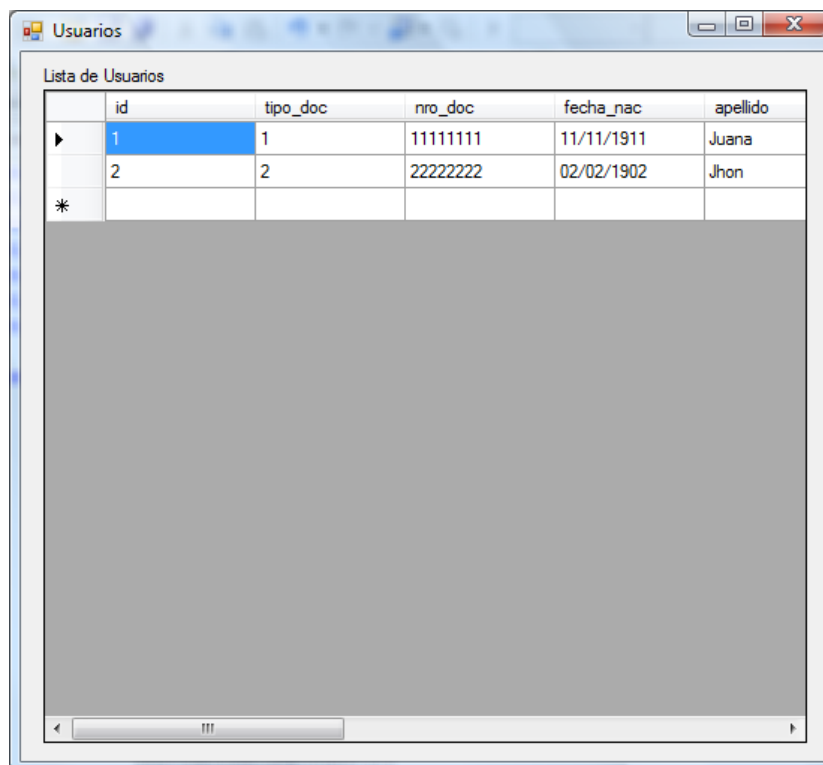
```

La DataGridView tiene una propiedad llamada DataSource (Origen de Datos).

A esta propiedad podemos asignarle cualquier objeto que implemente la interfaz IEnumerable, por ejemplo arreglos, colecciones, listas y DataTable.

Al hacer esto el DataGridView creará una línea para cada elemento de la lista.

- 3) Entonces ejecutamos la aplicación y probamos. Entonces veremos un formulario como el siguiente



- 4) Modificamos el formulario para incluir el botón Guardar y establecemos las siguientes propiedades del mismo:

- (Name): btnGuardar
- Text: Guardar
- Anchor: Bottom, Right

5) Luego agregamos un handler para el evento clic del btnGuardar y en el mismo escribimos el siguiente código:

```
private void btnGuardar_Click(object sender, EventArgs e)
{
    this.GuardarCambios();
    this.RecargarGrilla();
}
```

6) El método guardar cambios invocará al método GuardarCambios de oUsuarios y realizará los cambios en la base de datos. El método RecargarGrilla volverá a invocar al método GetAll de oUsuarios para volver a realizar la consulta de los usuarios y mostrarlos nuevamente así podremos visualizar los id que asigna automáticamente la base de datos.

```
private void RecargarGrilla()
{
    this.dgvUsuarios.DataSource = this.oUsuarios.GetAll();
}

private void GuardarCambios()
{
    this.oUsuarios.GuardarCambios((DataTable)this.dgvUsuarios.DataSource);
}
```

### 3. Editar columnas y propiedades de la grilla.

#### Objetivos

Editar las propiedades de una DataGridView.

Crear columnas de la grilla manualmente por el smart tag y por código y modificar las propiedades de las mismas.

#### Duración Aproximada

20 Minutos

#### Pasos

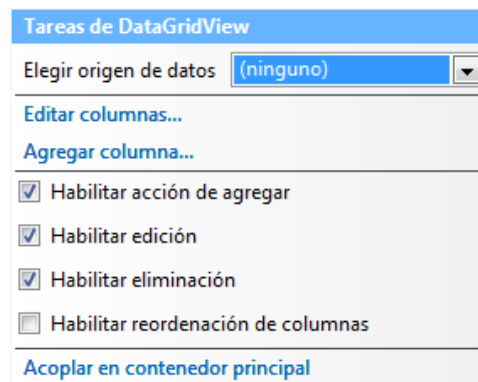
Antes vimos que la grilla permite visualizar todas las columnas del DataTable que se le asigna como DataSource, permite editar todas y en el encabezado (Header) pone el nombre de la columna del DataTable que en el caso de tipo\_doc, nro\_doc y fecha\_nac no es estéticamente bueno.

La causa de ello es que la grilla tiene una propiedad llamada AutoGenerateColumns. Cuando esta propiedad está en true genera automáticamente columnas en función de lo que se asigna en el DataSource.

Esto es un inconveniente ya que a veces debemos mostrar un dato pero no permitir que se modifique o debemos utilizar una columna pero que no es de utilidad para el usuario y por lo tanto no es necesario mostrarla. Otras veces debemos modificar el título de la columna o la letra que utilizamos en ella. Para esto debemos agregar manualmente las columnas a

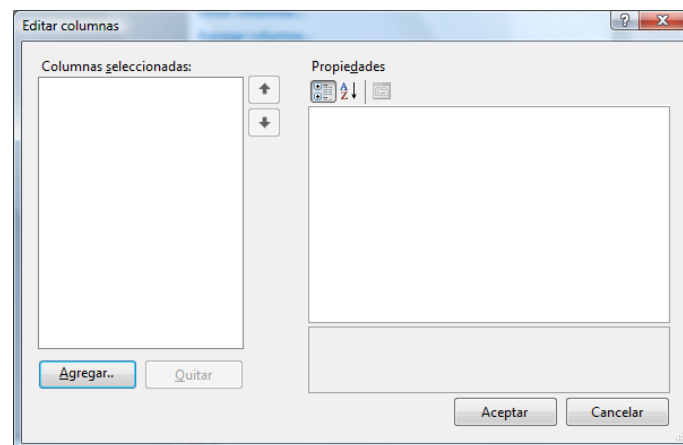
la grilla y luego vincularlas con los datos de la DataTable (DataBinding). Esto podemos realizarlo con el smart tag o manualmente.

- 1) Ir al diseñador de formularios y hacer clic sobre el DataGridView. En la parte superior Derecha aparece un pequeño cuadrado blanco con una flecha negra en su interior. Al hacer clic sobre el mismo se despliega una etiqueta con las tareas más comunes que podemos realizar sobre la DataGridView. Ese es el smart tag



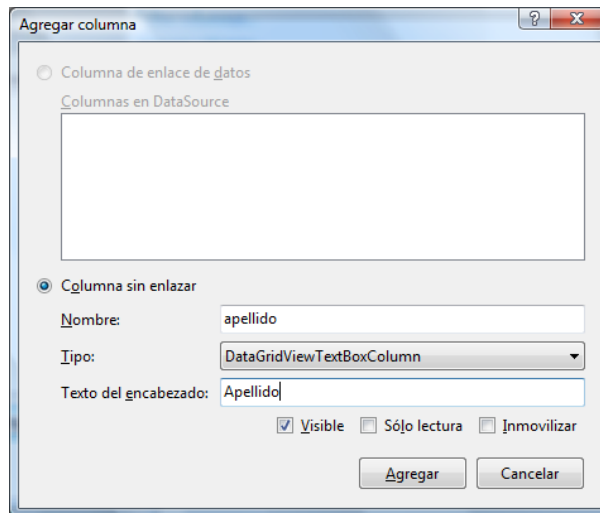
El smart tag está disponible en muchos controles de WinForms y para cada uno muestra distintas tareas, las cuales dependen del control.

- 2) En el smart tag haremos clic en Editar columnas... Entonces se abrirá una nueva ventana para editar cada una de las columnas y sus propiedades más comunes.



- 3) Allí hacemos clic en el botón Agregar... y aparecerá otra ventana.





Entonces indicamos el nombre de la columna, es el nombre que utilizaremos internamente en nuestro código para identificar la columna el usuario no lo verá y no tiene porqué coincidir con el nombre de la columna del DataTable con la cual será enlazada. También debemos ingresar el título de la columna y más importante el tipo de columna.

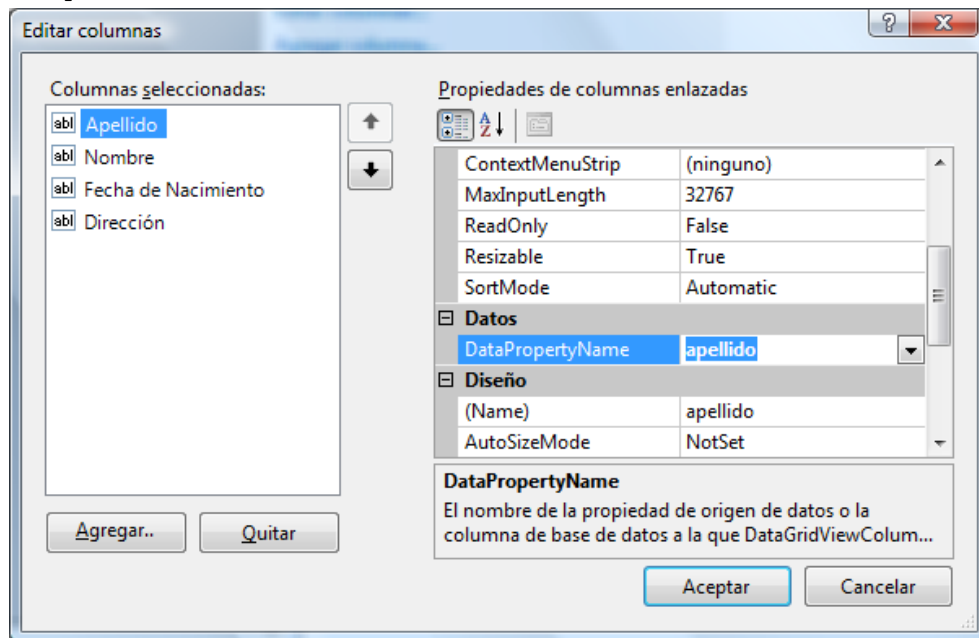
Los tipos de columnas de una grilla que VS 2005 proporciona por defecto son:

- `DataGridViewButtonColumn`: Muestra un botón en cada línea.
- `DataGridViewCheckBoxColumn`: Muestra un checkbox en cada línea.
- `DataGridViewComboBoxColumn`: Muestra una lista desplegable en cada línea. Puede agregarse una lista para todas las filas o una diferente para cada fila además hay varias alternativas para mostrar la lista de elementos que contiene
- `DataGridViewImageColumn`: Muestra un control que puede contener una imagen.
- `DataGridViewLinkColumn`: Muestra un hipervínculo
- `DataGridViewTextBoxColumn`: Muestra un cuadrado de texto donde se puede escribir texto libre pero si la columna de la tabla a la que se enlace es de un tipo de dato diferente al que se intenta ingresar (por ejemplo: columna de tipo entero y se intenta ingresar un texto alfanumérico) no permitirá salir de la celda y producirá un mensaje de error un tanto molesto, pero que puede ser modificado.

Estos son los tipos de columna que ofrece VS 2005 por defecto pero como los tipos de columnas son clases que heredan de `DataGridViewColumn` uno puede crear su propio tipo de columnas y utilizarlas, pero deberá agregarlas a mano.

- 4) Luego de que ingresamos los datos que vimos en la imagen anterior presionamos aceptar y agregamos entonces las columnas Nombre, Fecha de Nacimiento y Dirección de la misma forma, VS 2005 no ofrece un tipo de columna para cargar fechas con un calendario, por lo que también la crearemos de tipo `DataGridViewTextBoxColumn`. Para utilizar una columna que permita mostrar un calendario y cargar una fecha véase <http://msdn.microsoft.com/en-us/library/7tas5c80.aspx>

- 5) Cerramos entonces la ventana para agregar columnas y vemos que las mismas aparecen en la ventana de Editar Columnas



- 6) Al hacer clic sobre una de las columnas que se muestra a la izquierda a la derecha aparece un editor de propiedades de la columna seleccionada. Allí podremos modificar el ancho de la columna (propiedad Width), si permitiremos al usuario ordenar por la columna o no (SortMode), redimensionar la columna (Resizable), el formato de las celdas y muchas cosas más.

Lo que nos importa en esta instancia es DataPropertyName, que nos permitirá vincular la columna de la grilla con una columna de la DataTable que asignamos al DataSource.

Como se ve en la imagen anterior en la columna apellido en DataPropertyName escribimos **apellido** (El nombre de la columna del DataTable con la cual deseamos vincularla).

- 7) Repetimos lo mismo con Nombre, Fecha de Nacimiento y Dirección.
- 8) Ahora si ejecutamos podremos ver que aparecen estas columnas y todas las otras. Posiblemente también aparezcan desordenadas las columnas, esto se debe a la propiedad AutoGenerateColumns de la grilla.

Para resolver este inconveniente tenemos que setear AutoGenerateColumns en false antes de asignarle el valor a la propiedad DataSource.

- 9) Esta propiedad no aparece en la ventana de propiedades, por lo cual debemos modificarla por código. Para ello modificamos el constructor de formListaUsuarios de la siguiente forma:

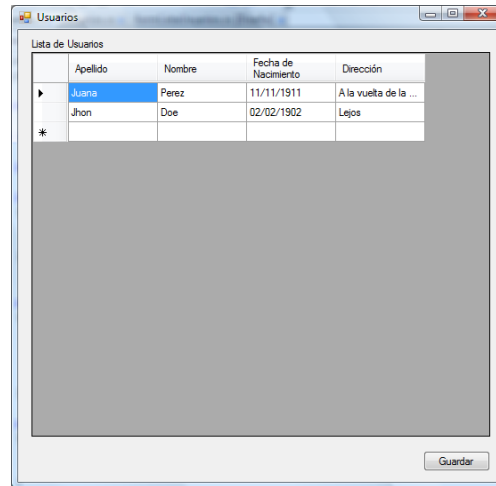
```
public formListaUsuarios()
{
    InitializeComponent();
    this.dgvUsuarios.AutoGenerateColumns = false;
    this.oUsuarios = new Negocio.Usuarios();
}
```

```

        this.dgvUsuarios.DataSource = this.oUsuarios.GetAll();
    }

```

- 10) Volvemos a ejecutar y entonces la grilla sólo mostrará las columnas que agregamos a mano.



- 11) Ahora agregaremos las columnas de Tipo Documento y Nro. Documento a mano.
- 12) Primero crearemos el método GenerarColumnas en formListaUsuarios e invocaremos al mismo en el constructor justo después de setear el AutoGenerateColumns en false.
- 13) El método GenerarColumnas contendrá el siguiente código:

```

private void GenerarColumnas()
{
    //Creando la columna Nro. Documento
    DataGridViewTextBoxColumn colNroDoc = new DataGridViewTextBoxColumn();
    //Creamos la nueva columna y definimos el tipo de columna
    colNroDoc.Name = "nro_doc";
    //asignamos un nombre a la columna
    colNroDoc.HeaderText = "Nro. Documento";
    //indicamos el título a mostrar
    colNroDoc.DataPropertyName = "nro_doc";
    //indicamos con cual columna del DataTable que asignamos al
    //DataSource de la grilla debe vincularse
    colNroDoc.DisplayIndex = 0;
    // en que posición debe mostrarse, todas las columnas a la derecha
    // de la posición que indiquemos se moverán una posición a la derecha

    this.dgvUsuarios.Columns.Add(colNroDoc);
    //agregamos la columna al DataGridView para que la muestre
}

```

- 14) Ejecutamos y probamos. Ahora en el método GenerarColumnas crearemos también la columna Tipo Documento pero de tipo ComboBoxColumn con el siguiente código

```

DataGridViewComboBoxColumn colTipoDoc = new DataGridViewComboBoxColumn();
colTipoDoc.Name = "tipo_doc";
colTipoDoc.HeaderText = "Tipo Documento";
colTipoDoc.DataPropertyName = "tipo_doc";

```

```
colTipoDoc.DisplayIndex = 0;
this.dgvUsuarios.Columns.Add(colTipoDoc);
```

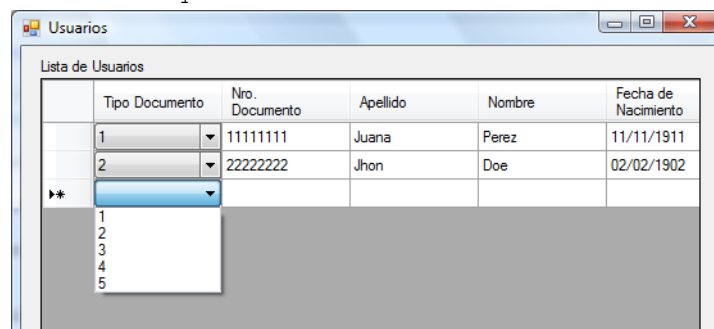
- 15) Si tratáramos de ejecutar esto nos daría un error tras otro y la única forma de detener la ejecución sería desde el botón de Detener de Visual Studio. Esto se debe a que las columnas de tipo DataGridViewComboBoxColumn en su configuración por defecto tienen que tener cargados ítems. Así cuando se asigna un valor a la celda este valor sea validado con los ítems que contiene el combo. Al añadir el DataSource a la grilla, como hemos definido el DataPropertyName de esta columna como tipo\_doc tratará de asignarle a las celdas los valores de las celdas de la columna tipo\_doc del DataTable. Para evitar el error podemos modificar el código agregando los ítems que hagan falta:

```
//Creando la columna Tipo Documento
DataGridViewComboBoxColumn colTipoDoc = new DataGridViewComboBoxColumn();
colTipoDoc.Name = "tipo_doc";
colTipoDoc.HeaderText = "Tipo Documento";
colTipoDoc.DataPropertyName = "tipo_doc";
colTipoDoc.DisplayIndex = 0;
```

```
//Agrego items manualmente
colTipoDoc.Items.Add(1);
colTipoDoc.Items.Add(2);
colTipoDoc.Items.Add(3);
colTipoDoc.Items.Add(4);
colTipoDoc.Items.Add(5);
```

```
this.dgvUsuarios.Columns.Add(colTipoDoc);
```

- 16) Ejecutamos y vemos que no hay errores (siempre y cuando no hayamos agregado un valor a tipo\_doc que sea mayor a 5 o menor a 1) y al hacer clic en el combo vemos que se encuentran todos los ítems que agregamos.



Pero esto es una solución bastante patética a nuestro problema, ya que para un usuario que los tipos de documento estén numerados es una molestia y no le resultará útil para nada.

- 17) Lo que necesitamos es asociar al combo una estructura permita mostrar un valor pero internamente asigne uno diferente pero que mantenga una relación entre ambos. Por ejemplo una tabla como la siguiente:

cod_tipo_doc	desc_tipo_doc
1	DNI
2	Cédula
3	Pasaporte

4	Libreta Cívica
5	Libreta Enrolamiento

18) Esta tabla podría obtenerse de una tabla de la base de datos llamado tipos\_documentos de manera similar a la que se utilizó para obtener la lista de usuarios. Pero para simplificar la generaremos manualmente. Para ello crearemos el método getTiposDocumento() en el formListaUsuarios con el siguiente código:

```
private DataTable getTiposDocumento()
{
    //Creo DataTable
    DataTable dtTiposDoc = new DataTable();

    //Agrego columnas al DataTable
    dtTiposDoc.Columns.Add("cod_tipo_doc", typeof(int));
    dtTiposDoc.Columns.Add("desc_tipo_doc", typeof(string));

    //Agrego filas al DataTable
    dtTiposDoc.Rows.Add(new object[] { 1, "DNI" });
    dtTiposDoc.Rows.Add(new object[] { 2, "Cédula" });
    dtTiposDoc.Rows.Add(new object[] { 3, "Pasaporte" });
    dtTiposDoc.Rows.Add(new object[] { 4, "Libreta Cívica" });
    dtTiposDoc.Rows.Add(new object[] { 5, "Libreta Enrolamiento" });

    return dtTiposDoc;
}
```

19) A continuación modificamos el método GenerarColumnas y reemplazamos la parte donde añadíamos los ítems a mano por las siguientes líneas de código:

```
colTipoDoc.DataSource = this.getTiposDocumento();
//Asigno la lista de items que son válidos

colTipoDoc.ValueMember = "cod_tipo_doc";
//indico que el valor interno del combo es el
//valor de la fila elegida y la columna cod_tipo_doc
//del DataSource que asigné a la columna colTipoDoc

colTipoDoc.DisplayMember = "desc_tipo_doc";
//indico que el valor que se muestra al usuario es el
//que se corresponde con la columna desc_tipo_doc
//del DataSource que asigné a colTipoDoc independientemente
//de la columna de la cual obtiene su valor

this.dgvUsuarios.Columns.Add(colTipoDoc);
```

Entonces el código del método GenerarColumnas queda así:

```
private void GenerarColumnas()
{
    //Creando la columna Nro. Documento
    DataGridViewTextBoxColumn colNroDoc = new DataGridViewTextBoxColumn();
    //Creamos la nueva columna y definimos el tipo de columna
    colNroDoc.Name = "nro_doc";
    //asignamos un nombre a la columna
    colNroDoc.HeaderText = "Nro. Documento";
    //indicamos el título a mostrar
    colNroDoc.DataPropertyName = "nro_doc";
    //indicamos con cual columna del DataTable que asignamos al
    //DataSource de la grilla debe vincularse
    colNroDoc.DisplayIndex = 0;
    // en que posición debe mostrarse, todas las columnas a la derecha
    // de la posición que indiquemos se moverán una posición a la derecha

    this.dgvUsuarios.Columns.Add(colNroDoc);
    //agregamos la columna al DataGridView para que la muestre
```

```

//Creando la columna Tipo Documento
DataGridViewComboBoxColumn colTipoDoc = new DataGridViewComboBoxColumn();
colTipoDoc.Name = "tipo_doc";
colTipoDoc.HeaderText = "Tipo Documento";
colTipoDoc.DataPropertyName = "tipo_doc";
colTipoDoc.DisplayIndex = 0;

colTipoDoc.DataSource = this.getTiposDocumento();
//Asigno la lista de items que son válidos

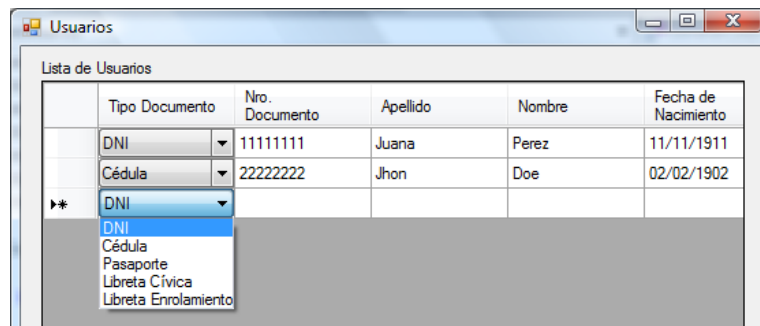
colTipoDoc.ValueMember = "cod_tipo_doc";
//indico que el valor interno del combo es el
//valor de la fila elegida y la columna cod_tipo_doc
//del DataSource que asigné a la columna colTipoDoc

colTipoDoc.DisplayMember = "desc_tipo_doc";
//indico que el valor que se muestra al usuario es el
//que se corresponde con la columna desc_tipo_doc
//del DataSource que asigné a colTipoDoc independientemente
//de la columna de la cual obtiene su valor

this.dgvUsuarios.Columns.Add(colTipoDoc);
}

```

- 20) Ejecutamos y vemos que es más fácil para el usuario utilizar el sistema así.



Si agregamos un nuevo usuario y elegimos con el combo el tipo de documento adecuado, rellenamos los campos y guardamos en la base de datos podremos ver que se registra el código del tipo de documento asociado al tipo de documento elegido y no el texto

- 21) Por código creamos entonces las columnas de Teléfono (colTel), E-Mail (colEmail), Celular (colCel), Usuario (colUsuario) y Clave (colClave) de forma similar a la columna de Nro. Documento en el método GenerarColumnas

- 22) Además en el método GenerarColumnas haremos las siguientes modificaciones a las propiedades de las mismas:

Columna	Modificación
<b>E-Mail</b>	Ancho 250
<b>Dirección</b>	Ancho 250
<b>Apellido</b>	Negrita
<b>Nombre</b>	Negrita

<b>Nro. Documento</b>	Alineado a la derecha
<b>Clave</b>	Invisible
<b>Fecha de Nacimiento</b>	Alineado a la derecha

Entonces el método GenerarColumnas quedará así:

```
private void GenerarColumnas()
{
    //Creando la columna Nro. Documento
    DataGridViewTextBoxColumn colNroDoc = new DataGridViewTextBoxColumn();
    //Creamos la nueva columna y definimos el tipo de columna
    colNroDoc.Name = "nro_doc";
    //asignamos un nombre a la columna
    colNroDoc.HeaderText = "Nro. Documento";
    //indicamos el título a mostrar
    colNroDoc.DataPropertyName = "nro_doc";
    //indicamos con cual columna del DataTable que asignamos al
    //DataSource de la grilla debe vincularse
    colNroDoc.DisplayIndex = 0;
    // en que posición debe mostrarse, todas las columnas a la derecha
    // de la posición que indiquemos se moverán una posición a la derecha

    this.dgvUsuarios.Columns.Add(colNroDoc);
    //agregamos la columna al DataGridView para que la muestre

    //Creando la columna Tipo Documento
    DataGridViewComboBoxColumn colTipoDoc = new DataGridViewComboBoxColumn();
    colTipoDoc.Name = "tipo_doc";
    colTipoDoc.HeaderText = "Tipo Documento";
    colTipoDoc.DataPropertyName = "tipo_doc";
    colTipoDoc.DisplayIndex = 0;

    colTipoDoc.DataSource = this.getTiposDocumento();
    //Asigno la lista de items que son válidos

    colTipoDoc.ValueMember = "cod_tipo_doc";
    //indico que el valor interno del combo es el
    //valor de la fila elegida y la columna cod_tipo_doc
    //del DataSource que asigné a la columna colTipoDoc

    colTipoDoc.DisplayMember = "desc_tipo_doc";
    //indico que el valor que se muestra al usuario es el
    //que se corresponde con la columna desc_tipo_doc
    //del DataSource que asigné a colTipoDoc independientemente
    //de la columna de la cual obtiene su valor

    this.dgvUsuarios.Columns.Add(colTipoDoc);

    DataGridViewTextBoxColumn colTel = new DataGridViewTextBoxColumn();
    colTel.Name = "telefono";
    colTel.HeaderText = "Teléfono";
    colTel.DataPropertyName = "telefono";

    DataGridViewTextBoxColumn colEmail = new DataGridViewTextBoxColumn();
    colEmail.Name = "email";
    colEmail.HeaderText = "E-Mail";
    colEmail.DataPropertyName = "email";

    DataGridViewTextBoxColumn colCel = new DataGridViewTextBoxColumn();
    colCel.Name = "celular";
    colCel.HeaderText = "Celular";
    colCel.DataPropertyName = "celular";

    DataGridViewTextBoxColumn colUsuario = new DataGridViewTextBoxColumn();
    colUsuario.Name = "usuario";
    colUsuario.HeaderText = "Usuario";
    colUsuario.DataPropertyName = "usuario";

    DataGridViewTextBoxColumn colClave = new DataGridViewTextBoxColumn();
    colClave.Name = "clave";
    colClave.HeaderText = "Clave";
```

```

colClave.DataPropertyName = "clave";

colEmail.Width = 250;
colNroDoc.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
colClave.Visible = false;

//como las columnas direccion, nombre, apellido y fecha de nacimiento las creamos
//con el diseñador de formularios no disponemos de una variable para hacer
//referencia a ellas. Entonces debemos referenciarlas con
//this.dgvUsuarios.Columns["nombre_columna"] donde el nombre_columna es lo que
//indicamos en la propiedad Name de las columnas
this.dgvUsuarios.Columns["direccion"].Width = 250;
this.dgvUsuarios.Columns["apellido"].DefaultCellStyle.Font =
    new Font(this.dgvUsuarios.DefaultCellStyle.Font, FontStyle.Bold);
this.dgvUsuarios.Columns["nombre"].DefaultCellStyle.Font =
    new Font(this.dgvUsuarios.DefaultCellStyle.Font, FontStyle.Bold);
this.dgvUsuarios.Columns["fecha_nac"].DefaultCellStyle.Alignment =
    DataGridViewContentAlignment.MiddleRight;

this.dgvUsuarios.Columns.Add(colTel);
this.dgvUsuarios.Columns.Add(colEmail);
this.dgvUsuarios.Columns.Add(colCel);
this.dgvUsuarios.Columns.Add(colUsuario);
this.dgvUsuarios.Columns.Add(colClave);
}

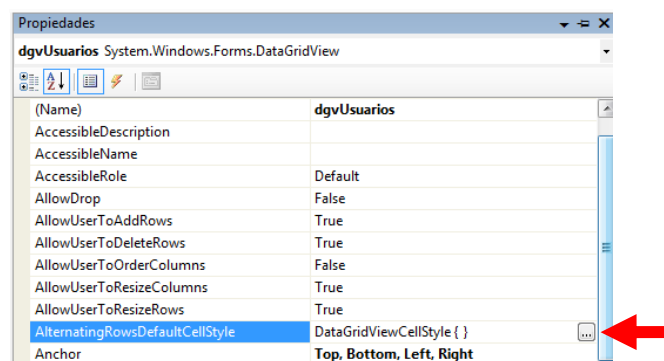
```

Las columnas tienen una propiedad llamada `DefaultCellStyle` (estilo de la celda por defecto) que contiene el estilo gráfico de las celdas de esa columna que se crean con cada nueva fila. Para editar la tipografía (Font), alineación (Alignment) y demás propiedades gráficas de todas las celdas de una columna tenemos que modificar el `DefaultCellStyle` de la columna.

Sin embargo la propiedad `Bold` (negrita) de una tipografía no puede modificarse, entonces lo que debemos hacer es cambiar tipografía por otra que si esté puesta en negrita. Ya que cambiar el estilo de una tipografía es algo común el constructor de la clase `Font` nos ofrece una "solución" a este problema y es que nos permite crear una tipografía en función de otra con un cambio del estilo, en este caso el `Bold`.

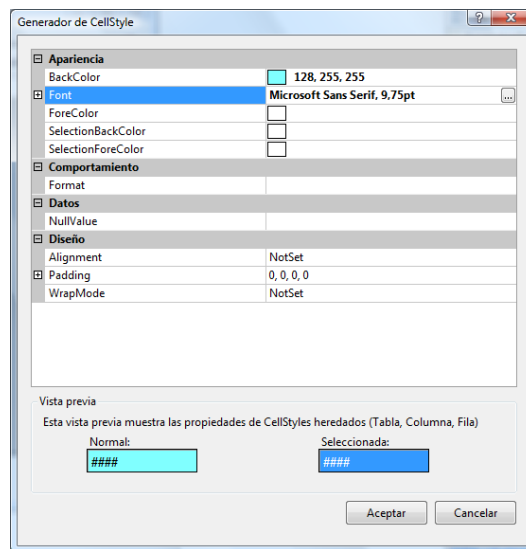
- 23) Ahora modificaremos 2 propiedades bastante útiles de una grilla: `AlternatingRowsDefaultCellStyle` y `EditMode`. `AlternatingRowsDefaultCellStyle` cambia el estilo de las filas pares pero no el de las impares.

Para ello en el diseñador gráfico hacemos clic sobre la grilla y luego en la ventana de propiedades vamos a `AlternatingRowsDefaultCellStyle` y en la parte de la derecha hacemos clic en el botón con los ...

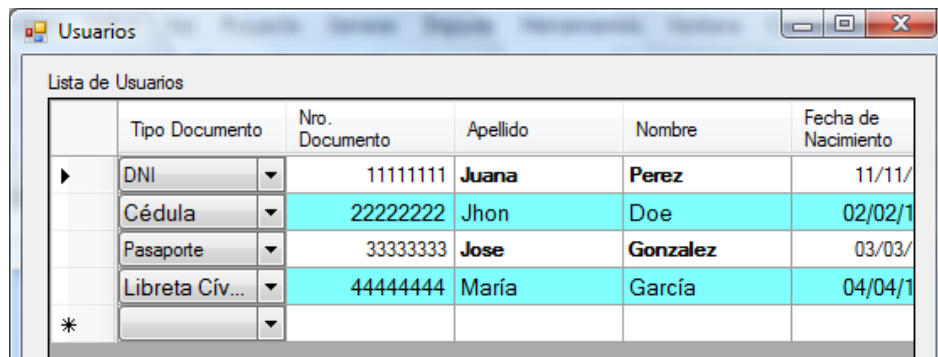




- 24) Aparece una ventana para modificar este estilo de celda y cambiaremos el color de fondo por un celeste (BackColor) y el tamaño de la fuente a 10



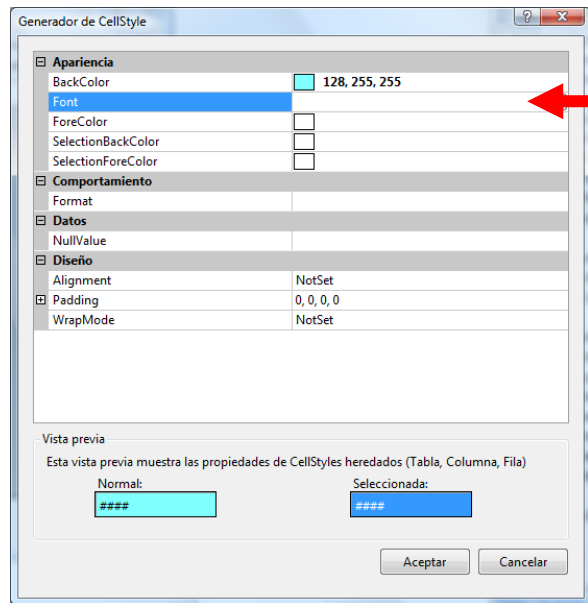
- 25) Ejecutamos y probamos. Vemos que el color de fondo funciona perfectamente pero no así la tipografía



	Tipo Documento	Nro. Documento	Apellido	Nombre	Fecha de Nacimiento
▶	DNI	11111111	Juana	Perez	11/11/
	Cédula	22222222	Jhon	Doe	02/02/1
	Pasaporte	33333333	Jose	Gonzalez	03/03/
	Libreta Cív...	44444444	María	García	04/04/1
*					

Cuando cambiamos la tipografía del `AlternatingRowsDefaultCellStyle` tenemos el problema que esto tiene prioridad por sobre el estilo de la celda de la columna. Por eso perdemos el estilo con negrita en las celdas de las filas pares de las columnas Apellido y Nombre.

- 26) Para volver a la normalidad volvemos al Generador de CellStyle y en el cuadro del Font BORRAMOS el contenido del texto y hacemos clic en Aceptar. Esta es la única forma de que las celdas vuelvan a utilizar el estilo especificado en las columnas. Si tratáramos de volver la tipografía al valor por defecto el editor no lo interpreta como el tipo por defecto sino como que deseamos sobrescribir este valor.



- 27) Probamos y vemos que hemos recuperado el estilo de negrita en las columnas de Apellido y Nombre

	Tipo Documento	Nro. Documento	Apellido	Nombre	Fecha de Nacimiento
▶	DNI	11111111	Juana	Perez	11/11/
	Cédula	22222222	Jhon	Doe	02/02/
	Pasaporte	33333333	Jose	Gonzalez	03/03/
	Libreta Cívica	44444444	Maria	Garcia	04/04/
*					

- 28) Ahora el EditMode. Esta propiedad define como se empieza a editar una celda.

Si ejecutamos y probamos hacer clic en un combo de la columna Tipo Documento que no sea la que estaba seleccionada notarán que deben hacerse 2 clics para que se despliegue la lista de los tipos de documento, si además la lista de un combo estaba desplegada necesitaremos 3 clics para desplegar la lista de otro combo y si hacemos clic sobre una celda queda seleccionada toda la celda y no sólo el texto interno.

Esto se debe a que el EditMode se encuentra en EditOnKeystrokeOrF2. Por ello para que la celda entre en modo de edición se debe empezar a escribir o presionar F2 y en el caso de los ComboBox debemos hacer un clic para seleccionar la celda y otro para acceder al ComboBox

- 29) Entonces vamos al diseñador de formularios hacemos clic en la grilla y en la ventana de propiedades cambiamos el EditMode a EditOnEnter. Así apenas seleccionamos una celda empezamos a editarla. Podremos notarlo seleccionando una celda de tipo texto en lugar de tener seleccionada una celda se selecciona el texto interno. Y si hacemos clic en un combo sin que ninguna lista esté desplegada sólo necesitaremos un clic y estando una lista desplegada necesitaremos 2 clics para desplegar la lista de otro combo.

Esto permite una mayor eficiencia al usuario para trabajar con grillas, pero también significa que cada vez que cambiamos de celda se ejecuta el evento CellBeginEdit. Por lo que si pretendemos realizar alguna acción en este evento, la misma se ejecutará cada vez que cambiemos de celda.

## ANEXO - Creación de la base de datos

### Objetivos

Crear la base de datos para el laboratorio actual.

### Aviso

Esto es necesario sólo para realizar este laboratorio en la PC de cada uno. En el servidor del departamento de sistemas, la misma ya se encuentra disponible.

### Duración Aproximada

10 Minutos

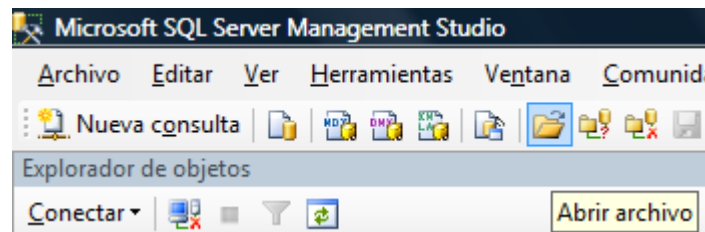
### Pasos

- 1) Abrir el Sql Management Studio
- 2) Conectarse al servidor local con los siguientes datos y haga clic en Conectar

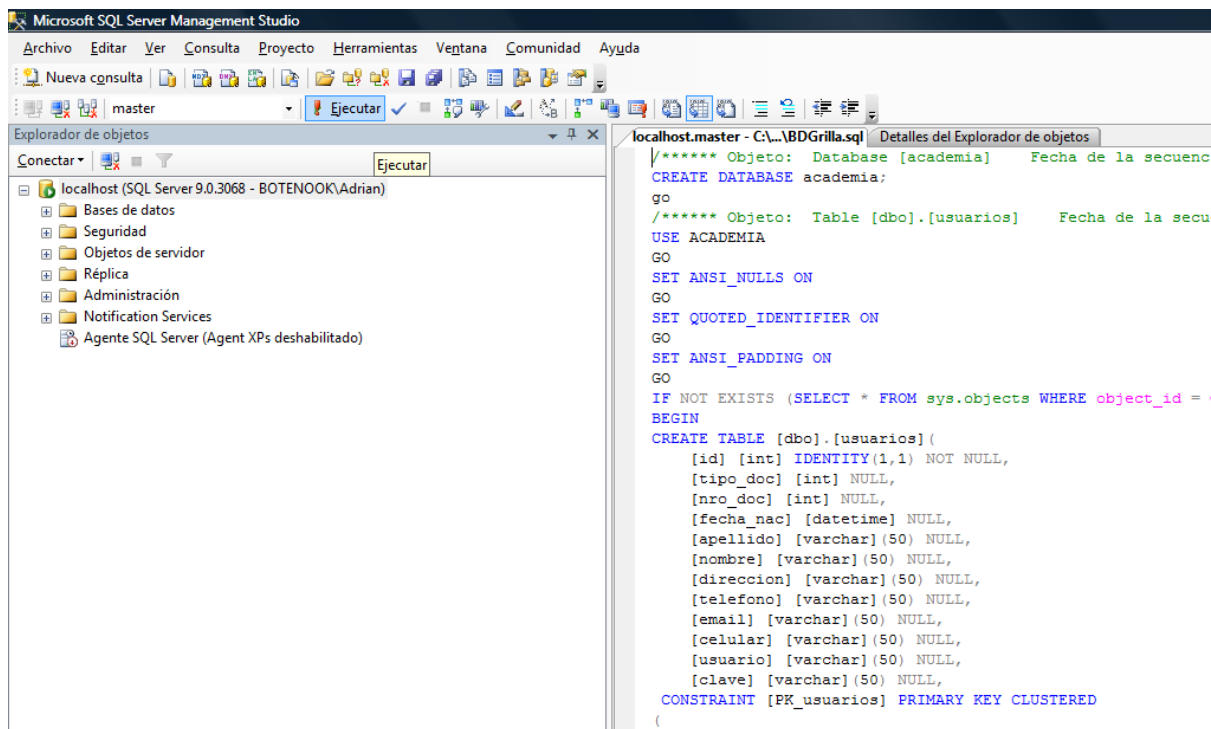


Si utiliza el Sql Express Edition en lugar de localhost debe utilizar localhost\SqExpress

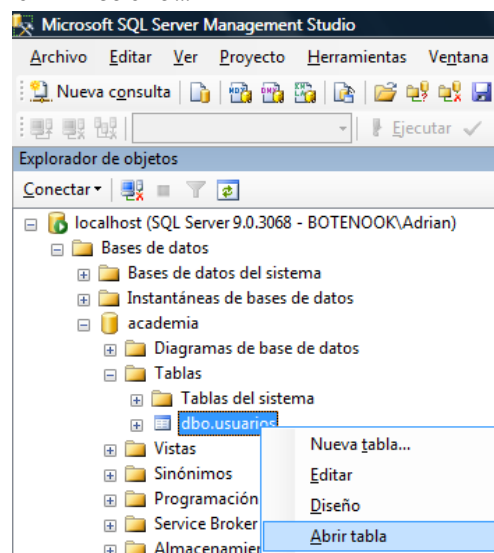
- 3) Clic en el botón Abrir archivo



- 4) Se abrirá una ventana allí elegimos el Archivo DataGridView.sql que está junto a este documento y hacemos clic en aceptar
- 5) Se abrirá allí una pestaña a la derecha con un script de sql para crear la base de datos academia, la tabla usuarios y cargar dos registros. Para ello hacemos clic en ejecutar



- 6) Para comprobar si la base de datos se creó correctamente en el árbol de la izquierda hacemos clic con botón derecho sobre Bases de datos y luego hacemos clic en Actualizar. Entonces debería aparecer la Base de Datos academia.
- 7) Si hacemos clic en el signo + a la izquierda de academia debería aparecer varias subcarpetas, una de ellas llamada Tablas. Allí hacemos clic sobre el signo + a la izquierda y debería aparecer la tabla usuarios.
- 8) Sobre la tabla usuarios hacemos clic con el botón derecho y luego hacemos clic sobre Abrir tabla...



- 9) Entonces se abrirá otra pestaña a la derecha con 2 registros. Si estos registros aparecen la base se ha creado correctamente.

Microsoft SQL Server Management Studio

Archivo Editar Ver Proyecto Diseñador de consultas Herramientas Ventana Comunidad Ayuda

Nueva consulta

Cambiar tipo

Explorador de objetos

Conectar

localhost (SQL Server 9.0.3068 - BOTENOOK\Adrian)

- Bases de datos
  - Bases de datos del sistema
  - Instantáneas de bases de datos
  - academia
    - Diagramas de base de datos
    - Tablas

**BOTENOOK.academia - dbo.usuarios**

	id	tipo_doc	nro_doc	fecha_nac	apellido	nombre
▶	1	1	11111111	11/11/2011 12:...	Juana	Perez
	2	2	22222222	02/02/2002 12:...	Jhon	Doe
*	NULL	NULL	NULL	NULL	NULL	NULL

localhost.academia - C:\...BDGrilla.sql

Detalles del Explorador de objetos