

# Comencemos a programar con VBA - Access

## Entrega 22

### Objetos de Access Formularios (1)

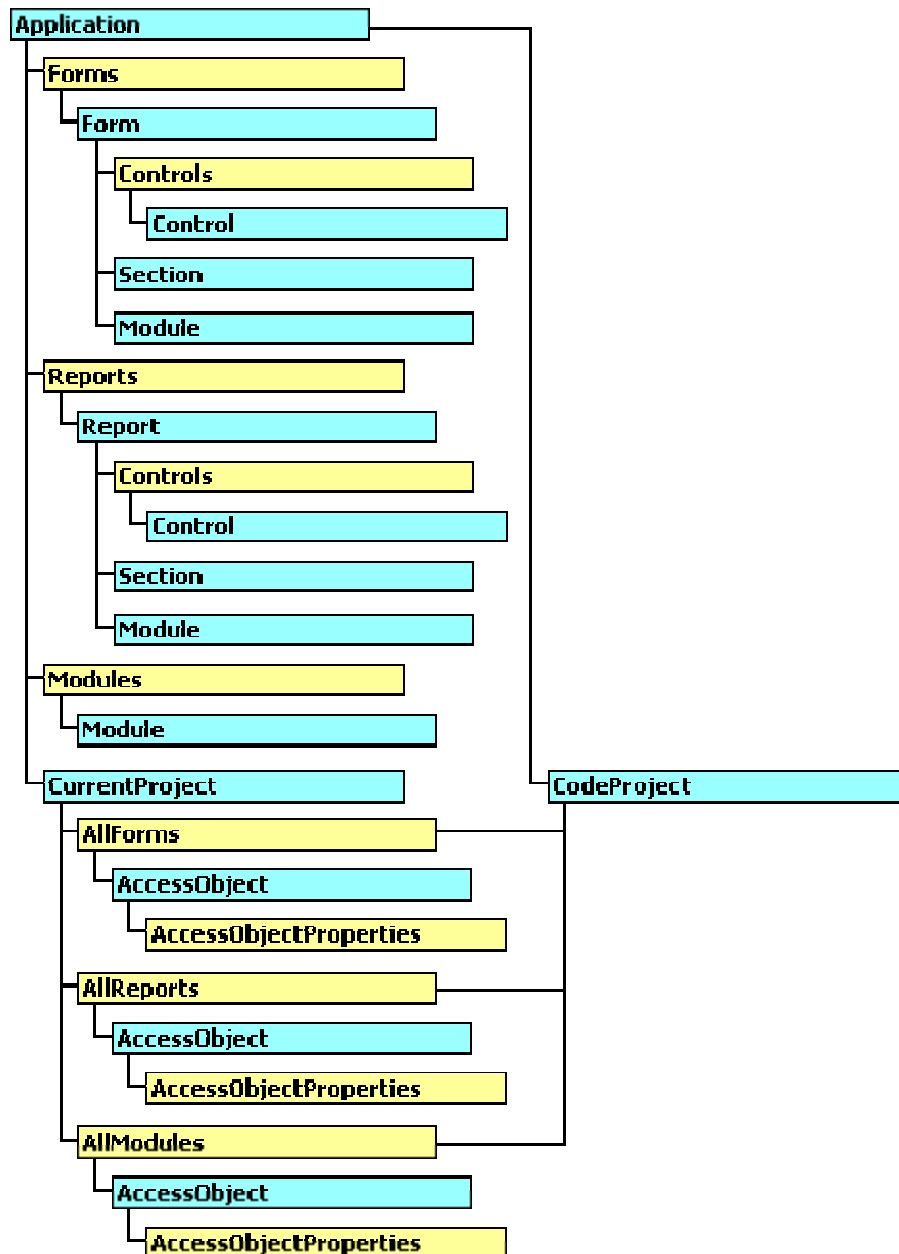
## Estructura de los Objetos de Access

Una aplicación Access está compuesta de una gran cantidad de objetos que se interrelacionan.

Algunos de ellos son objetos individuales, pero otros están agrupados en Colecciones.

En la cúspide de todos ellos está el objeto **Application**

El objeto **Application** hace referencia a la aplicación de Access que está activa en ese momento en el ordenador.



Del objeto **Application** descienden una gran variedad de objetos y colecciones

- **CodeData**: Objetos guardados en la base de datos de códigos por la aplicación (servidor) de origen (Jet o SQL).
- **CodeProject**: Proyecto para la base de datos de códigos de Microsoft. Una de sus colecciones es **AllForms**, que referencia a cada formulario.

- **CurrentData**: Objetos guardados en la base de datos activa.
- **CurrentProject**: Proyecto actual de Access. Una de sus colecciones es **AllForms**.
- **DataAccessPages** (Colección): Páginas de acceso a datos abiertas actualmente.
- **DefaultWebOptions**: Atributos globales en el nivel de aplicación en una página Web.
- **DoCmd**: Objeto para ejecutar las acciones de Microsoft Access desde Visual Basic. (Ya lo hemos visto en el capítulo 16)
- **Forms** (Colección): Formularios abiertos de Access.
- **Modules** (Colección): Módulos estándar y de clase abiertos.
- **Printers** (Colección): Representan a las impresoras disponibles en el sistema.
- **References** (Colección): Las referencias establecidas actualmente.
- **Reports** (Colección): Informes abiertos.
- **Screen**: Formulario, informe o control que tiene el enfoque actualmente.

La estructura de objetos representada en el gráfico anterior es sólo un resumen del total de objetos propios de Access.

Los objetos representados en color amarillo, con nombre en plural, representan colecciones.

Por ejemplo la colección **Forms** contiene objetos del tipo **Form**, que a su vez contiene una colección **Controls** de objetos **Control**.

Un objeto **Control** tiene la colección **Properties** aunque no está representada en el gráfico.

Esta colección, como su nombre indica, contiene las propiedades de ese control.

La colección **Properties** la poseen, no sólo los controles; también la tienen los objetos **Form**, **Subform**, **Report**, **Section**, y como veremos más adelante, los objetos de acceso a datos de **DAO** y **ADO**.

Diferentes objetos pueden tener el mismo tipo de colecciones; por ejemplo vemos que los objetos **Form** y **Report** poseen la colección **Controls** que contiene a sus respectivos controles.

También podemos ver el paralelismo existente entre el objeto **CurrentProject** y **CodeProject**, ambos pertenecientes al objeto **Application**.

En ésta y próximas entregas, estudiaremos los objetos presentados en el gráfico de la página anterior.

El objeto **DoCmd**, no representado en el gráfico, ya lo estudiamos en el capítulo 16, aunque lo seguiremos utilizando en ésta y en las próximas entregas.

## Formularios

De todos los objetos de Access vamos a empezar a trabajar con los formularios.

Un formulario es el elemento básico para la introducción de datos y su mantenimiento, por parte del usuario final de nuestra aplicación.

También puede servir para mostrar avisos, al estilo de un Cuadro de mensaje, como formulario de inicio de una aplicación, como formulario del tipo “Acerca de...” o incluso como contenedor de otro subformulario u objetos ActiveX.

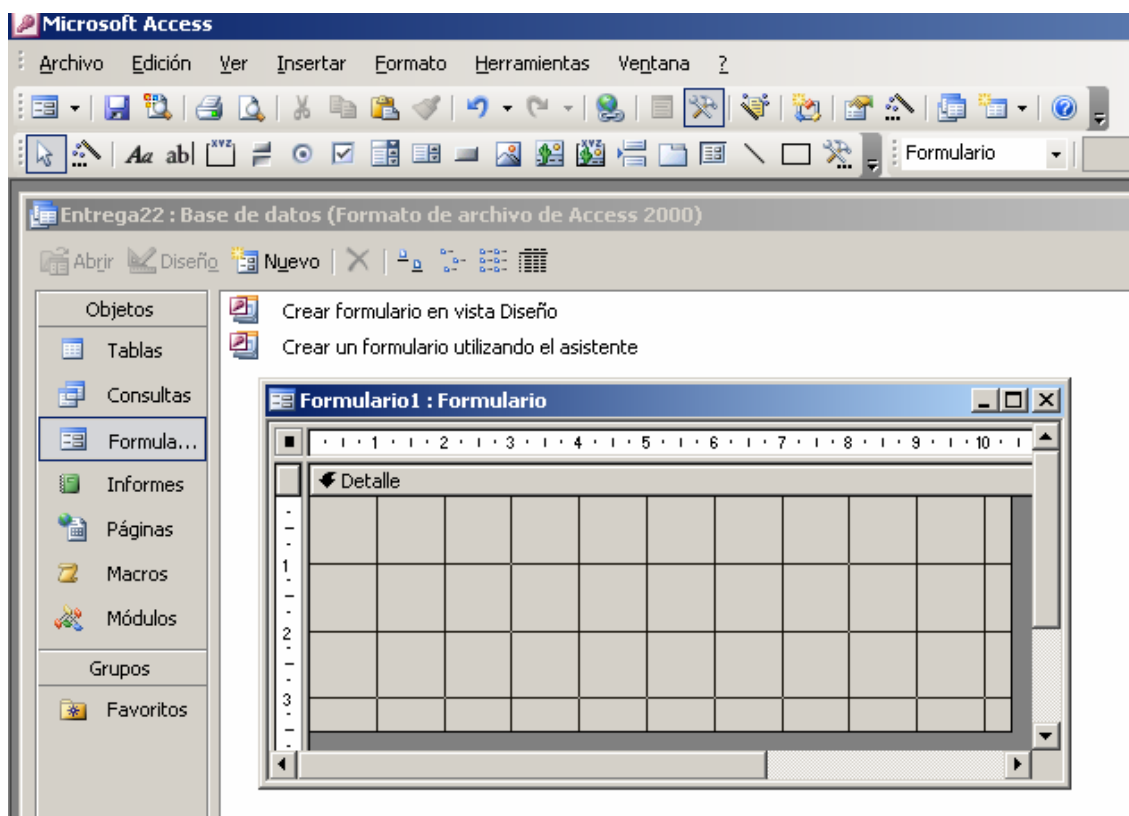
Ya hemos comentado que el objeto **Form**, que representa a un formulario, es un elemento de la colección **Forms**.

Vemos también que un objeto **Form** contiene como atributo el objeto **Module**.

El objeto **Module** representa el módulo de clase del formulario, en el que podemos, al igual que en una “clase normal” definir propiedades y métodos personalizados.

Para efectuar las pruebas de código, vamos a crear un formulario “sencillo” y a trabajar con su código de clase.

Para simplificar más las cosas no lo enlazaremos con ningún origen de datos.

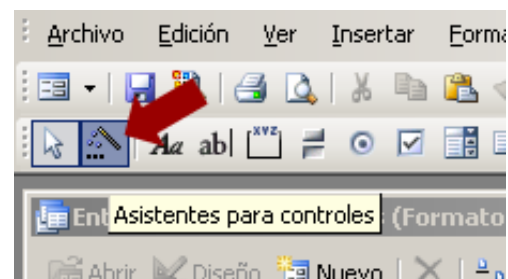


## Eventos de un formulario

Ahora vamos a poner un botón que nos servirá para cerrar el formulario, una vez abierto.

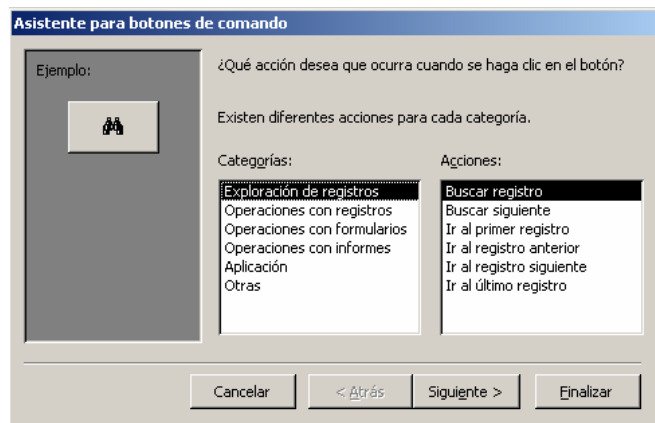
Usaremos el [Asistente para Controles].

Compruebe que está activada la “Varita mágica”

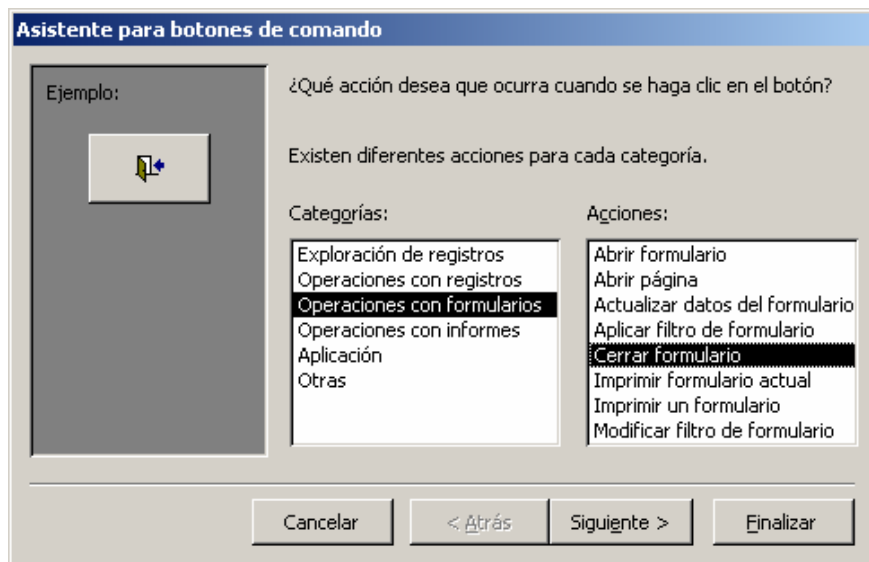


Seleccionamos el Botón de Comando y lo dibujamos en la esquina inferior derecha del formulario.

Tras esto se nos abre el asistente y nos pregunta qué queremos hacer con el botón:



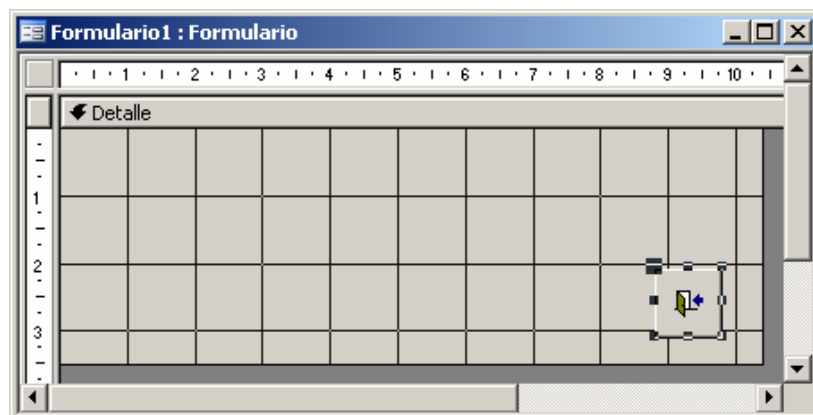
Seleccionamos en la lista izquierda **Operaciones con Formularios**, y en la derecha **Cerrar formulario**.



Como imagen seleccionamos Salir (la puerta con la flecha).

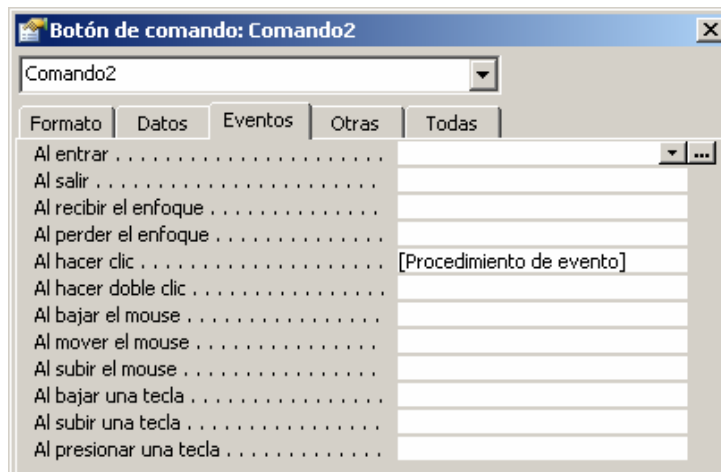
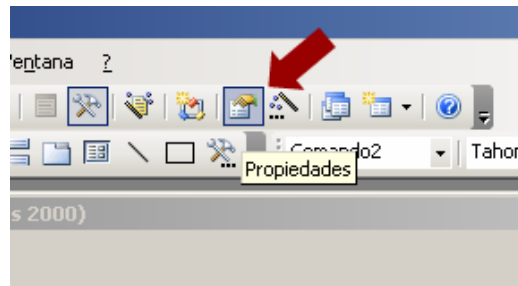
Tras esto pasan dos cosas:

El botón se nos muestra en el formulario, con el gráfico de la puerta y la flecha:



Pero, aparte de esta imagen, más o menos estética, ha ocurrido algo muy importante en el módulo de código asociado al formulario.

Si teniendo seleccionado el botón, abrimos el editor de Propiedades, pulsando el botón [Propiedades] del menú, seleccionamos la pestaña [Eventos] y en ella podemos ver que el evento Al hacer clic tiene asociado un procedimiento de evento en el código de clase asociado al formulario.



Si pinchamos con el ratón en la palabra [Procedimiento de evento], se nos abre un botón con puntos suspensivos a su derecha.

Pulsando en ese botón se nos abriría el editor de código teniendo posicionado el cursor en el procedimiento correspondiente.

También podríamos ver el código pulsando en el botón **[Código]** del menú:



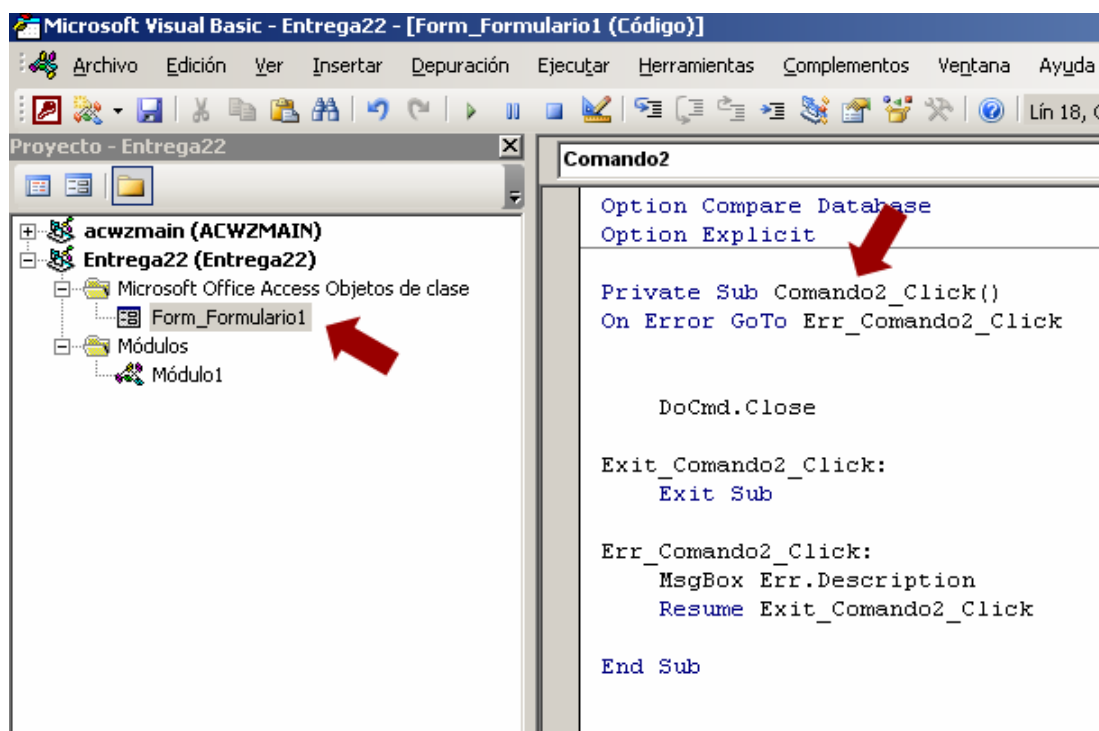
En ambos casos se nos abrirá el editor de código, y en él podremos apreciar una serie de hechos:

En la ventana del **Explorador de proyectos**, vemos que de la carpeta Microsoft Office Objetos de clase, cuelga el objeto **Form\_Formulario1**.

Asociado a este objeto podemos ver el código de su módulo de clase, en el que ha aparecido el procedimiento

```
Sub Comando2_Click ()
```

Este procedimiento se ejecutará cada vez que presionemos sobre el botón de nombre **Comando2**. Este botón es el que hemos colocado en el formulario.



La parte del león de este procedimiento es la línea **DoCmd.Close** que lo que hace es cerrar el formulario actual, como ya vimos en el capítulo 16.

Además, de forma automática se han creado varias líneas de código para la gestión de posibles errores.

Como vimos en el capítulo 11, la primera línea indica la etiqueta de la línea a la que debe saltar el código si se produjera un error; en este caso a la línea **Err\_Comando2\_Click:**

Una vez que salte a esa línea, el código hará que se muestre la descripción del error, tras lo que anulará el error y efectuará un salto a la etiqueta **Exit\_Comando2\_Click:** desde donde saldrá del procedimiento mediante **Exit Sub**.

Podemos cambiar ese código.

Una cosa que no me hace especialmente feliz es el nombre del botón.

En general me gustan más nombres que sean descriptivos, por ejemplo **cmdCerrar**.

Tampoco me gustan las etiquetas del control de errores, que ha creado el asistente por lo que voy a modificar el código para quede así:

```
Private Sub cmdCerrar_Click()
On Error GoTo HayError
    DoCmd.Close
Salir:
    Exit Sub
HayError:
    MsgBox Err.Description
    Resume Salir
End Sub
```

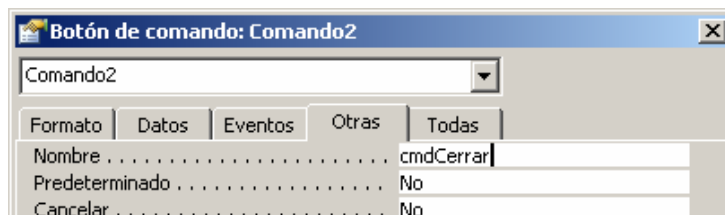
Lo primero que he hecho ha sido cambiar el nombre del procedimiento gestor del evento **Click** del botón. Esto es así porque quiero que el botón se llame **cmdCerrar**. **cmd** indica que es un botón y **Cerrar** indica el procedimiento que ejecuta, Cerrar el formulario.

Al haber eliminado el procedimiento **Comando2\_Click()**, hemos eliminado también el enlace entre el evento Clic del botón y su procedimiento gestor.

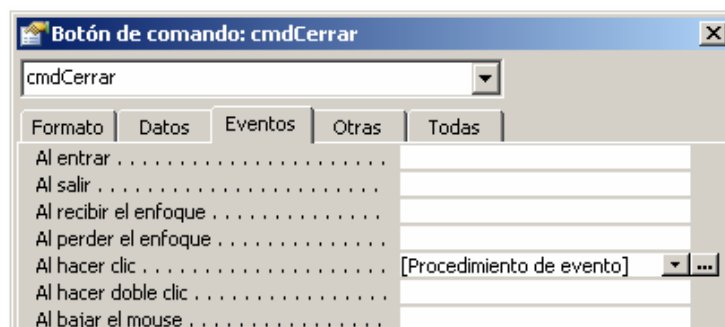
Por lo tanto es algo que deberemos hacer manualmente.

Nos volvemos al diseño del formulario cerrando la ventana del editor de código, por ejemplo pulsando en el aspa superior derecha de la ventana.

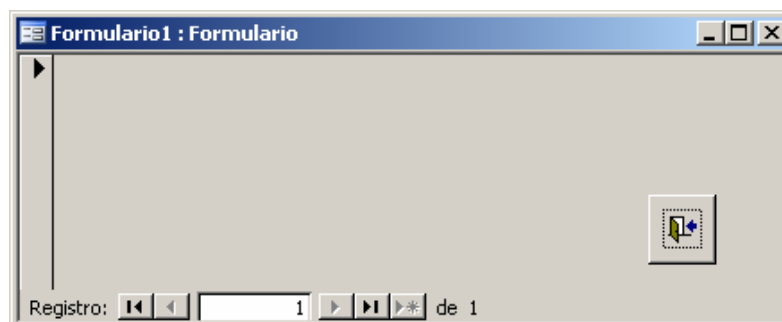
Volvemos a abrir la ventana de propiedades, y lo primero que vamos a hacer es cambiar el nombre del botón, cosa que haremos pulsando en la pestaña [Otras] seleccionando la propiedad Nombre y escribiendo **cmdCerrar**.



A continuación asignamos el procedimiento **cmdCerrar\_Click()** al evento **Al hacer clic** del botón, como hemos visto en un párrafo anterior.



Guardamos los cambios y abrimos el formulario.



No es nada espectacular, pero podemos comprobar que al apretar el botón, el formulario se cierra, que es lo que en principio queríamos.

Durante la vida de un formulario ocurren una serie de eventos.

Es interesante saber en qué orden se producen éstos, tanto al abrirse el formulario, como al cerrarse.

Para ello vamos a seleccionar una serie de eventos del formulario y a escribir un pequeño código que indicará en qué orden se han producido estos eventos.



Primero creamos, a nivel del módulo, una variable que se irá incrementando cuando se produzcan determinados eventos.

Vamos a seleccionar los siguientes eventos del formulario:

- |                     |                      |   |
|---------------------|----------------------|---|
| ▪ <b>Activate</b>   | Al activar           | Cuando el formulario pasa a ser ventana activa    |
| ▪ <b>Close</b>      | Al cerrar            | Al cerrarse y retirarse de la pantalla            |
| ▪ <b>Current</b>    | Al activar registro  | Cuando se enfoca un registro como actual          |
| ▪ <b>Deactivate</b> | Al desactivar        | Cuando la ventana del formulario pierde el foco   |
| ▪ <b>GotFocus</b>   | Al recibir el foco   | Cuando el formulario recibe el foco               |
| ▪ <b>Load</b>       | Al cargar            | Al abrir un formulario y mostrar sus registros    |
| ▪ <b>LostFocus</b>  | Al perder el foco    | Cuando el formulario pierde el foco               |
| ▪ <b>Open</b>       | Al abrir             | Al abrir pero antes de mostrar el primer registro |
| ▪ <b>Resize</b>     | Al cambiar el tamaño | Al abrir un formulario o cambiar de tamaño        |
| ▪ <b>Unload</b>     | Al descargar         | Al cerrar un formulario, antes de desaparecer     |

Vamos asociando cada uno de los eventos y escribimos su código

El código sería tan simple como éste:

```
Option Compare Database
Option Explicit

Dim intNumero As Integer

Private Sub Form_Activate()
    MuestraOrden "Activate"
End Sub

Private Sub Form_Close()
    MuestraOrden "Close"
End Sub

Private Sub Form_Current()
    MuestraOrden "Current"
End Sub

Private Sub Form_Deactivate()
    MuestraOrden "Deactivate"
End Sub

Private Sub Form_GotFocus()
    MuestraOrden "GotFocus"
End Sub

Private Sub Form_Load()
    MuestraOrden "Load"
```

```
End Sub

Private Sub Form_LostFocus()
    MuestraOrden "LostFocus "
End Sub

Private Sub Form_Open(Cancel As Integer)
    MuestraOrden "Open"
End Sub

Private Sub Form_Resize()
    MuestraOrden "Resize"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MuestraOrden "Unload"
End Sub

Private Sub MuestraOrden(ByVal Evento As String)
    intNumero = intNumero + 1
    Debug.Print CStr(intNumero) & " " & Evento
End Sub

Private Sub cmdCerrar_Click()
On Error GoTo HayError
    DoCmd.Close
Salir:
    Exit Sub
HayError:
    MsgBox Err.Description
    Resume Salir
End Sub
```

¿Qué hace este código?

Cuando se produce cualquiera de los eventos incluidos en el mismo, éste llama al procedimiento **MuestraOrden**, pasándole como parámetro el nombre del evento.

Este procedimiento incrementa la variable **intNumero** e imprime su valor, junto con el nombre del evento pasado, en la ventana **Inmediato**.

Con este simple procedimiento podemos averiguar en qué orden se van produciendo los eventos.

Grabamos, abrimos el formulario y lo cerramos presionando el botón de cierre del formulario **cmdCerrar**.

Si abrimos el editor de código para ver qué ha escrito en la ventana **inmediato**, veremos lo siguiente:

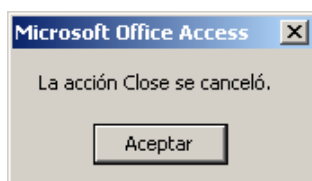
- 1 Open
- 2 Load
- 3 Resize
- 4 Activate
- 5 Current
- 6 Click del botón
- 7 Unload
- 8 Deactivate
- 9 Close

Hasta que presionamos el botón, evento número 6, vemos que el primer evento generado es el **Open**. A continuación el **Load**. El evento **Resize** se genera la primera vez que se “dibuja” el formulario. A continuación se activa el formulario (**Activate**) y por último trata de mostrar el posible registro activo (**Current**).

Tras presionar el botón de cierre, se genera el evento **Unload** a continuación se desactiva el formulario (**Deactivate**), y justo antes de que se cierre y se descargue de memoria, el evento **Close**.

Si nos fijamos en el evento Unload, vemos que incluye el parámetro Cancel.

Si en el evento, asignáramos a la variable cancel el valor **True**, detendríamos la descarga del mismo, y nos mostraría un aviso de que la acción **Close** se canceló.



Esto nos impediría cerrar el formulario.

Para salir del bucle en el que nos metería, podríamos pulsar en el botón diseño del formulario, abrir la ventana de su módulo de clase y eliminar, poner a **False** o dejar comentada la línea

```
' Cancel = True
```

Si nos fijamos en la lista de eventos generados, nos puede sorprender que no se ha generado ni el evento **GotFocus** ni el **LostFocus**.

¿Quiere decir que el formulario, como tal en ningún momento recibe el foco, y por tanto tampoco lo pierde?

Puede sorprender la respuesta, pero en este caso es así.

¿Por qué?

Porque el que recibe el foco es el botón como único control capaz de recibir el foco.

El formulario, al contrario que un Botón de comando, no posee la propiedad **TabStop** (Punto de tabulación) por lo que el botón tiene prioridad a la hora de recibir el foco.

Las secciones del formulario tampoco tienen esta propiedad. El que sí la tiene es un formulario insertado como Subformulario.

¿Qué pasa si en el botón le ponemos la propiedad **Punto de tabulación** al valor no?

En el editor del formulario seleccionamos el botón y ponemos su propiedad a no.

Abrimos y cerramos el formulario; el resultado mostrado por la ventana inmediato es:

- 1 Open
- 2 Load
- 3 Resize
- 4 Activate
- 5 GotFocus
- 6 Current
- 7 Click del botón
- 8 GotFocus
- 9 Unload
- 10 LostFocus
- 11 Deactivate
- 12 Close

Como el botón tiene desactivada la propiedad Punto de tabulación, y no hay ningún otro control que la tenga activada, es el propio formulario el que recibe el foco.

Incluso no se termina de perder totalmente el foco, ya que cuando presionamos el botón, no se genera el evento **LostFocus**, aunque sí se vuelve a generar inmediatamente después el evento **GotFocus**.

Si no nos damos cuenta de lo dicho en los puntos anteriores, podríamos tener la sorpresa de que no se ejecutara el código que diseñáramos para el evento **GotFocus** o **LostFocus** del formulario.

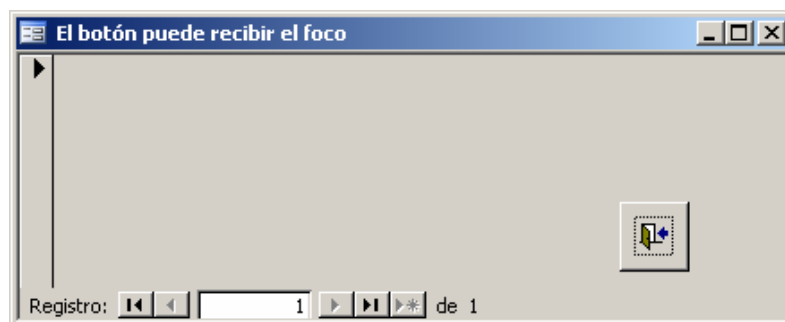
Para dejar el formulario como estaba ponemos volvemos a restituir el valor de la propiedad **TabStop** del botón a **True**, pero esta vez lo vamos a hacer mediante código.

Para ello aprovecharemos el gestor del primer evento que se produce en el formulario, en concreto el evento **Open**.

Su código quedará así:

```
Private Sub Form_Open(Cancel As Integer)
    MuestraOrden "Open"
    cmdCerrar.TabStop = True
    Caption = " El botón puede recibir el foco"
End Sub
```

Abrimos el formulario y lo volvemos a cerrar.



Si lo observamos vemos ahora, que en vez del anterior título **Formulario1: Formulario** aparece el mensaje **El botón puede recibir el foco**.

Si miramos lo que ha escrito en la ventana Inmediato, veremos que formulario ya no recibe el foco y se generan los 9 eventos iniciales, en vez de los 12 que se generaban con la propiedad **TabStop** puesta a false en el botón.

La propiedad **Caption** del formulario, de lectura y escritura, devuelve ó establece el texto que se muestra en la barra de título del formulario.

Esta propiedad controla el texto que aparece en objetos como formularios, botones de comando, páginas de objetos **TabControl** o controles ficha, etiquetas, botones de alternar e informes.

Como vemos, podemos cambiar, de una forma sencilla, las propiedades de un formulario y sus controles, en tiempo de ejecución.

Esto nos abre inmensas posibilidades para el diseño de nuestras aplicaciones.

## Crear propiedades y métodos públicos en el módulo de clase del formulario.

Hemos dicho que el código asociado a un formulario es su código de clase, por lo tanto podemos escribir nuestras propias propiedades y métodos en él.

Vamos a crear la propiedad **Numero** que contendrá un dato numérico que asociaremos al formulario.

Para ello crearemos un nuevo formulario al que llamaremos **FormularioNumerado**.

Le pondremos un botón de cierre, como en el formulario anterior, con su mismo nombre y gestor del evento Al hacer Clic.

Como inciso comentaré que la clase del formulario que acabamos de crear tiene por nombre **Form\_FormularioNumerado**

En el código del módulo de la clase le pondremos una variable privada de tipo numérico y crearemos la propiedad **Numero**.

El código de este nuevo formulario será:

```
Option Compare Database
Option Explicit

Dim intNumero As Integer

Public Property Get Numero() As Integer
    Numero = intNumero
End Property

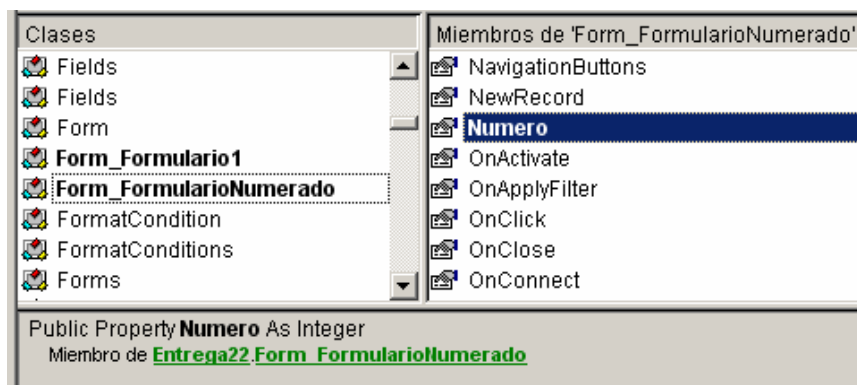
Public Property Let Numero(ByVal NuevoNumero As Integer)
    intNumero = NuevoNumero
    Caption = "Formulario N° " & _
        & Format(.Numero, "000")
End Property
```

```

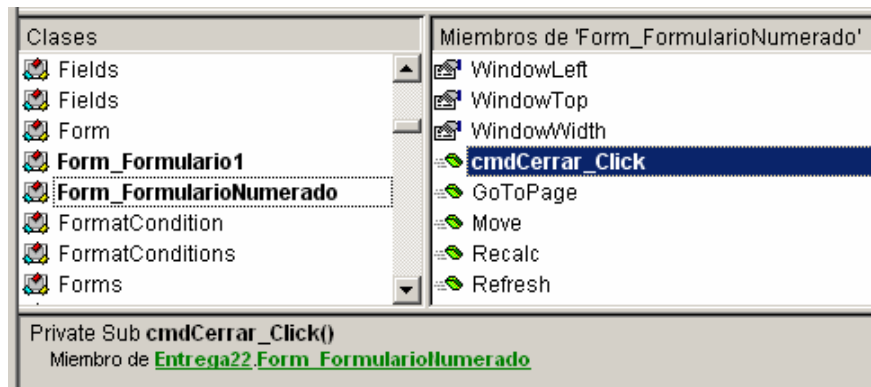
Private Sub cmdCerrar_Click()
On Error GoTo HayError
    DoCmd.Close
Salir:
    Exit Sub
HayError:
    MsgBox Err.Description
    Resume Salir
End Sub

```

Si abrimos el Examinador de objetos, vemos que en la clase del formulario ha aparecido la propiedad **Numero**.



Igualmente podríamos comprobar que existe el procedimiento privado **cmdCerrar\_Click**



## Instanciar un formulario

Existen varias formas de instanciar un formulario, o lo que es lo mismo, asignar un formulario concreto a una variable.

Por cierto, una variable que vaya a hacer referencia a un formulario debe ser del tipo **Variant**, **Object** o **Form**.

Como vimos en capítulos anteriores, el tipo **Variant** es el más genérico de todos, admitiendo casi cualquier cosa. El tipo **object** admite prácticamente cualquier tipo de objeto, pero al contrario que el **Variant**, no puede admitir valores que no sean objetos.

La vinculación en tiempo de ejecución, usando variables del tipo **Variant** u **Object** genera una serie de inconvenientes, como un control más impreciso de los posibles errores, un código menos eficiente y la falta de ayuda “en línea” al escribir el código.

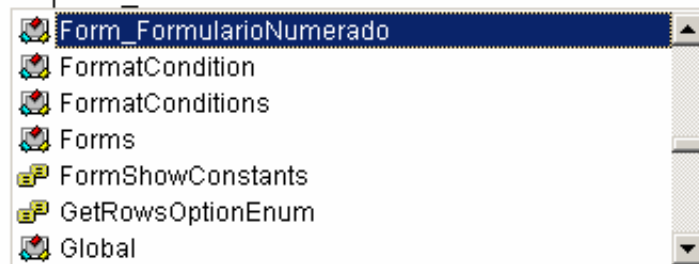
Por ello, cuando a una variable hay que asignarle un objeto concreto, es mejor declararla como del tipo de ese objeto; en nuestro caso del tipo **Form**; y mejor aún como Ya hemos dicho que nuestro formulario es un objeto del tipo **Form\_FormularioNumerado**, podríamos declarar una variable de su tipo, con lo que para activarlo bastaría con asignarlo a la variable con **Set** y **New**.

Para comprobarlo, vamos a crear un módulo estándar y en él crearemos una variable del tipo **Form\_FormularioNumerado**,

Fijémonos que el asistente en línea, nos lo muestra como una de las posibilidades, acompañándolo con el icono que define a las clases.

```
Option Compare Database
Option Explicit
```

```
Dim MiFormulario As Form_FormularioNumerado
```



Vamos ahora a crear un procedimiento que presente una instancia de ese formulario y le asigne algunas propiedades

```
Option Compare Database
Option Explicit
```

```
Public MiFormulario As Form_FormularioNumerado
```

```
Public Sub CargaFormulario()
```

```
    ' Creamos la instancia del formulario
```

```
    Set MiFormulario = New Form_FormularioNumerado
```

```
    With MiFormulario
```

```
        .Numero = 1
```

```
        .Caption = "Formulario N° " & _  
            & Format(.Numero, "000")
```

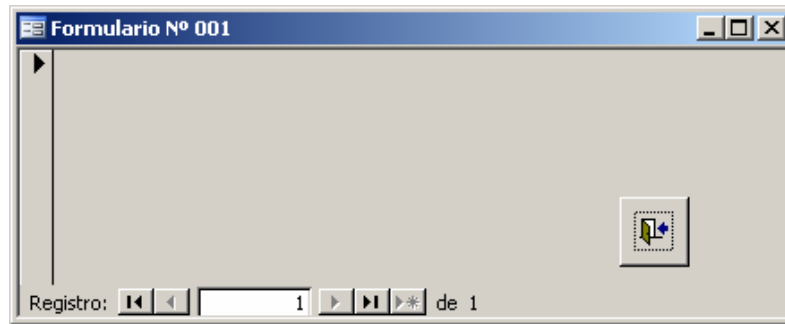
```
    End With
```

```
End Sub
```

Ejecutamos el procedimiento y aparentemente no pasa nada.

En realidad sí ha pasado. Lo que ocurre es que el formulario está cargado pero no está visible.

Si a continuación del ejecutar el procedimiento **CargaFormulario** ejecutamos en la ventana inmediato la línea **MiFormulario.visible=True**, el formulario se nos aparecerá.

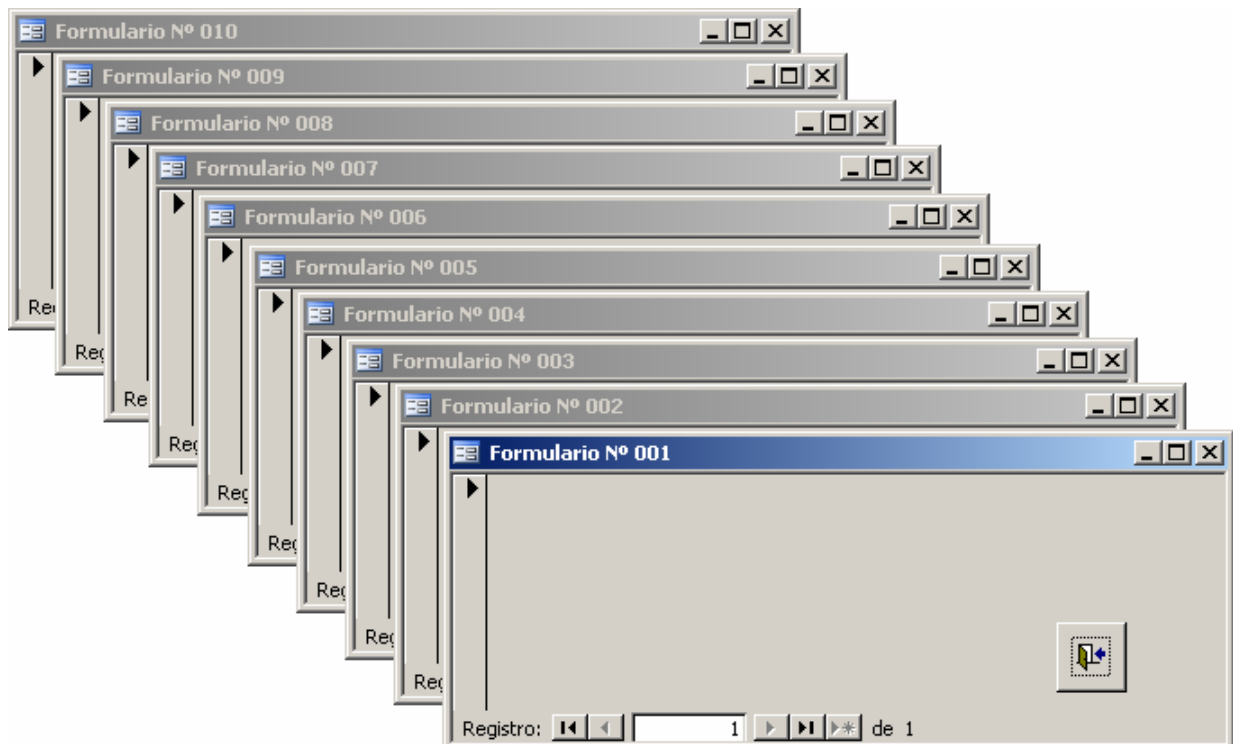


## Múltiples instancias de un formulario

También, como con el resto de las clases, podríamos crear Instancias múltiples de un formulario. Por ejemplo, si tuviéramos el formulario Clientes, podríamos crear 5 instancias, cada una de ellas mostrándonos los datos de un cliente diferente.

Vamos a ver cómo podríamos hacerlo con nuestro formulario. En un módulo ponemos

```
Public aFormularios(1 To 10) As Form
Public Sub FormulariosMultiples()
    Dim i As Integer
    For i = 1 To 10
        Set aFormularios(i) = New Form_FormularioNumerado
        With aFormularios(i)
            .Caption = " Formulario N° " & Format(i, "000")
            .Visible = True
        End With
    Next i
End Sub
```



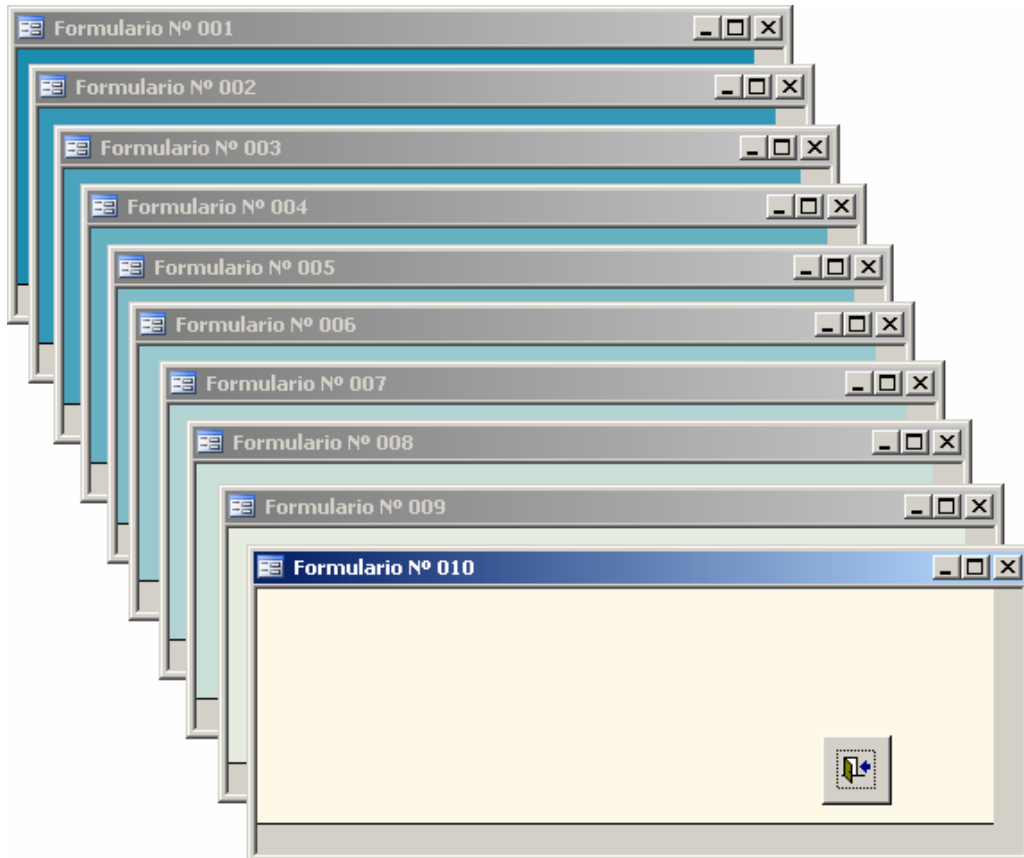


En el caso anterior hemos creado diez formularias, cada uno de ellos con una barra de título diferente.

En realidad los formularios se muestran todos en la misma posición, pero los he desplazado individualmente para que aparezcan tal como aparecen en la imagen.

Con esto hemos visto que tenemos libertad para cambiar las propiedades individuales de cada formulario.

En concreto casi cualquier propiedad del formulario que sea de escritura, por ejemplo:



Las variaciones de color en la sección Detalle se ha realizado de la siguiente forma:

```
Public aFormularios(1 To 10) As Form

Public Sub FormulariosMultiples()
    Dim i As Integer
    Dim lngColor As Long

    For i = 1 To 10
        Set aFormularios(i) = New Form_FormularioNumerado
        With aFormularios(i)
            ' Pongo el título del formulario
            .Caption = " Formulario Nº " & Format(i, "000")
            ' Color de la sección Detalle
            lngColor = RGB(25.5 * i, 128 + 12 * i, 170 + 6 * i)
        End With
    Next i
End Sub
```

```

.Detalle.BackColor = lngColor
' Elimino el selector de registros
.RecordSelectors = False
' Elimino los botones de navegación
.NavigationButtons = False
' Hago visible el formulario
.Visible = True

End With
Next i

```

```
End Sub
```

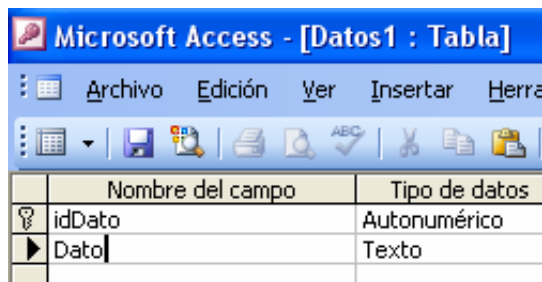
## Conexión con datos en un formulario

Un formulario puede existir sin estar conectado a ningún tipo de datos.

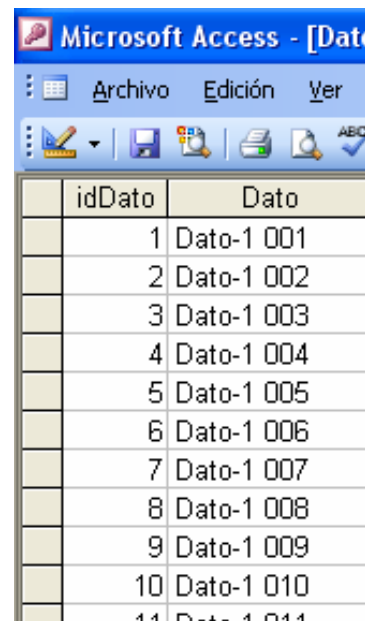
Pero a su vez podemos establecer la conexión de un formulario con un origen de datos por código en tiempo de ejecución.

E incluso, como veremos cuando estudiemos la biblioteca de **ADO**, podríamos enlazarlo a un conjunto de datos (**Recordset**) que exista únicamente en memoria.

Para comprobarlo, vamos a crear la tabla **Datos1** con los siguientes campos



Nombre del campo	Tipo de datos
idDato	Autonumérico
Dato	Texto

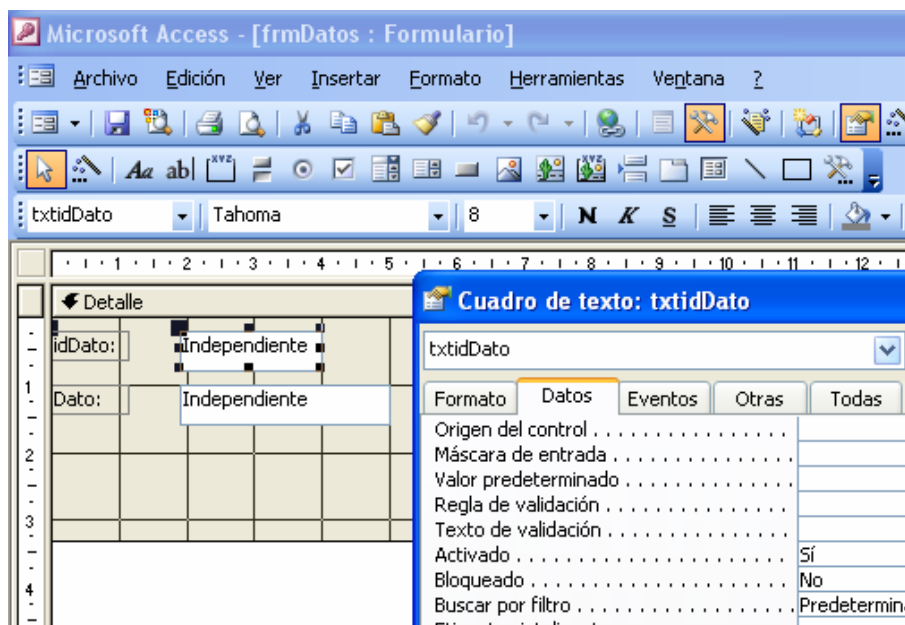


idDato	Dato
1	Dato-1 001
2	Dato-1 002
3	Dato-1 003
4	Dato-1 004
5	Dato-1 005
6	Dato-1 006
7	Dato-1 007
8	Dato-1 008
9	Dato-1 009
10	Dato-1 010
11	Dato-1 011

Para ver el efecto Creamos una serie de registros

A continuación creamos un formulario, de nombre **frmDatos**, con dos cuadros de texto. Sus nombres serán **txtidDato** y **txtdato**.

Al cuadro de texto que contendrá el campo **idDato** (autonumérico) le cambiaremos su propiedad **Activado (Enabled)** y **Bloqueado (Locked)**, para que no se pueda acceder, desde el formulario al campo **idDato**, que es **autonumérico**



Pero esto lo haremos por código en el evento **Al cargar (OnLoad)** del formulario.

```
Me.txtidDato.Enabled = False
```

```
Me.txtidDato.Locked = True
```

Para asignar una tabla, o consulta, a un formulario, utilizaremos la propiedad **Origen del registro (RecordSource)**.

Esto lo podemos hacer también en el evento **Al cargar**.

A la propiedad le asignaremos una cadena que puede contener,

El nombre de una tabla

El nombre de una consulta guardada

Una cadena SQL.

En nuestro caso serían igualmente válidas las siguientes opciones:

```
Me.RecordSource = "Datos"
```

```
Me.RecordSource = "Select * From Datos1;"
```

```
Me.RecordSource = "Select idDato, Dato From Datos1;"
```

```
Me.RecordSource = "Select Datos1.idDato, Datos1.Dato From Datos1;"
```

```
Me.RecordSource = "Select [idDato], [Dato] From Datos1;"
```

Respecto a la palabra **Me**, que estamos utilizando podemos recordar que la vimos cuando analizábamos las clases.

Igual que entonces, **Me** representa el objeto creado mediante la clase, es decir, representa al propio formulario.

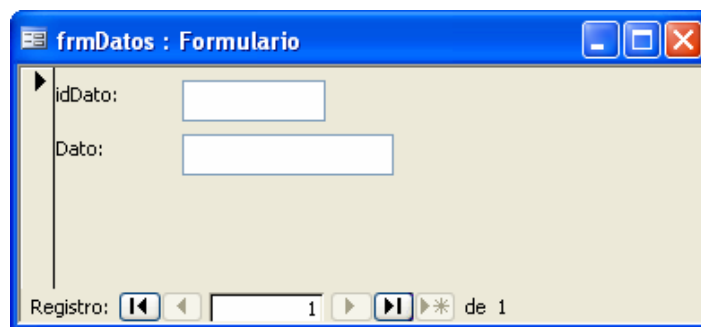
Por eso, si escribimos **Me** y a continuación el punto, nos aparece la ayuda en línea que nos suministra el editor de Visual Basic.

No es estrictamente necesario utilizarla.

Las siguientes instrucciones serían tan válidas, como los utilizadas en las líneas anteriores.

```
txtidDato.Enabled = False
txtidDato.Locked = True
RecordSource = "Select idDato, Dato From Datos1;"
-----
-----
```

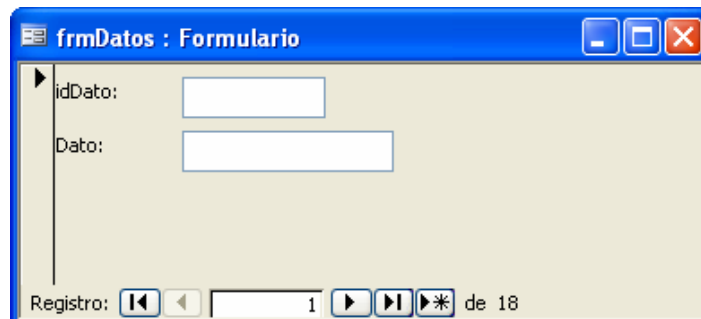
Si abrimos el formulario, sin asignarle un origen de datos, tendrá un aspecto semejante a éste:



Veamos cómo cambia al asignarle un origen de datos.

En el evento, **Al cargar** del formulario, escribimos lo siguiente:

```
Private Sub Form_Load()
    Dim strSQLDatos As String
    txtidDato.Enabled = False
    txtidDato.Locked = True
    strSQLDatos = "Select idDato, Dato From Datos1;"
    RecordSource = strSQLDatos
End Sub
```



Vemos que ahora nos indica que tenemos un determinado número de registros, por lo que podemos suponer que efectivamente está conectado a la tabla Datos.

Pero todavía no vemos nada en los cuadros de texto.

La conexión entre los cuadros de texto y los correspondientes campos de la tabla, debe efectuarse después de que hayamos conectado la tabla al formulario.

En un control, el campo al que se conecta, lo determina la propiedad **Origen del control (ControlSource)**.

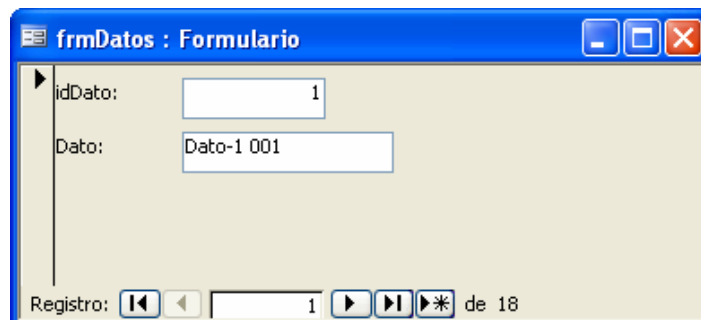
Su sintaxis es así

```
NombreDelControl.ControlSource = NombreDelCampo
```

Para ello modificaremos el código del evento **Al cargar** de la siguiente manera:

```
Private Sub Form_Load()  
    Dim strSQLDatos As String  
    txtidDato.Enabled = False  
    txtidDato.Locked = True  
    strSQLDatos = "Select idDato, Dato From Datos;"  
    RecordSource = strSQLDatos  
    txtidDato.ControlSource = "idDato"  
    txtDato.ControlSource = "Dato"  
End Sub
```

Con lo que el formulario se verá de una forma semejante a esta:



Podemos comprobar que ahora sí tenemos acceso a los datos.

### **Cambio, en tiempo de ejecución del origen de datos.**

Vamos a crear ahora una segunda tabla a la que vamos a llamar **Datos2**.

Para simplificar haremos que esta nueva tabla contenga los mismos campos que la tabla **Datos**.

Cambiaremos el contenido de la columna datos para que podamos apreciar la diferencia.

Yo le he puesto valores del tipo

Dato-2 001

Dato-2 002

Dato-2 003

---

---

Para comprobar el proceso, copiamos el formulario anterior y le ponemos como nombre **frmDatosCambiados**.

Al copiar el formulario, se copiará con sus propiedades y código asociado.

A continuación, a este nuevo formulario le añadiremos dos botones que serán los que efectúen el “cambio” al hacer **Clic** sobre ellos.

A estos botón le pondremos por nombre **cmdTabla1** y **cmdTabla2**.

Si tenemos activado el botón del **[Asistente para Controles]** (la varita mágica que hemos visto en un punto anterior, lo desactivamos para tener un control completo del proceso.

En cada uno de los eventos **Al hacer clic** , escribimos respectivamente lo siguiente.

```
Private Sub cmdTabla1_Click()  
    RecordSource = "Select idDato, Dato From Datos1;"  
End Sub
```

```
Private Sub cmdTabla2_Click()  
    RecordSource = "Select idDato, Dato From Datos2;"  
End Sub
```

Sólo con esto podemos comprobar que cambiamos el origen de datos para el formulario simplemente modificando la cadena de SQL que define el conjunto de datos con el que queremos trabajar, utilizando simplemente la propiedad **RecordSource**.

Más adelante veremos la gran utilidad que nos brinda esta propiedad.

## **Cambio, en tiempo de ejecución del origen de datos, con nombre de campos diferentes.**

En los puntos anteriores, hemos conectado un formulario a dos tablas diferentes, con sólo pulsar un botón.

El problema era muy sencillo de resolver, ya que los campos tienen el mismo nombre en las dos tabla.

Pero ¿cómo se resuelve el caso en el que los nombres de los campos de las tablas no sean iguales?

Volveremos a usar la propiedad **ControlSource** del control Cuadro de texto (**TextBox**).

Vamos a comprobar todo lo dicho

Creamos la tabla Provincias, con los campos

**idProvincia** y **Provincia**.

Copiamos otra vez el formulario y le ponemos como nombre **frmConTresOrigenes**.

Además le añadimos un nuevo botón de nombre **cmdProvincias**.

Vamos a hacer que cuando se pulse el botón **cmdProvincias** se muestren las provincias, y cuando se pulsen los otros botones, se muestren sus respectivas tablas.

Además vamos a hacer que la barra del título del formulario muestre el nombre de la tabla conectada usando la propiedad **Caption** del formulario.

Vamos a cambiar también el nombre de las etiquetas asociadas a los cuadros de texto, poniéndoles como nombres respectivos, **lblidDato** y **lblDato**.

El prefijo **lbl** nos ayuda a saber que ese nombre se corresponde al de una etiqueta (Label).

Para conseguir nuestro objetivo, añadimos código para gestionar el evento Clic del nuevo botón, y modificamos el código anterior.

El código completo quedará así:

```
Option Compare Database
Option Explicit

Private Sub Form_Load()
    Dim strSQLDatos As String
    txtidDato.Enabled = False
    txtidDato.Locked = True
    cmdTabla1_Click
End Sub

Private Sub cmdTabla1_Click()
    AjustaCamposTablas
    Caption = "Tabla Datos1"
    RecordSource = "Select idDato, Dato From Datos1;"
End Sub

Private Sub cmdTabla2_Click()
    AjustaCamposTablas
    Caption = "Tabla Datos2"
    RecordSource = "Select idDato, Dato From Datos2;"
End Sub

Private Sub cmdProvincias_Click()
    RecordSource = "Provincias"
    Caption = "Tabla Provincias"
    txtidDato.ControlSource = "idProvincia"
    txtDato.ControlSource = "Provincia"
    txtidDato.Format = "00"
    If lblidDato.Caption <> "Clave:" Then
        lblidDato.Caption = "Clave:"
        lblidDato.Caption = "Provincia:"
    End If
End Sub

Private Sub AjustaCamposTablas()
    txtidDato.ControlSource = "idDato"
    txtDato.ControlSource = "Dato"
    txtidDato.Format = "#,##0"
    If lblidDato.Caption <> "idDato:" Then
        lblidDato.Caption = "idDato:"
    End If
End Sub
```

```
        lblidDato.Caption = "Dato:"  
    End If  
End Sub
```

El código es lo suficientemente sencillo como para que el lector, si ha seguido estas entregas, lo pueda entender sin dificultad.

Sólo una matización:

Podemos ver que desde el evento Load, se llama directamente al gestor del evento Clic del botón **cmdTabla1** mediante la instrucción:

**cmdTabla1\_Click**

Esto es así porque un gestor de evento no deja de ser un procedimiento **Sub** más, y por tanto se puede llamar de forma directa.

El formulario mostrará la información que seleccionemos mediante los botones:



**Ejercicio.**

Aprovechando lo visto en el punto **Formularios múltiples**, le sugiero al lector que cree varias instancias del mismo formulario, cada una de ellas con un origen distinto de datos.

**Próxima entrega.**

En la próxima entrega jugaremos con algunas de las propiedades de los formularios, y digo jugaremos, ya que diseñaremos un formulario panel para jugar al **Bingo** o **Lotería**, en casa, que incluso nombre el número de las bolas conforme vayan saliendo.

Cuando vemos los informes, completaremos el programa de Bingo con la posibilidad de imprimir los cartones.