

Aprenda

los fundamentos
de programación y el
diseño de aplicaciones
en

Power Builder 7.0

Edición 1.0

Prohibida la reproducción total o parcial
Sin nuestro consentimiento
Derechos de copia reservados
www.LibrosDigitales.NET

Introducción

Objetivos

En este primer módulo se desarrollaran los siguientes puntos:

- ✓ Introducción al concepto "Orientado a Objeto"
- ✓ Descripción del entorno de desarrollo
- ✓ Descripción de los principales objetos
- ✓ Descripción de los principales controles
- ✓ Pasos para Iniciar una aplicación
- ✓ Desarrollo de una aplicación

Introducción al Concepto “Orientado a Objeto”

Paralela a la búsqueda de la computación distribuida, dentro del área de la informática se ha venido madurando la tecnología “**Orientado a Objeto**”, la cual soluciona muchos problemas de los sistemas estructurados, permitiendo un desarrollo de software más rápido y dinámico. Además de su potencial reusabilidad al ser cada objeto una entidad autónoma.

Existen muchas definiciones en la literatura pero todas coinciden en que se trata de un modelo de programación para representar la realidad. Además se puede decir que para que un sistema sea “**Orientado a Objeto**” debe soportar determinadas características, como son:

- Encapsulamiento o capacidad de ocultar la información.
- Herencia.
- Polimorfismo.

Veamos un ejemplo, el mundo real está compuesto por pequeñas entidades que interactúan con otras entidades y/o sistemas. Para el desarrollo de aplicaciones software estas entidades son encapsuladas dentro de unidades estructurales llamadas objetos, lo que permite tratar a cada uno de una manera independiente de los demás, haciendo que cada objeto sea una unidad operativa con otros objetos y lograr un fin común. Un buen ejemplo de la programación orientada a objetos es el ensamblaje de una computadora, la cual está constituida por diferentes partes independientes, que a su vez, están contruidos por pequeñas partes ensambladas con anterioridad, dando la posibilidad de escoger su configuración, lo cual sería imposible si nos tocara elaborar cada uno de los componentes, que es prácticamente lo que se hace con el software actual, donde cada nueva aplicación arranca casi desde cero.

Cada objeto es construido a partir de una plantilla llamada **Clase**, en donde se consignan todas sus características y operaciones, los cuales se conocen como **atributos y métodos** respectivamente. Cada una de estas clases representa un tipo de objeto. Siguiendo con el ejemplo anterior, se puede citar una clase **Computadora**, la cual entre sus características tiene capacidad de memoria RAM, capacidad de disco duro y velocidad, se pueden armar gran cantidad de configuraciones diferentes, a partir de un mismo tipo de características con tan solo cambiar sus valores, en este caso se dice que cada objeto (Computadora con atributos ó configuración particular) es una instancia de su clase (Computadora).

Herencia

Este sistema permite derivar una clase de otra, permitiendo agregarle funcionalidad y características que la clase base no posee, dando origen a una clase derivada, la cual posee todos los encantos de su clase padre más los suyos propios. Por ejemplo, si se define una clase “**Procesador Pentium**”, podemos derivar una clase “**Procesador Pentium MMX**”, la cual ofrece todos los servicios de su clase base, clase “**Procesador Pentium**”, más otros, tales como sus procesos para multimedia.

Polimorfismo

Es la capacidad que tienen los objetos de soportar métodos con un mismo nombre y actuar de manera diferente según sea su tipo. En el ejemplo de la **Computadora**, se puede observar como podemos leer de cada uno de sus dispositivos de almacenamiento tales como disco duro, disco flexible, CDROM memoria RAM y ROM, etc. Lo cual se puede hacer por medio de una misma orden para todos, tal como **leer**, actuando esta orden de forma diferente en cada dispositivo, de acuerdo a su tipo, ya que cada uno utiliza un método diferente de almacenamiento y por tanto de lectura.

Agregación

La agregación es la propiedad por la cual un objeto puede estar compuesto por otros objetos, los cuales funcionan de una manera independiente dentro del objeto contenedor, pero de una manera cooperativa, siguiendo con el ejemplo de la Computadora, se puede observar como esta contiene otros objetos como es el caso de motherboard, tarjeta de vídeo, disco duro, etc. Los cuales se pueden considerar objetos independientes, por ser unidades operativas por si solas, pero que al unirse pueden conformar una unidad mas grande.

Encapsulación

Esta es una característica que presentan los objetos, por medio de la cual sólo dan a conocer a los otros objetos, los atributos y métodos a los que los pueden acceder, siendo esta la forma que el objeto se presenta para que los demás se comuniquen con él, esta se podría mencionar como la interfaz del objeto, de acuerdo al ejemplo que se trae, una interfaz de un objeto sería análogo a los dispositivos o interfaces que trae la computadora para comunicarse con el mundo exterior, tal como el teclado, Mouse, pantalla, etc. Los cuales nos permite comunicarnos con el objeto Computadora, sin importar la manera misma como lo hace.

Entre los resultados de todas estas poderosas características de los objetos tenemos una potencial reusabilidad, tanto hacia adentro como hacia fuera del proyecto, teniendo en cuenta que para esto no solo bastan las herramientas, también hace falta "**Cultura hacia el Rehúso**", como resultado de una buena metodología de análisis y diseño, punto que se puede anotar como un acierto mas de la tecnología de objetos, ya que sus metodología hacen mucho más fácil el desarrollo de la aplicación, ya que se trata de emular el mundo real representándolo a partir de los objetos, donde el paso de los procesos de análisis y diseño a la codificación se hace de una manera casi automática y mucho menos traumática que en el análisis estructurado.

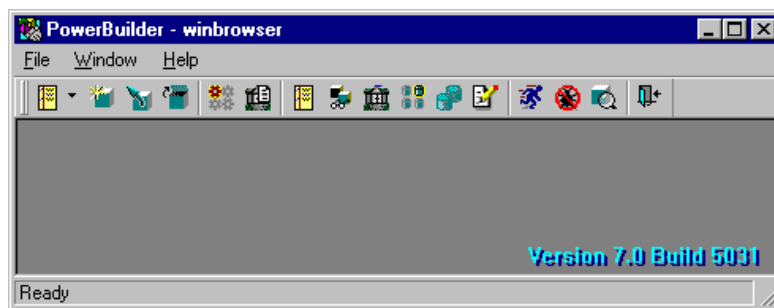
Descripción del entorno de desarrollo

Los Painters

Son las herramientas utilizadas para construir los objetos que van a formar parte de una aplicación. **PowerBuilder** proporciona un Painter para cada tipo de objeto a construir. Un **Painter** está formado por una hoja de trabajo (ventana) y un menú (PainterBar).

La Ventana de PowerBuilder

Inicialmente, PowerBuilder presenta la barra de menú (Menú Bar) y el PowerBar.

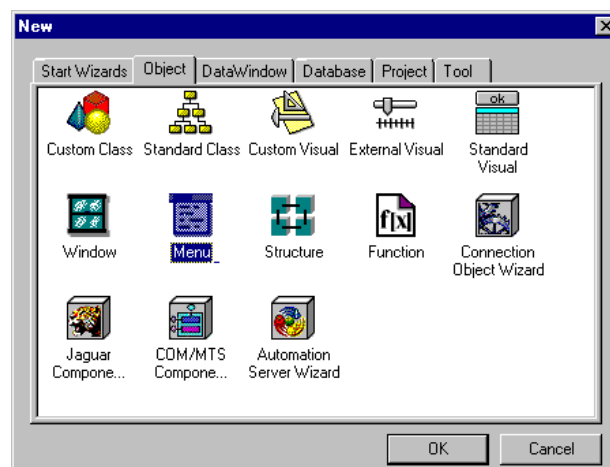


El PowerBar

Es el punto de control principal para la construcción de una aplicación. Es desde donde abrimos los Painter, depuramos y ejecutamos la aplicación, personalizamos el entorno según nuestras necesidades.

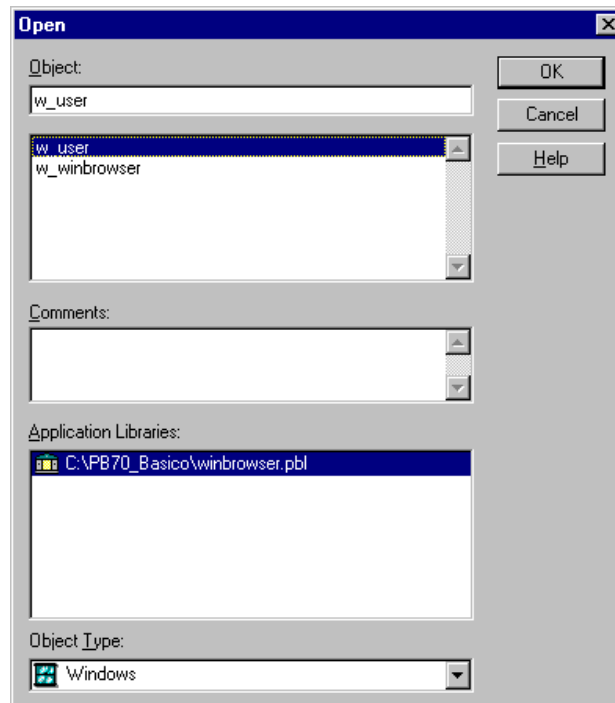
Botón New

Este botón lo utilizamos cuando vamos a crear un nuevo objeto.



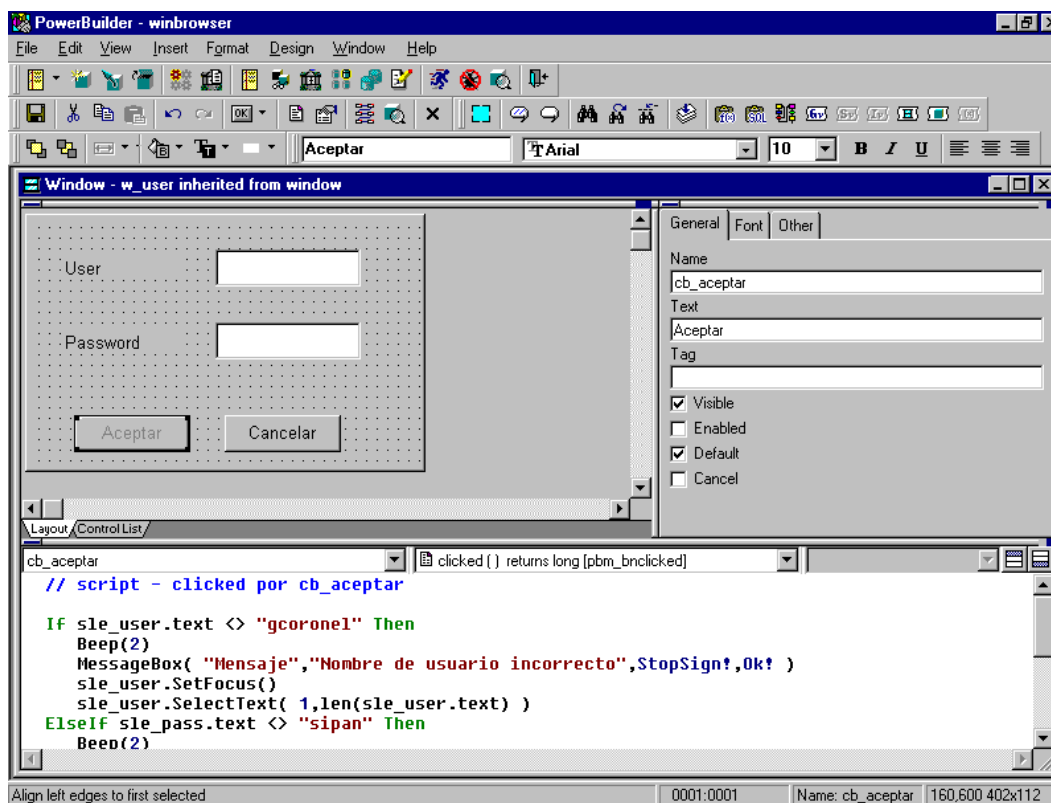
Botón Open

Este botón lo vamos a utilizar cuando queremos editar un objeto, por ejemplo si deseamos realizar modificaciones en una ventana.



El PainterBar

Cuando se abre un Painter, PowerBuilder muestra una ventana que proporciona el área de trabajo en donde diseñaremos el objeto a construir. Adicionalmente se muestra el PainterBar, constituido por botones que proveen acceso a las herramientas disponibles en el Painter y el menú del Painter que reemplaza al Menú Bar en la barra de menú.



Descripción de los principales objetos

El Objeto Application (Aplicación)

Es el punto de inicio de una aplicación. Es un objeto discreto que se guarda en una librería PowerBuilder, tal como los demás objetos.

Este objeto define el comportamiento de la aplicación, tales como las librerías a usar, fuentes por defecto para el texto, y que procesos deben ocurrir cuando la aplicación se inicia y finaliza.

Cuando corre una aplicación, el evento **Open** se activa en el objeto Application, ejecutando el script que contiene para iniciar las actividades de la aplicación. Cuando se finaliza la aplicación, el evento **Close** se activa, ejecutándose el script que contiene, que típicamente cierra las bases de datos o escribe las preferencias a un archivo de datos. Si se produce un error serio durante la ejecución, se activa el evento **SystemError**.

El objeto Window (Ventana)

Es la interfaz entre el usuario y una aplicación PowerBuilder. Se utiliza para mostrar información, solicitar información al usuario y responde a acciones del Mouse y el teclado.

La definición de una ventana incluye propiedades, eventos y funciones. Las propiedades determinan el estilo de la ventana (como la apariencia). Los eventos son acciones de la ventana; cuando un evento es disparado, el script asociado es ejecutado. Las funciones pueden disparar eventos de la ventana, manipular o cambiar la ventana, o proveer información acerca de la ventana.

El objeto DataWindow (Ventana de Datos)

Un objeto DataWindow se utiliza para recuperar, presentar y manipular datos desde una base de datos relacional u otra fuente de datos (por ejemplo una hoja de Excel ó un archivo de dBase).

Se puede seleccionar diferentes estilos de presentación. Por ejemplo, se puede mostrar los datos en formato Tabular o FreeForm.

El objeto Menu (Menú)

Los menús son listas de comandos u opciones que el usuario puede seleccionar en la ventana activa.

Un menú desplegable es un menú bajo un elemento de la barra de menú. Un menú en cascada es un menú que aparece al lado de un elemento de un menú desplegable. Cada elemento de un menú es definido como un objeto menú en PowerBuilder.

Functions (Funciones)

PowerBuilder permite definir dos tipos de funciones:

- **Funciones a nivel de objeto**, son definidas para un tipo particular de ventana, menú u otro tipo de objeto; son encapsuladas dentro del objeto en el cual se definen.
- **Funciones Globales**, no son encapsuladas dentro de un objeto, pero son guardadas como objetos independientes.

A diferencia de las funciones a nivel de objeto, las funciones globales no actúan sobre una instancia particular de un objeto. Por el contrario, estas funciones ejecutan procesos de propósito general como cálculos matemáticos o manipulación de cadenas.

Queries (Consultas)

Una consulta es una sentencia SQL que se guarda con un nombre de tal forma que puede ser usada repetidamente como la fuente de datos para un objeto DataWindow. Las consultas incrementan la productividad por que son codificadas una vez y pueden ser rehusadas tan frecuentemente como sean necesarias.

Strutures (Estructuras)

Una estructura es una colección de una o más variables relacionadas de uno o más tipos de datos agrupados bajo un mismo nombre. En algunos lenguajes, como Pascal y Cobol, las estructuras son llamadas registros.

Las estructuras permiten agrupar a entidades relacionadas como una unidad en vez de individualmente.

Existen dos clases de estructuras:

- **Estructuras a nivel de objeto**, están asociadas con un tipo particular de objeto como una ventana o menú. Estas estructuras son usadas en scripts para el mismo objeto. También se puede definir la estructura para que pueda ser accesible desde otros scripts.
- **Estructuras Globales**, no están asociadas con ningún objeto en su aplicación. Podemos hacer referencia a estas estructuras desde cualquier parte de su aplicación.

User objects (Objeto de Usuario)

Las aplicaciones, frecuentemente tienen características comunes. Por ejemplo, usualmente se necesita un botón **Cerrar** que ejecute un conjunto de operaciones y que cierre la ventana. O un cuadro de lista que muestre todos los departamentos. Podría requerirse también que todos los controles DataWindow ejecuten el mismo tipo de chequeo de error. También podría necesitarse un archivo visor predefinido que pueda conectarse de manera simple dentro de su ventana cuando se le requiera.

Si usted se encuentra usando repetidamente aplicaciones con las mismas características, entonces deberá definir un **objeto de usuario**. El objeto de usuario se crea una vez y se utiliza tantas veces se necesite.

Tenemos dos tipos de objetos del usuario:

- **Objetos Visuales**, es un control o un conjunto de controles, que se pueden reutilizar y tienen un comportamiento definido.
- **Clase de Objetos**, son módulos de procesamiento reutilizables que no tienen componente visual.

Libraries (Librerías)

Los objetos, tales como ventanas y menús, se guardan en librerías PowerBuilder (archivos PBL). Cuando se ejecuta una aplicación, PowerBuilder recupera los objetos desde la librería. Las aplicaciones pueden usar tantas librerías como se necesite. Cuando se crea una aplicación, se debe especificar que librerías se va a utilizar.

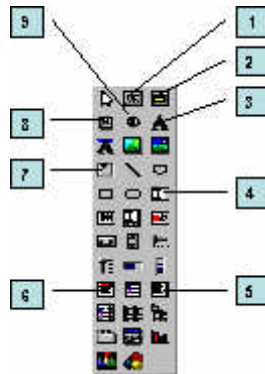
Projects (Proyectos)

Para permitir que un usuario ejecute su aplicación de la misma forma como ejecuta otras aplicaciones, se crea un objeto **Project**. El objeto **Project** puede empaquetar su aplicación en cualquiera de las siguientes formas:

- Como un archivo ejecutable individual que contiene todos los objetos de la aplicación.
- Como un archivo ejecutable y una o más librerías dinámicas del PowerBuilder que contienen objetos que serán enlazados en tiempo de ejecución.

Cuando se empaqueta la aplicación, se podrá también proveer algunos recursos adicionales, tales como mapas de bits o iconos. Estos recursos se incluirán en los ejecutables o librerías dinámicas, o se distribuirán en forma separada.

Descripción de los principales controles

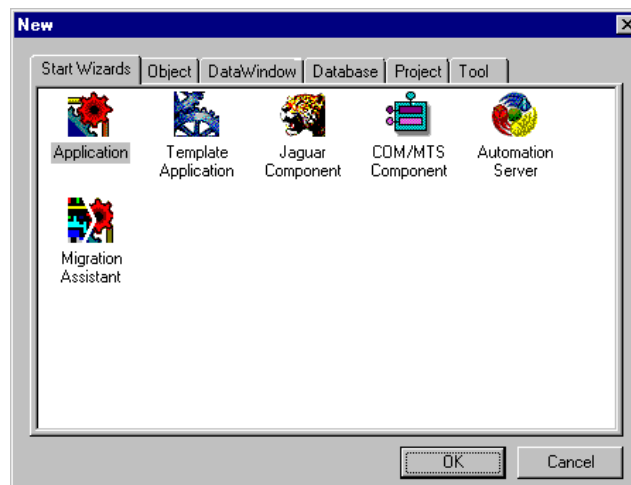


1. CommandButton
2. PictureCommandButton
3. StaticText
4. SingleLineEdit
5. ListBox
6. DropDownListBox
7. GroupBox
8. CheckBox
9. RadioButton

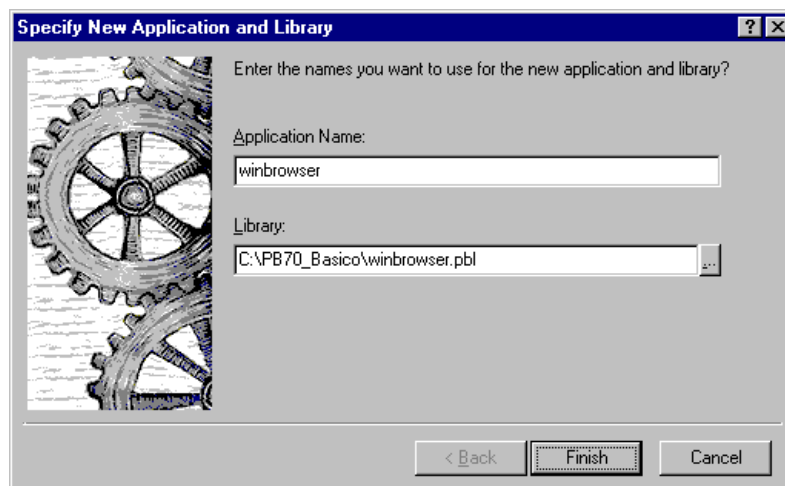
Pasos para Iniciar una Aplicación

Cuando usted carga PowerBuilder, este lee la última aplicación con la que estuvo trabajando. Para iniciar una nueva aplicación, debe seguir los siguientes pasos:

1. Cerrar todos los painters.
2. En el PowerBar haga clic en el botón **New** (también puede ejecutar el comando **New** del menú **File**).
3. En el diálogo **New**, seleccione el icono **Application**, en la ficha **Start Wizards**. Finalmente haga clic en el botón **OK**.



4. Finalmente en el diálogo **Specify New Application and Library**, ingrese el nombre del objeto aplicación, y el nombre y ubicación de la librería. Para terminar haga clic en el botón **Finish**.



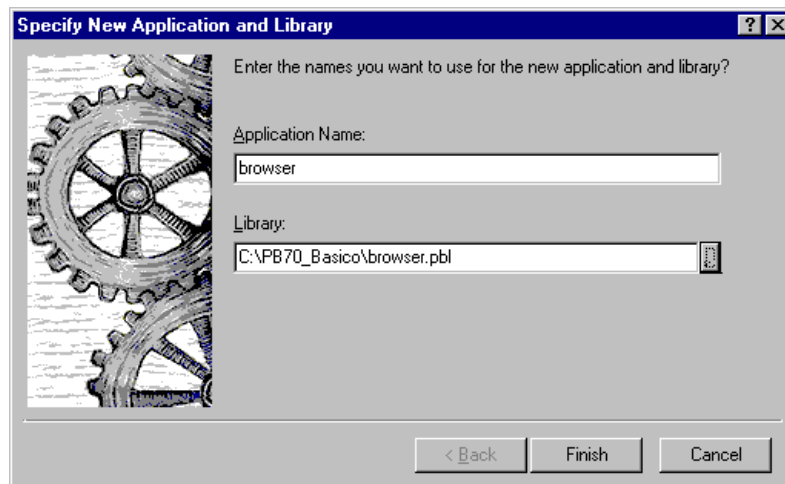
Desarrollo de una aplicación

Carpeta de Trabajo

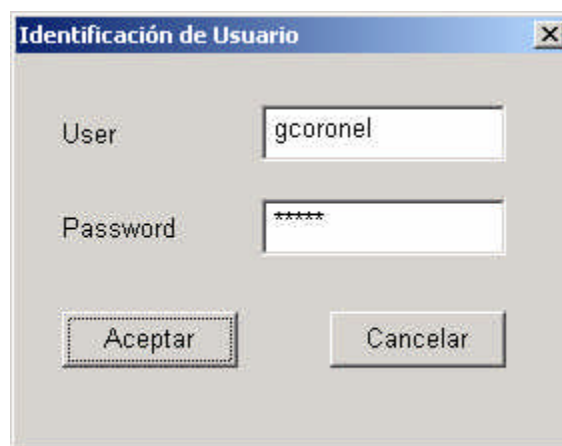
La carpeta de trabajo para el desarrollo del curso es: C:\PB70_Basico.

Iniciar la aplicación

El nombre de la aplicación es **browser** y la librería **browser.pbl**.



Crear una Ventana de Identificación



Esta ventana permite identificar a la persona que desea ingresar a un sistema, el botón aceptar se debe habilitar solo cuando el usuario halla ingresado sus datos completos.

Creación de una Nueva Ventana

Proceda a crear una nueva ventana, y asígnele los siguientes valores a sus principales propiedades:

- Name: w_user
- Title: Identificación de Usuario
- Type: Response

Añadir Controles a la Ventana

Proceda a añadir los siguientes controles:

- 2 StaticText
- 2 SingleLineEdit
- 2 CommandButton

Establecer las Propiedades de los Controles

- Establezca el tamaño adecuado
- Asígnele un nombre apropiado, por ejemplo sle_user, sle_pass, cb_aceptar, cb_cancelar.
- A otras propiedades asígnele valores según su criterio.

Programar el ingreso de datos

script - modified for sle_user

```
if text = "" or sle_pass.text = "" then
    cb_aceptar.enabled = false
else
    cb_aceptar.enabled = true
end if
```

script - modified for sle_pass

```
if text = "" or sle_user.text = "" then
    cb_aceptar.enabled = false
else
    cb_aceptar.enabled = true
end if
```

Programación del Botón Aceptar

script - clicked por cb_aceptar

```
If sle_user.text <> "gcoronel" Then
    Beep(2)
    MessageBox( "Mensaje","Nombre de usuario incorrecto",StopSign!,Ok! )
    sle_user.SetFocus()
    sle_user.SelectText( 1,len(sle_user.text) )
ElseIf sle_pass.text <> "sipan" Then
    Beep(2)
    MessageBox( "Mensaje","Contraseña del usuario es
incorrecta",StopSign!,Ok! )
    sle_pass.SetFocus()
    sle_pass.SelectText( 1,len(sle_pass.text) )
Else
    MessageBox( "Mensaje","Los datos son correctos",Information!,Ok! )
End If
```

Programación del Botón Cancelar

Script - Clicked for cb_cancelar

```
Close( Parent )
```

Ejecutar la Ventana w_user

Para ejecutar la ventana siga los siguientes pasos:

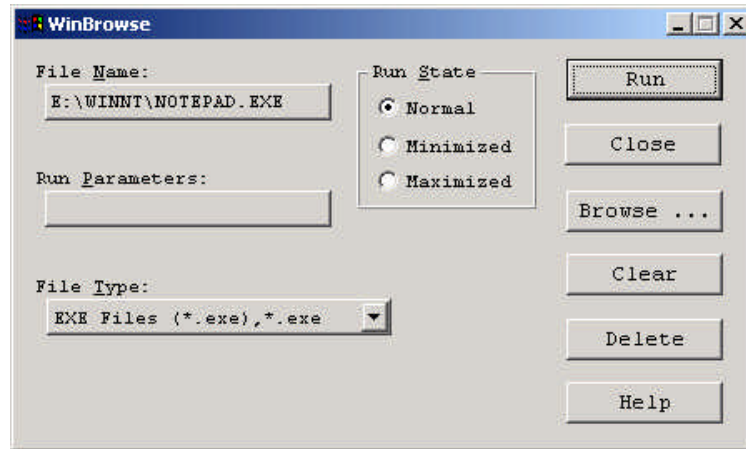
1. Del menú File ejecute el comando **Run/Preview...**
2. En el diálogo **Run/Preiew** seleccione la ventana **w_user**
3. Haga clic en el botón **Ok**

Ahora puede interactuar con la ventana como lo haría el usuario final.



En www.LibrosDigitales.NET encontrará la solución de este ejercicio.

Programar la Ventana WinBrowser



Esta ventana permite buscar un archivo, y si es un programa permite ejecutarlo.

Creación de una Nueva Ventana

Proceda a crear una nueva ventana, y asígnele los siguientes valores a sus principales propiedades:

- Name: w_winbrowser
- Title: WinBrowser
- Type: Main

Colocar y definir las propiedades de los controles en la ventana

En la ventana que acaba de crear, debe colocarse 6 tipos diferentes de controles. En la lista a continuación, indique el tipo de control y el número de ocurrencias del mismo en la ventana.

| Nro. | Tipo de Control | Nro. De ocurrencias |
|------|-----------------|---------------------|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |



En www.LibrosDigitales.NET encontrará la solución de este ejercicio.

Defina los controles de tipo StaticText

Para cada uno de ellos:

- Especifique el texto señalado en el diseño.
- Asígnele un nombre adecuado.
- Seleccione una alineación apropiada.

Defina los controles SingleLineEdit

Para cada uno de ellos:

- Establezca el tamaño apropiado.
- Asígnele un nombre adecuado.
- Habilite la propiedad que permita realizar un scroll horizontal.
- Establezca el estilo del borde según el diseño.

Defina el control DropDownListBox

- Establezca el tamaño apropiado.
- Asígnele un nombre adecuado.
- Defina la lista asociada al control. Use la siguiente lista:
 - All files (*.*), *.*
 - Exe files (*.exe), *.exe
 - Com files (*.com), *.com
 - Batch files (*.bat), *.bat
- Deshabilite la propiedad que ordena la lista.

Defina el control GroupBox para los RadioButtons

- Asígnele el texto señalado en el diseño.
- Asígnele un nombre apropiado.

Defina los controles de tipo RadioButton

- Asígnele un nombre adecuado.
- El botón Normal debe ser la opción por defecto.

Defina los controles de tipo **CommandButton**

- Asígneles un nombre apropiado.
- El botón Ejecutar debe ser el botón por defecto.

Establecer el orden de tabulación

- Del menú **Format**, ejecute el comando **Tab Order**; ó haga clic en el botón **TabOrder** del **PainterBar**.
- Establezca para cada control, el correspondiente número de orden.
- Cuando termine, deshabilite el botón **TabOrder** del **PainterBar**.

Codificación del script que establece el estado por defecto de los controles

- En el evento **open** de la ventana se debe establecer el estado por defecto de los controles.
- El estado por defecto de los controles es:
 - En el control **File Type** debe seleccionar el primer elemento de la lista.
 - En el control **Run State** debe seleccionar como estado inicial **Normal**.
 - El foco debe ubicarse en el control **File Name**.

Codificar el script para el botón **Browse...**

Las especificaciones para el botón **Browse...** son las siguientes:

- El script debe determinar qué tipo de archivo ha seleccionado el usuario en el control **File Type**.
- Para abrir el diálogo **File Select** debe invocar a la función **GetFileOpenName**.
- Al abandonar el diálogo **File Select** debe visualizarse en el control **File Name**, el nombre del archivo seleccionado por el usuario.
- Si el usuario seleccionó un archivo en el diálogo **File Select**, el script debe entregar el foco al control **Run Parameters**.
- Si el usuario abandona el diálogo **File Select**, ejecutando el botón **Cancelar**, el script debe entregarle el foco al control **File Type**.
- Utilice las variables locales que crea necesarias.

Codificación del script para el botón Run

Las especificaciones para el botón **Run** son:

- El script debe ejecutar el archivo seleccionado por el usuario.
- La ejecución debe efectuarse teniendo en cuenta los parámetros indicados por el usuario en el control **Run Parameters**, y el estado inicial de la ventana de la aplicación según lo establecido en el control **Run State**.

Codificación del script para el botón Clear

Las especificaciones para el botón **Clear** son las siguientes:

- Debe limpiar los controles **File Name** y **Run Parameters**.
- Debe restaurar la selección por defecto para los controles **File Type** y **Run State**.
- Debe entregar el foco al control File Name.

Codificación del script para el botón Delete

Las especificaciones para el botón **Delete** son las siguientes:

- Debe sonar la alarma si el usuario no ha especificado el nombre del archivo a eliminar.
- Si ha especificado un archivo, debe mostrar un mensaje solicitando confirmación de su eliminación.
- Si el usuario confirma la eliminación del archivo, además de eliminarlo, debe limpiar el control **File Name**.

Codificación del script para el botón Help

El botón Help debe invocar al archivo de ayuda de PowerBuilder.

Ejecutar la Ventana w_WinBrowser

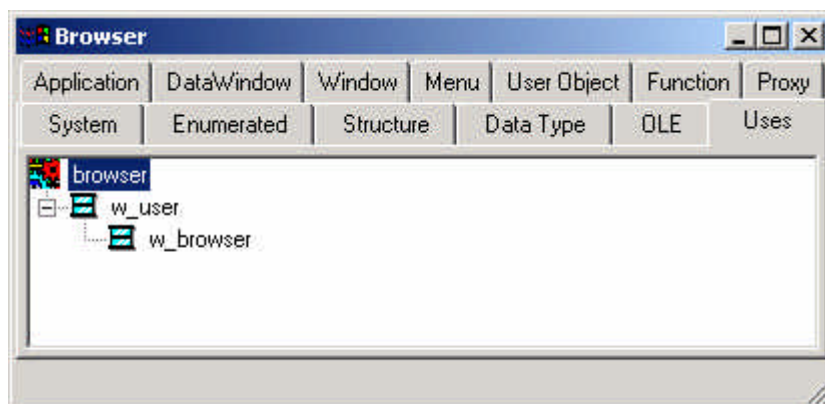
Para ejecutar la ventana siga los siguientes pasos:

- Del menú File ejecute el comando **Run/Preview...**
- En el diálogo **Run/Preiew** seleccione la ventana **w_winbrowser**
- Haga clic en el botón **Ok**

Ahora puede interactuar con la ventana como lo haría el usuario final.

Enlazar los objetos

Ahora debemos enlazar los objetos, como se muestra en el siguiente gráfico.



- Desde el evento **Open** del objeto aplicación **Browser** debe abrir la ventana **w_user**.
- Desde el evento **Clicked** del botón **Aceptar**, que se encuentra en la ventana **w_user**, se debe abrir la ventana **w_WinBrowser**, siempre y cuando los datos del usuario sean los correctos.

Ejecutar la Aplicación

Ahora ya puede ejecutar la aplicación, con el comando **Run** del menú **File**.

Diseño de una Interfaz

Objetivos

En este primer módulo se desarrollaran los siguientes puntos:

- ✓ Tipos de Ventanas
- ✓ Descripción de una Interfaz MDI
- ✓ Creación de Menús
- ✓ Manejo de Ventanas
- ✓ Uso de Pronombres
- ✓ Estructuras
- ✓ Paso de Parámetros a una Ventana
- ✓ Recibir un Parámetro de una Ventana
- ✓ Manejo de Variables

Tipos de Ventanas

La propiedad **WindowType** define el tipo de una ventana, y estas se describen a continuación.

Ventana Marco (Mdi!)

Es una ventana que abarca toda una aplicación. Todas las otras ventanas que se abren dentro de la aplicación lo hacen dentro del marco de esta ventana. Tienen la propiedad de que al cerrarse provoca el cierre de todas las ventanas abiertas en su ámbito. Pueden tener un menú asociado y es este es el menú general de la aplicación.

Ventana Marco con Micro Ayuda (MdiHelp!)

Variante de la ventana anterior que tiene una línea en la parte inferior donde podemos enviar mensajes informativos al usuario, sin que esto afecte el rendimiento de la aplicación.

Ventana Principal (Main!)

Probablemente sean las mas utilizadas de una aplicación, en una interfaz MDI son las hojas de trabajo, y pueden tener un menú asociado.

Ventana Hija (Child!)

Representa una ventana dependiente de la ventana en la que ha sido abierta, siendo esta la ventana padre. Solo puede moverse dentro de la ventana padre y no se le puede asociar un menú.

Ventana Emergente (Popup!)

Las ventanas de este tipo son secundarias y se abren desde una ventana principal. Se utilizan normalmente como ventanas de soporte. Por ejemplo, si se pide que se ingrese el número de un cliente, la ventana emergente podría mostrar todos los números de clientes con sus nombres, para elegir uno en particular.

Ventana Modal (Response!)

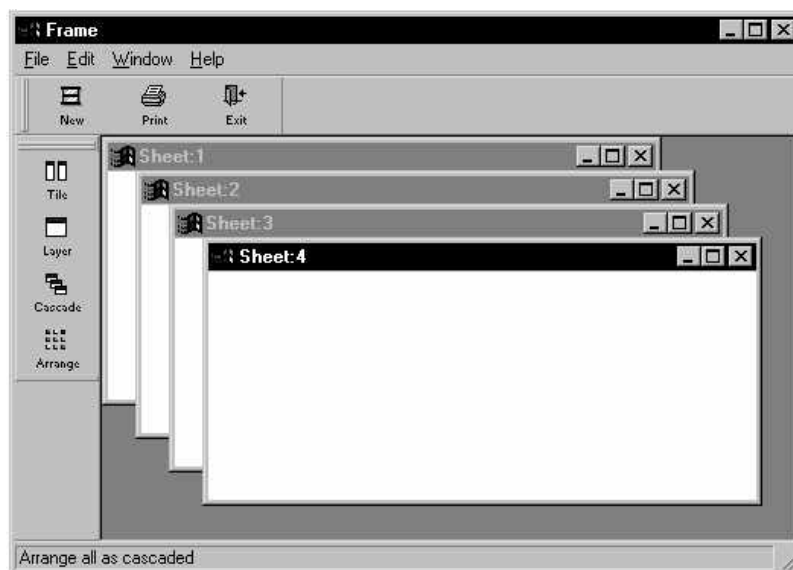
Es una ventana que sirve fundamentalmente para avisar sobre errores o dar información al usuario. Es una ventana que no puede perder el foco, y bloquea la aplicación todo el tiempo que se encuentre abierta.

Descripción de una Interfaz MDI

Es una aplicación con una interfaz para múltiples documentos. Una aplicación en donde los usuarios trabajan dentro de una ventana marco en donde las diversas tareas se distribuyen en múltiples hojas de trabajo. Es usual que los usuarios requieran en las aplicaciones habilitar varias tareas al mismo tiempo.

Un marco MDI es una ventana con diversas partes: un menú bar, un área del cliente, hojas de trabajo, y usualmente una barra de herramientas y una área de estado donde poder mostrar una micro-ayuda.

Una hoja MDI es una ventana que se abre en el área del cliente de un marco MDI. Usted puede usar cualquier tipo de ventana, excepto las de tipo MDI Frame como hoja MDI.



Ejercicio 01

Iniciar una nueva aplicación con las siguientes especificaciones:

- Objeto Aplicación : SistNotas
- Librería : SistNotas.PBL



La librería debe ser creada en su carpeta de trabajo.

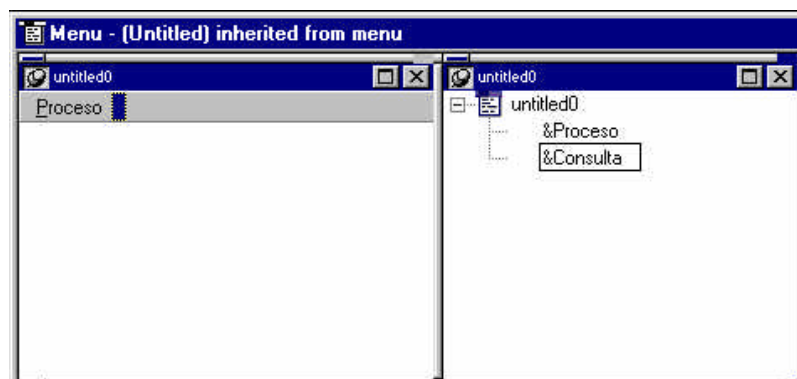
Creación de Menús

Vamos a ver en forma práctica la creación de un menú, para lo cual previamente usted debe abrir el Menu Painter con un nuevo objeto menú.

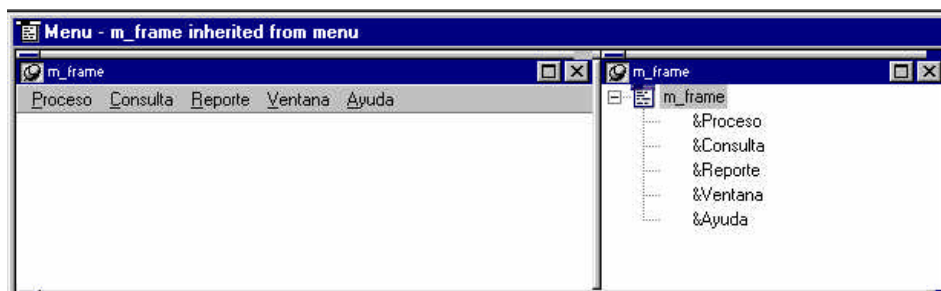
Ejercicio 02

Crear la barra de menú. Los pasos que debe seguir son los siguientes:

1. Del menú **Insert** debe ejecutar el comando **Submenu Item** e ingresar el siguiente texto: **&Proceso**, luego presionar la tecla **Enter**.
2. Del menú **Insert** debe ejecutar el comando **Menu Item At End** e ingrese el siguiente texto: **&Consulta**, luego presionar la tecla **Enter**.



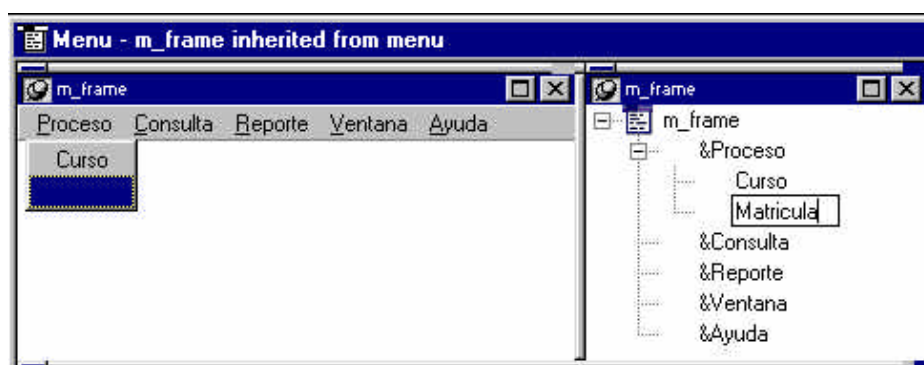
3. Repita la ejecución del comando **Menu Item At End** hasta completar la barra de menú.
4. Luego debe grabarlo con el nombre **m_frame**.



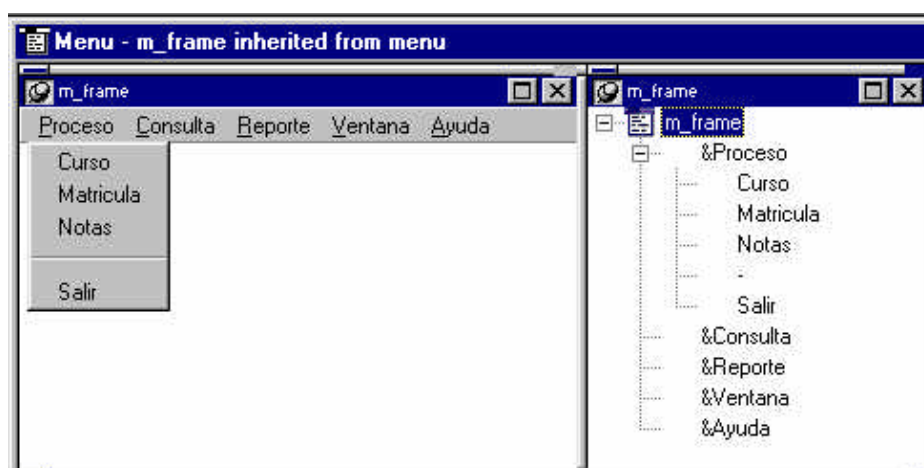
Ejercicio 03

Definir las opciones del menú Proceso. Los pasos que debe seguir son los siguientes:

1. Debe iluminar la opción proceso.
2. Del menú **Insert** debe ejecutar el comando **Submenu Item** e ingresar el siguiente texto: **Curso**, luego presionar la tecla **Enter**.
3. Del menú **Insert** debe ejecutar el comando **Menu Item At End** e ingresar el siguiente texto: **Matricula**, luego presionar la tecla **Enter**.



4. Repita la ejecución del comando **Menu Item At End** hasta completar el menú Proceso.
5. Luego debe grabarlo.



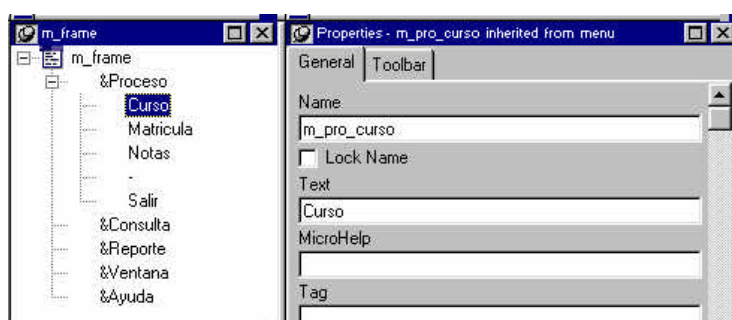
Ejercicio 04

Redefinir el nombre de los comandos del menú Proceso. Para este ejercicio debe guiarse de la siguiente tabla:

| Barra de Menú | Comando | Nombre |
|---------------|-----------|-----------------|
| Proceso | | m_proceso |
| | Curso | m_pro_curso |
| | Matricula | m_pro_matricula |
| | Notas | m_pro_notas |
| | - | m_pro_linea |
| | Salir | m_pro_salir |

Por ejemplo para cambiar el nombre del comando **Curso** debe seguir los siguientes pasos:

1. Debe iluminar el comando Curso.
2. En la ventana de propiedades, desbloquear el nombre.
3. Asignarle el nuevo nombre.
4. Volver a bloquear el nombre.



Repita los pasos 1, 2, 3 y 4 para los otros comandos, y no se olvide de grabar los cambios.

Ejercicio 05

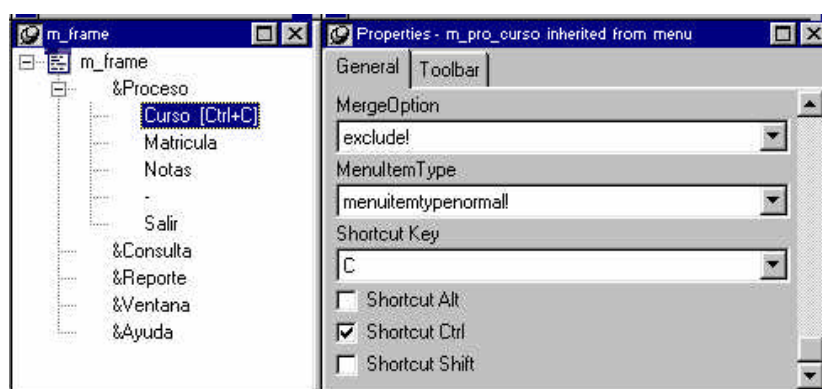
Complete el menú según el siguiente cuadro.

| Barra de Menú | Comando | Nombre |
|-----------------|--------------|-------------------|
| Consulta | | m_consulta |
| | Por curso | m_con_curso |
| | Por Alumno | m_con_alumno |
| Reporte | | m_Reporte |
| | Alumnos | m_rep_alumnos |
| | Notas | m_rep_notas |
| Ventana | | m_ventana |
| | Vertical | m_ven_vertical |
| | Horizontal | m_ven_horizontal |
| | Capas | m_ven_capas |
| | Cascada | m_ven_cascada |
| Ayuda | | m_ayuda |
| | Indice | m_ayu_indice |
| | - | m_ayu_linea |
| | Acerca de... | m_ayu_acerca |

Ejercicio 06

Usted puede crear atajos para los comandos mas utilizados del menú. Por ejemplo crear el atajo **Ctrl + C** para la secuencia **Proceso/Curso**, los pasos que debe seguir son:

1. Debe iluminar el comando **Curso** en el menú **Proceso**.
2. En la ficha General del Panel de Propiedades:
 1. Marcar la casilla de verificación **Shortcut Ctrl..**
 2. En el desplegable **Shortcut Key**, seleccionar la letra **C**.
3. Grabar los cambios.



Puede repetir los pasos para los comandos que usted considere necesarios.

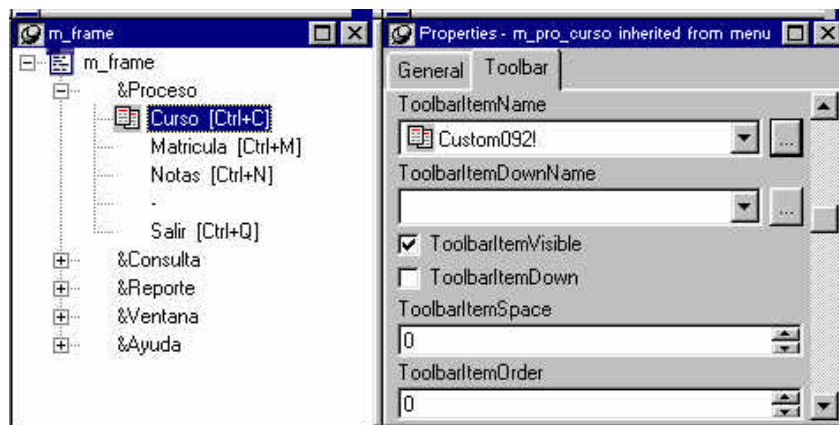


En www.LibrosDigitales.NET encontrará la solución de este ejercicio.

Ejercicio 07

Crear una barra de botones. Usted puede asociar cada comando del menú con un gráfico que se verá como un botón en una barra de botones que se creará de manera automática. Para asociar el gráfico a un comando del menú debe seguir los siguientes pasos:

1. Debe iluminar el comando del menú.
2. En la ficha **Toolbar** del Panel de Propiedades:
 - El campo **ToolbarItemText** define el **Tip** que aparece cuando apunta al botón con Mouse.
 - El campo **ToolbarItemName** define el gráfico asociado con el comando del menú. Puede seleccionar uno predefinido ó seleccionar uno de un archivo haciendo clic en el botón que aparece a su costado.
 - El campo **ToolbarItemSpace** define el espacio entre el botón actual y al anterior.
 - El campo **ToolbarItemOrder** define el orden en que aparecerán los botones en la barra.
 - El campo **ToolbarItemBarIndex** define a que barra pertenece el botón, por ejemplo puede crear dos barras de botones, la segunda tendría valor dos en este campo.

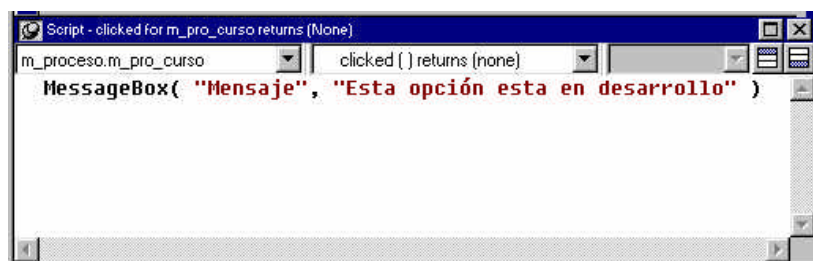


3. Después de establecer los gráficos para los comandos que usted considere necesarios, no se olvide de grabar los cambios.



Ejercicio 08

Programemos en comando **Curso** del menú **Proceso**, para que muestre un mensaje. La programación se realiza en el **Panel Script**, se selecciona el comando, luego el evento y se programa.



Ejercicio 09

Crear una ventana de tipo MDI con micro ayuda, asóciela el menú marco **m_frame** y grábela con el nombre **w_frame**. Ahora ejecute la ventana desde el comando **Run/Preview** del menú **File**, y pruebe el script creado en el ejercicio anterior.



En www.LibrosDigitales.NET encontrará la solución de este ejercicio, la librería es sistnotas01.pbl.

Manejo de Ventanas

Función: Open()

Abre una ventana. Open muestra la ventana y hace todas sus propiedades y controles disponibles en el script.

Sintaxis

```
Open ( WindowVar {, Parent } )
```

Descripción de Argumentos

| Argumento | Descripción |
|-----------|--|
| WindowVar | El nombre de la ventana que usted necesita mostrar. Usted puede especificar un objeto ventana definido en el Window Painter (el cual es un tipo de dato Window) o una variable de un tipo de dato Window discreto. Open guarda una referencia a la ventana abierta en WindowVar. |
| Parent | Este argumento es opcional, y solo se debe utilizar cuando la ventana que se quiere abrir es de tipo Hija o Emergente. Este argumento es la ventana Padre de la ventana Hija o Emergente que se esta abriendo. |

Valor de Retorno

Retorna un valor de tipo Integer. Retorna 1 si se ejecuta en forma satisfactoria, -1 si ocurre un error. Si algún argumento es NULL, retorna también NULL.

Ejemplos

El siguiente ejemplo abre una instancia de la ventana w_cursos.

```
Open( w_cursos )
```

Las siguientes sentencias abren una instancia de una ventana de tipo w_cursos.

```
w_cursos wi_cursos
Open( wi_cursos )
```


El siguiente script abre una instancia de una Ventana Hija de nombre w_datos y la ventana w_empleado es la Ventana Padre.

```
w_datos wi_datos
Open( wi_datos, w_empleado )
```

El siguiente ejemplo abre dos instancias de la ventana w_empleado.

```
w_empleado w_e1, w_e2
Open(w_e1)
Open(w_e2)
```

Función: OpenSheet()

Abre una hoja dentro de una Ventana Marco MDI y crea un menú item para la hoja seleccionada en el menú especificado.

Sintaxis

```
OpenSheet ( SheetRefVar {,WindowType },MDIFrame {,Position {,ArrangeOpen }})
```

Descripción de Argumentos

| Argumento | Descripción |
|-------------|---|
| SheetRefVar | Nombre de una variable de cualquier tipo de ventana que no sea un Marco MDI. |
| WindowType | Es un argumento opcional. Una cadena cuyo valor es el tipo de ventana que se desea abrir. El tipo de dato de WindowType debe ser igual o un descendiente de SheetRefVar. |
| MDIFrame | El nombre de la Ventana MDI Marco. |
| Position | Es un argumento opcional. El número del menú item (en el menú asociado con la hoja) en el cual usted necesita tener el nombre de las hojas abiertas. Los items del Menú Bar están enumerados desde la izquierda, empezando en el valor 1. |
| ArrangeOpen | Es un valor opcional. Es un valor de tipo enumerado que especifica como se arreglará la hoja en el marco MDI en relación con la otras hojas abiertas, los valores que puede tomar son: <ul style="list-style-type: none">▪ Cascaded! (Valor por defecto)▪ Layered!▪ Original! |

Valor de Retorno

Retorna un valor de tipo Integer. Retorna 1 si se ejecuta en forma satisfactoria, -1 si ocurre un error. Si algún argumento es NULL, retorna también NULL.

Ejemplos

Ejemplo 01

```
OpenSheet(w_cursos, w_frame, 2, Original!)
```

Ejemplo 02

```
window w_child  
OpenSheet(w_child, "w_cursos", w_frame, 4, Layered!)
```

Función: Close()

Cierra una ventana.

Sintaxis

```
Close ( WindowName )
```

Descripción de Argumentos

| Argumento | Descripción |
|------------|---|
| WindowName | Nombre de la ventana que se desea cerrar. |

Valor de Retorno

Retorna un valor de tipo Integer. Retorna 1 si se ejecuta en forma satisfactoria, -1 si ocurre un error. Si WindowName es NULL, retorna también NULL.

Ejemplo

```
Close(w_cursos)
```

Sentencia Halt

Termina una aplicación. La cláusula **CLOSE** hace que se desencadene el evento **Close** del objeto aplicación.

Sintaxis

```
HALT {CLOSE}
```

Uso de Pronombres

PowerScript tiene pronombres que permiten a usted hacer una referencia general a objetos y controles. Estos son los pronombres:

- This
- Parent
- ParentWindow
- Super

Usted puede usar los pronombres en el script de cualquier función o evento donde necesita hacer referencia al nombre de un objeto. Por ejemplo puede usar el pronombre para:

- Causar un evento en un objeto o control.
- Manipular o cambiar un objeto o control.
- Obtener o cambiar los valores de una propiedad.

Todo pronombre tiene un significado y uso que se describe a continuación:

| Pronombre | Aplicable a | Se refiere a |
|--------------|---|---|
| This | Ventana, Objeto de Usuario Custom, Menú, Objeto Aplicación y Controles. | Al objeto o control mismo. |
| Parent | Control en una ventana. | Ventana que contiene al control. |
| | Control en un objeto de usuario. | Objeto de usuario que contiene al control. |
| | Menú | Item en el menú en el nivel inmediatamente superior. |
| ParentWindow | Menú. | Ventana a la que se encuentra asociado el menú en tiempo de ejecución. |
| Super | Objeto o Control Descendiente. | Parent. |
| | Ventana o Objeto de Usuario | Al ancestro inmediato de la ventana descendiente o del objeto de usuario. |
| | Un control en una ventana descendiente o en un objeto de usuario. | El ancestro inmediato de la ventana Parent del control o del objeto de usuario. |

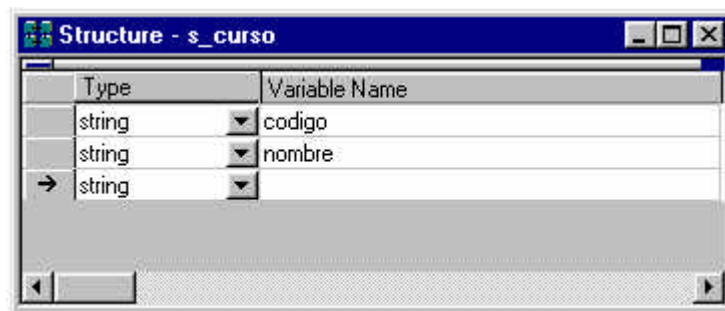
Estructuras

Las estructuras son objetos que agrupan un conjunto de variables relacionadas como una sola unidad.

Pueden ser creadas a nivel de objeto, por ejemplo dentro de una ventana. También pueden ser globales, creadas como objetos independientes.

Ejercicio 10

Crear la estructura global **s_curso**, como se indica en el siguiente gráfico.



En www.LibrosDigitales.NET encontrará la solución de este ejercicio, la librería es sistnotas01.pbl.

Paso de Parámetros a una Ventana

Sintaxis 1

```
OpenWithParm ( windowvar, parameter {, parent } )
```

Sintaxis 2

```
OpenSheetWithParm ( sheetrefvar, parameter {, windowtype },  
mdiframe {, position {, arrangeopen } } )
```

Las funciones **OpenWithParm** y **OpenSheetWithParm** permiten pasar un parámetro a la ventana que se está abriendo. El parámetro se guarda en el objeto **Message** y puede ser uno de los siguientes tipos:

- String
- Numeric
- PowerObject

El objeto Message tiene tres propiedades para guardar el dato. Dependiendo el tipo de dato que se le pasa, en el evento **open** de la ventana debe chequear una de las siguientes propiedades:

| Propiedad del Objeto Message | Tipo de Dato del Argumento |
|------------------------------|--|
| Message.DoubleParm | Numérico |
| Message.PowerObjectParm | PowerObject (PowerBuilder Object, incluido estructuras definidas por el usuario) |
| Message.StringParm | String |

Ejemplo 01

```
OpenWithParm( w_alumnos, "PB1" )
```

Script – Open For w_alumnos

```
String Dato  
Dato = Message.StringParm  
• • •  
• • •
```

Ejemplo 02

```
OpensheetWithParm( w_alumnos,"PB1",w_frame, 5, Original! )
```

Script – Open For w_alumnos

```
String Dato  
Dato = Message.StringParm  
• • •  
• • •
```

Ejemplo 03

```
s_curso s_origen  
s_origen.Codigo = "PB1"  
s_origen.Nombre = "PowerBuilder - Nivel Básico"  
OpensheetWithParm( w_alumnos, s_origen, w_frame, 5, Original! )
```

Script – Open For w_alumnos

```
S_curso s_destino  
S_destino = Message.PowerObjectParm  
• • •  
• • •
```



La variable utilizada en el evento **open** de la ventana puede ser una variable local o de tipo instancia, dependiendo si se necesita el parámetro en otros eventos de algún otro control.

Recibir un Parámetro de una Ventana

Solo se puede recibir parámetros de ventanas de tipo Modal (Response!). Para cerrar la ventana se debe usar la función **CloseWithReturn**. Esta función lo que hace es copiar el parámetro en el objeto Message, así que se debe verificar la propiedad que corresponde al dato que se espera.

Sintaxis

```
CloseWithReturn ( windowname, returnvalue )
```

Ejemplo

```
String user  
Open(w_user)  
User = Message.StringParm  
If User = "admin" Then  
    Open(w_admin)  
Else  
    Open(w_frame)  
End If
```

En este ejemplo, si en la ventana w_user ha ingresado el administrador (admin), entonces se abre la ventana de administración (w_admin), de lo contrario se abre la ventana de operación (w_frame).

Manejo de Variables

Alcance

El alcance de una variable determina su visibilidad en la aplicación. En el siguiente cuadro explicamos el alcance que puede tomar una variable.

| Alcance | Descripción |
|----------|--|
| Global | Es accesible en cualquier parte de la aplicación. Es independiente del objeto donde se define. |
| Instance | Pertenece a un objeto y esta asociado con una instancia del objeto. Las variables de tipo Instance se crean en cada instancia del objeto en el cual han sido definidas, y se tendrá acceso a estas variables desde el script del objeto y los controles asociados con el objeto. |
| Shared | Pertenecen a la definición de un objeto y existe a través de todas las instancias del objeto. Las variables Shared retienen el valor cuando un objeto es cerrado. Solo son accesibles en el script del objeto y los controles asociados con el objeto. |
| Local | Es una variable temporal a la cual se puede acceder solo en el script donde es definida. Cuando el script finaliza la variable deja de existir. |

Global declarations

Las variable globales pueden ser definidas en una ventana, un objeto de usuario, un menú, o en el PowerScript painter para cualquier objeto.

Instance y shared declaraciones

Instance y shared variables pertenecen a un tipo particular de objeto: ventana, objeto de usuario, menú, o el objeto aplicación. Antes de que usted pueda declarar, necesita abrir el objeto en el painter.

Local declaraciones

Estas variables se declaran en el script al que pertenecen.

Accediendo a una Base de Datos

Objetivos

En este primer módulo se desarrollaran los siguientes puntos:

- ✓ Descripción de la Base de Datos a Utilizar
- ✓ El DataBase Painter
- ✓ Creación de una Base de Datos
- ✓ Creación de tablas
- ✓ Creación de claves primarias
- ✓ Creación de claves foráneas
- ✓ Creación de Fuente de Datos (DSN)
- ✓ Verificar las Fuentes de Datos
- ✓ Perfiles de Bases de Datos
- ✓ Conexión de una Aplicación PowerBuilder con una Base de Datos

Descripción de la Base de Datos a Utilizar

La base de datos a utilizar se llama SistNotas.DB y por razones académicas solo va a tener dos tablas, que pasamos a describir a continuación.

Tabla: Curso

Esta tabla sirve para guardar la información de cursos.

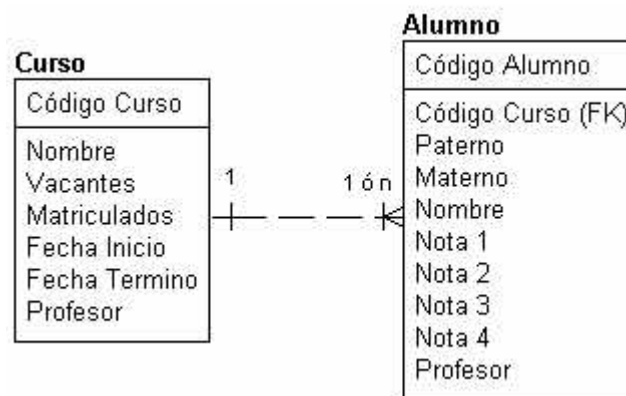
Tabla: Alumno

Esta tabla sirve para guardar datos de los alumnos, el curso en el que se encuentran matriculados, con sus respectivas notas.

Relación entre las tablas

Como mencione al inicio de esta sección, por razones académicas esta base de datos solo va a tener dos tablas, y la cardinalidad es la siguiente:

- Un curso puede tener uno ó muchos alumnos matriculados.
- Un alumno solo puede estar matriculado en un solo curso.



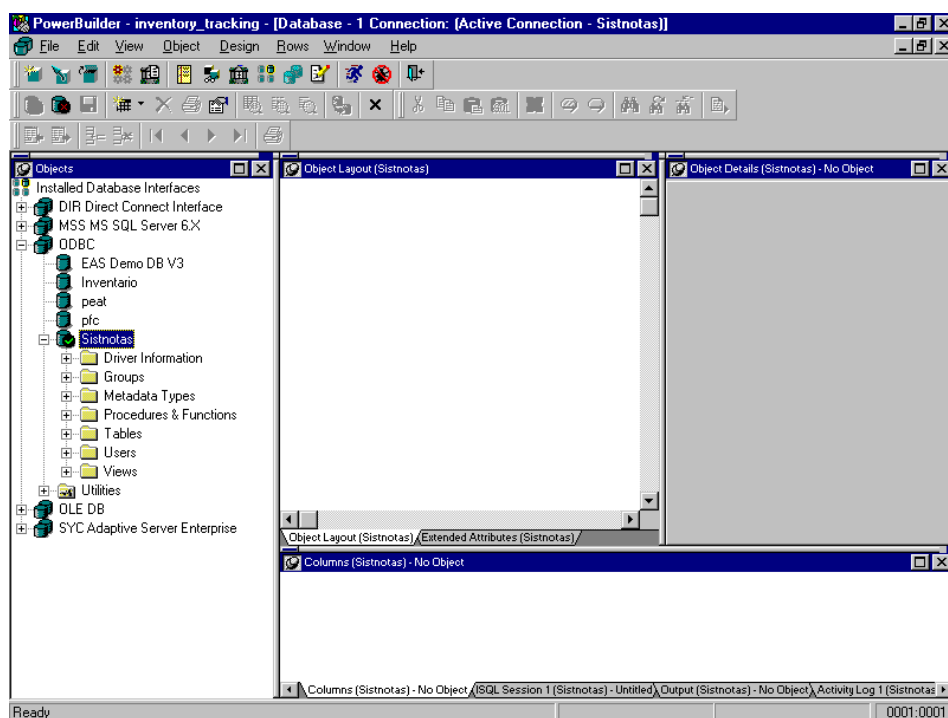
En www.LibrosDigitales.NET encontrará una guía completa sobre **Modelamiento de Base de Datos**.

El Database Painter

Usted puede realizar la mayor parte del trabajo de su base de datos usando el Database Painter. Cuando usted abre este painter, PowerBuilder muestra el nombre de las tablas, vistas, y otras entidades contenidas en la base de datos y que usted tiene acceso. Puede trabajar con la base de datos actual hasta que se conecte con otra base de datos. Puede tener varias conexiones abiertas al mismo tiempo, no obstante, sólo una puede estar activa.

Tareas que puede llevar a cabo desde el painter:

- Configurar fuentes de datos.
- Crear y modificar perfiles a bases de datos.
- Conectarnos a bases de datos.
- Crear tablas, vistas, claves, índices.
- Modificar los objetos de la base de datos.
- Ver gráficamente la relación entre las tablas.
- Manipular datos.
- Construir y ejecutar sentencias SQL.
- Definir y modificar atributos extendidos.
- Definir atributos extendidos para las columnas.
- Acceso a utilitarios de bases de datos.

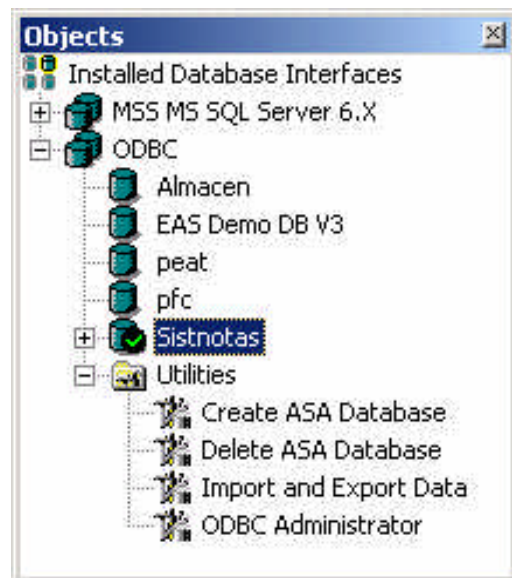
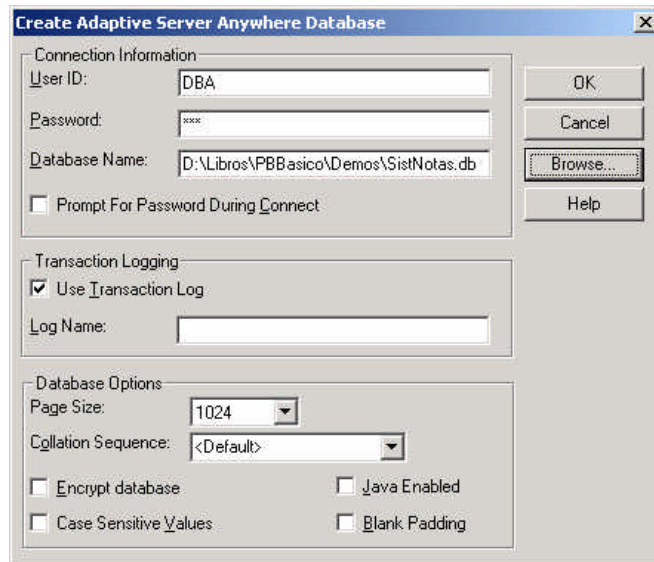


Creación de la Base de Datos

Ejercicio 01

Crear la base de datos SistNotas.DB. Para crear la base de datos debe seguir los siguientes pasos:

1. Abrir el **DataBase Painter**.
2. En el panel **Objects**, expanda la rama ODBC/Utilities y ejecute el comando **Create ASA Database**.
3. En el diálogo **Create Adaptive Server Anywhere Database** indique la ruta y el nombre de la base de datos (le recomiendo que utilice el botón Browse) y luego haga clic en el botón Ok. El nombre por defecto del usuario es **DBA** y su contraseña es **SQL**.
4. Finalmente tendremos la base de datos creada, usted podrá darse cuenta por que en el panel **Objects**, dentro de **ODBC** tendrá una rama SistNotas que corresponde a la base de datos recientemente creada.



En www.LibrosDigitales.NET encontrará una guía completa sobre **Modelamiento de Base de Datos**.

Creación de Tablas

Tablas a Crear

Tabla: Curso

| Column Name | Data Type | Width | Null | Header/Label |
|-------------|-----------|-------|------|--------------|
| cur_cod | char | 3 | No | Código |
| cur_nom | varchar | 30 | No | Nombre |
| cur_vac | integer | | No | Vacantes |
| cur_mat | integer | | No | Matriculados |
| cur_fec_ini | date | | No | Inicia |
| cur_fec_fin | date | | No | Finaliza |
| cur_pro | varchar | 30 | No | Profesor |

Tabla: Alumno

| Column Name | Data Type | Width | Null | Header/Label |
|-------------|-----------|-------|------|--------------|
| alu_cod | char | 5 | No | Código |
| alu_pat | varchar | 15 | No | Paterno |
| alu_mat | varchar | 15 | No | Materno |
| alu_nom | varchar | 15 | No | Nombre |
| cur_cod | char | 3 | No | Curso |
| alu_n1 | float | | Yes | Nota 1 |
| alu_n2 | float | | Yes | Nota 2 |
| alu_n3 | float | | Yes | Nota 3 |
| alu_n4 | float | | Yes | Nota 4 |
| alu_pr | float | | Yes | Promedio |

Ejercicio 02

Crear la tabla Curso. Para crear la tabla curso debe seguir los siguientes pasos:

1. Ejecutar el comando **Create New Table**, que lo puede hacer del desplegable del PainterBar ó del menú **Object/Insert**.
2. Luego en el panel **Columns** definir las columnas de la nueva tabla.

| Column Name | Data Type | Width | Dec | Null | Default |
|-------------|-----------|-------|-----|------|---------|
| cur_cod | char | 3 | | No | (None) |
| cur_nom | varchar | 30 | | No | (None) |
| cur_vac | integer | | | No | (None) |
| cur_mat | integer | | | No | (None) |
| cur_fec_ini | date | | | No | (None) |
| cur_fec_fin | date | | | No | (None) |
| cur_pro | varchar | 30 | | No | (None) |

3. Después de haber definido las columnas, debe grabar la tabla con el comando **Save** que lo puede ejecutar del **PainterBar** ó del menú **File**.

Create New Table

Owner Name: dba

Table Name: Curso

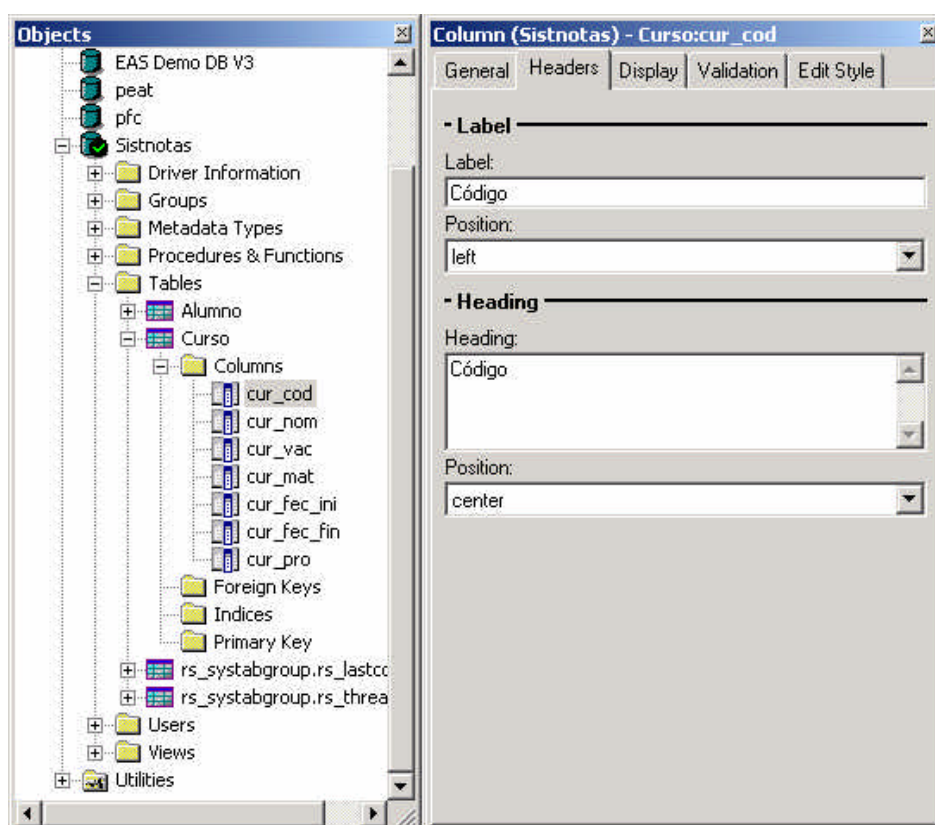
OK Cancel

Ejercicio 03

Siguiendo los pasos del ejercicio anterior proceda a crear la tabla Alumno.

Propiedades Extendidas

Podemos especificar las propiedades extendidas de los objetos, tales como tablas y columnas, tal como se muestra en el siguiente gráfico. En el panel Objects, debe hacer doble clic en la columna que desea especificar sus propiedades extendidas.



Ejercicio 04

Defina las propiedades extendidas de las columnas de las tablas creadas.

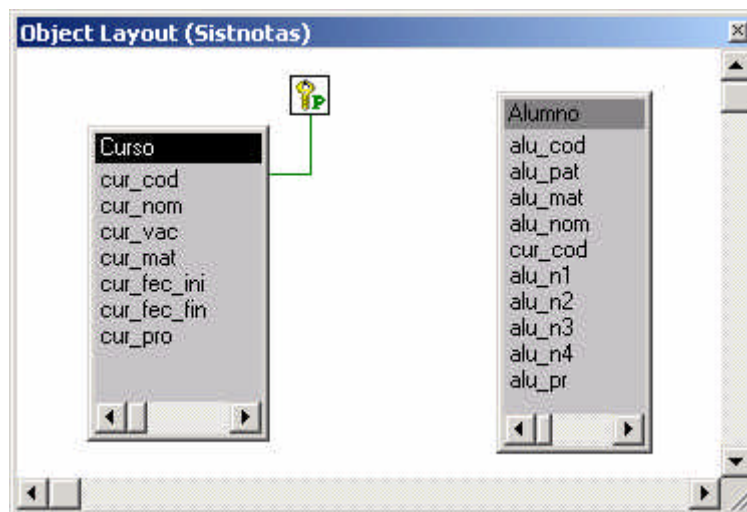
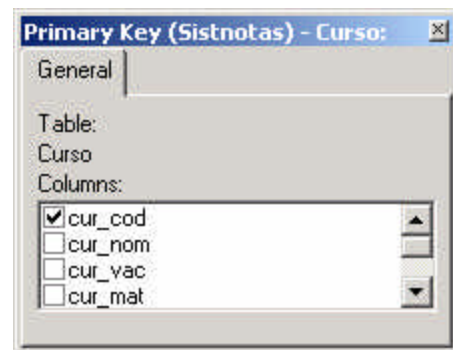
Creación de Claves Primarias

Un aspecto muy importante en una tabla es la clave primaria que identifica cada uno de los registros de manera única, a continuación veremos en forma práctica como se crea una clave primaria.

Ejercicio 05

Crear la Clave Primaria para la tabla Curso. Los pasos que debe seguir son los siguientes:

1. Seleccionar la tabla curso, lo puede realizar en el panel **Objects** ó en el panel **Object Layout**.
2. Ejecutar el comando **Create New Primary Key**, lo puede realizar del desplegable del **PainterBar** ó del menú **Object/Insert**.
3. En el panel **Primary Key**, marcar la columna (las columnas cuando sea el caso) que corresponde a la clave primaria.
4. Actualizar la base de datos.



Ejercicio 06

Siguiendo los mismos pasos que se han utilizado para la tabla Curso, proceda a crear la clave primaria para la tabla Alumno.

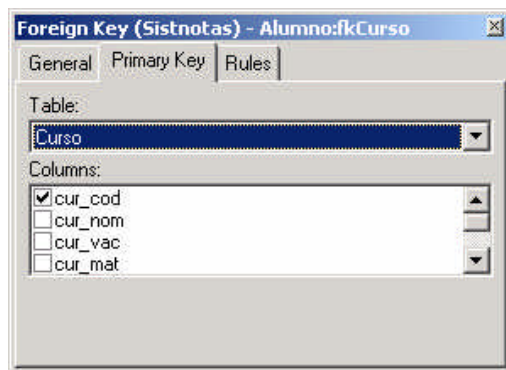
Creación de Claves Foráneas

La clave foránea de una tabla es otro aspecto muy importante, ya que no solo define la relación que tiene con otras tablas de la base de datos, si no que también se utiliza para definir reglas de integridad referencial. A continuación veremos en forma práctica como se crea una clave foránea.

Ejercicio 07

Crear la Clave Foránea para la tabla Alumno. Los pasos que debe seguir son los siguientes:

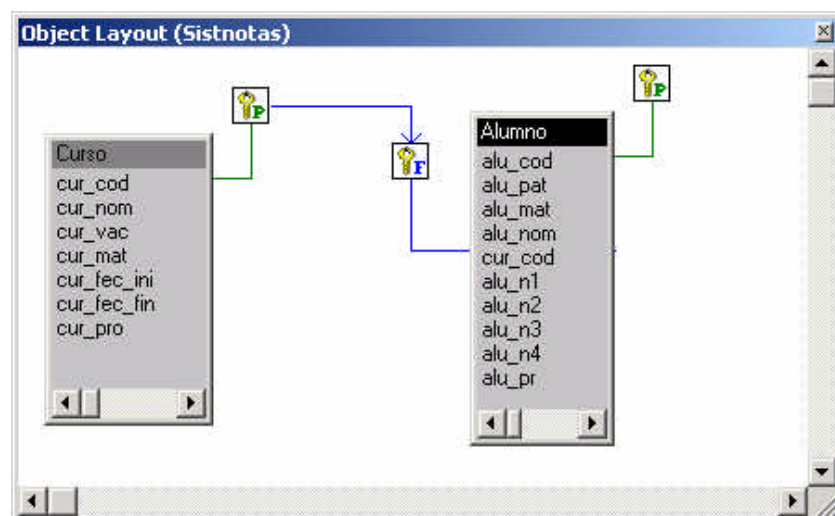
1. Seleccionar la tabla alumno, lo puede realizar en el panel **Objects** ó en el panel **Object Layout**.
2. Ejecutar el comando **Create New Foreign Key**, lo puede realizar del desplegable del **PainterBar** ó del menú **Object/Insert**.



3. En la ficha **General** del panel **Foreign Key**, seleccionar la columna de la tabla Alumno que corresponde a la clave foránea, y asígnele un identificador, por ejemplo **fkCurso**.

4. En la ficha **Primary Key** del panel **Foreign Key** seleccionar la tabla Curso.

5. Actualizar la base de datos.



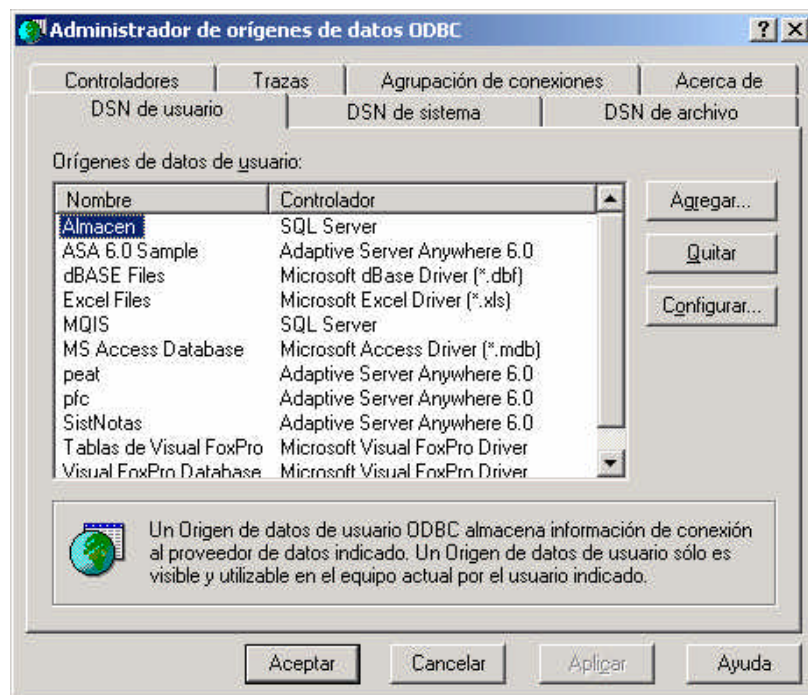
Creación de Fuente de Datos (DSN)

Una fuente de datos (DSN) es una lista de parámetros para la conexión con una base de datos utilizando ODBC.

Creación de una Fuente de Datos (DSN)

Los pasos que debe seguir son:

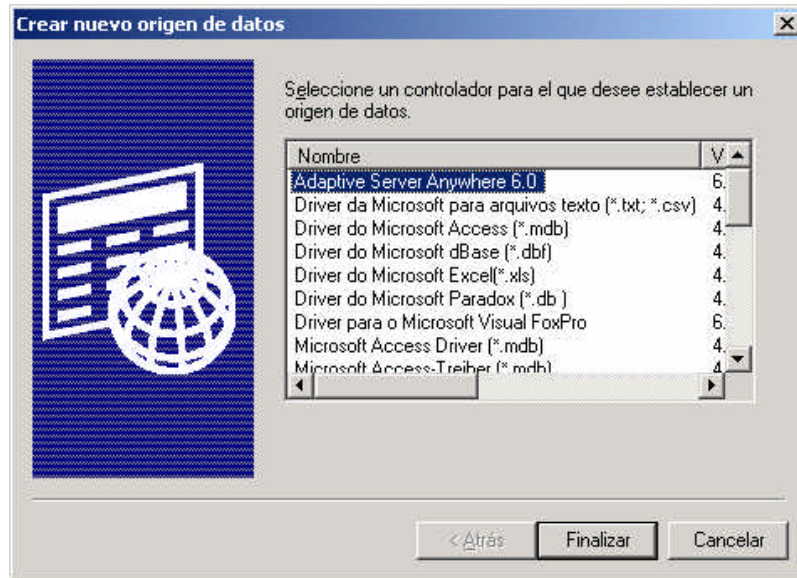
1. Cargar el Administrador ODBC, lo puede realizar desde el panel de control ó desde los utilitarios de **Database Painter**.



2. Seleccionar el tipo de fuente de datos que desea crear, de usuario, de sistema ó de archivo.
3. Ejecutar el botón **Agregar**.



Pronto se ampliará éste tema, no olvide revisar la actualización en www.LibrosDigitales.NET



4. En el diálogo **Crear nuevo origen de datos**, seleccionar el controlador según la base de datos a la cual nos vamos va conectar, y luego hacer clic en el botón **Finalizar**.
5. Luego, según el controlador que ha seleccionado usted tendrá un nuevo diálogo donde ingresará los parámetros que le solicita, después debe ejecutar el diálogo haciendo clic en el botón **Aceptar**.
6. Finalmente debe cerrar el **Administrador ODBC** haciendo clic en el botón **Aceptar**.

Ejercicio 08

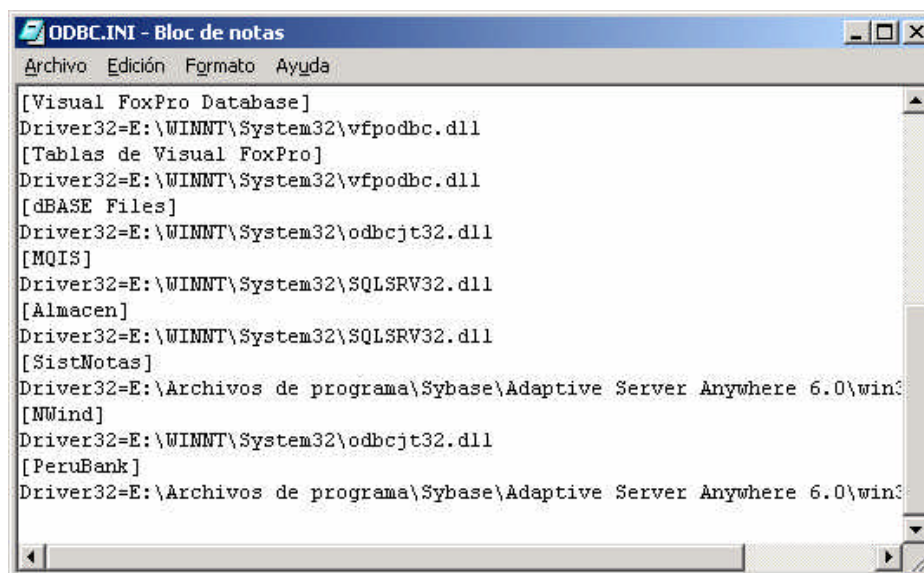
En el laboratorio se le proporcionara dos bases de datos, para que usted proceda a crear dos fuentes de datos, según el siguiente cuadro:

| Base de Datos | Fuente de Datos (DSN) | Comentario |
|---------------|-----------------------|----------------------------|
| NWind.MDB | NWind | Búsquelo en su disco duro. |
| PeruBank.DB | PeruBank | Solicítelo a su profesor. |

Verificar la Fuentes de Datos

Ejercicio 09

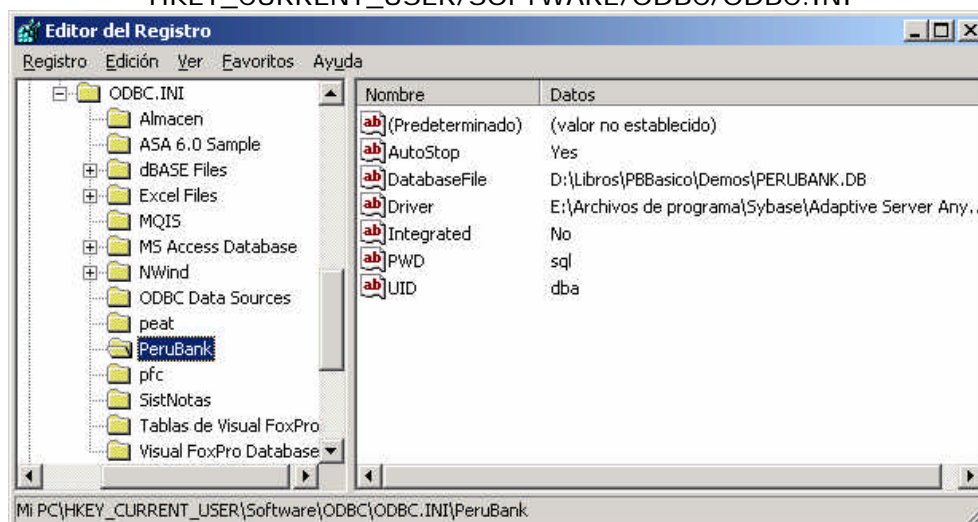
Revisar el archivo ODBC.INI.



Ejercicio 10

Revisar el Registro de Windows.

HKEY_CURRENT_USER/SOFTWARE/ODBC/ODBC.INI



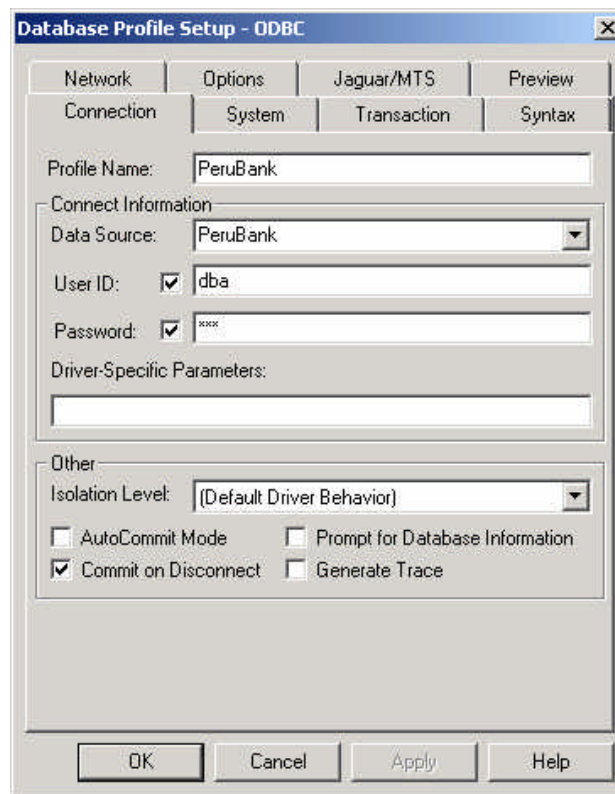
Perfiles de Bases de Datos

Es un nombre que identifica un conjunto de parámetros en el archivo de inicialización de PowerBuilder (PB.INI), y define una conexión particular con una base de datos.

Creación de un nuevo perfil de base de datos

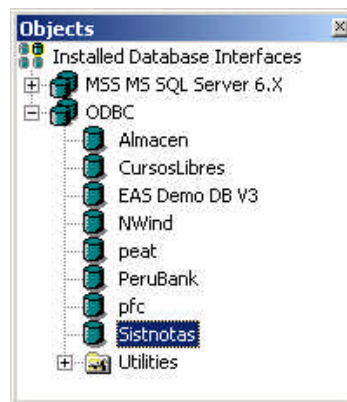
Para crear un nuevo perfil de base de datos, debe seguir los siguientes pasos:

1. En el Database Painter, iluminar la rama ODBC.
2. De la barra de menú ejecute el siguiente comando: Object/Insert/Profile.
3. En el diálogo **Database Profile Setup – ODBC** ingrese los datos necesarios, y luego ejecute el diálogo.



Ejercicio 11

Definir dos perfiles para las dos fuentes de datos creadas anteriormente.



Ejercicio 12

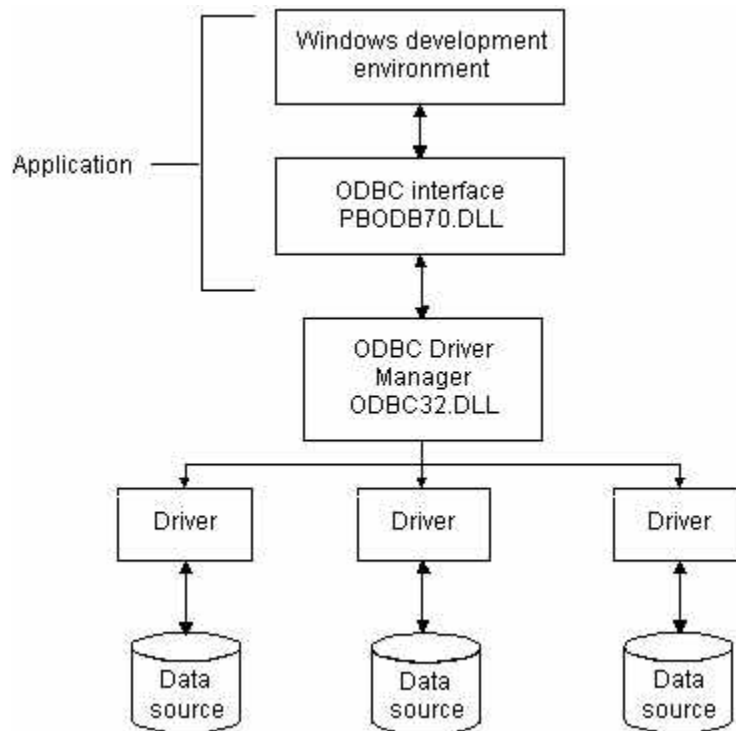
Revisar el archivo PB.INI



Conexión de una Aplicación PowerBuilder con una Base de Datos

Esquema de la conexión

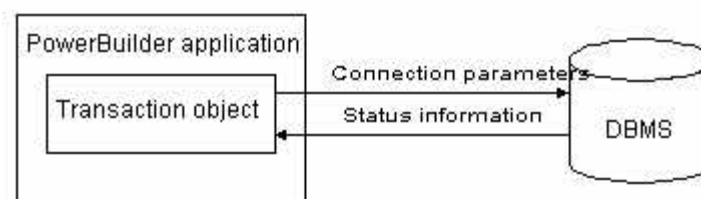
Hay varias posibilidades para que una aplicación PB se conecte con una base de datos, pero ahora lo haremos utilizando ODBC, el siguiente gráfico nos ilustra la forma como se lleva a cabo.



El Objeto de Transacciones

Es un objeto no visual, que establece la comunicación entre una aplicación PB y una base de datos. En el objeto de transacciones especificamos los parámetros que PowerBuilder utiliza para conectarse con la base de datos.

El siguiente gráfico ilustra como una aplicación utiliza el objeto de transacciones para conectarse con la base de datos.



Objeto de Transacciones por Defecto

Cuando usted inicia la ejecución de una aplicación, PB crea un objeto de transacciones, que es global y por defecto, el nombre de este objeto es SQLCA (SQL Communications Area). Usted puede utilizar este objeto para su aplicación (lo más usual) ó crear uno en particular.

Propiedades del Objeto de Transacciones

La siguiente tabla describe cada propiedad del objeto de transacciones. Para cada una de las diez propiedades de conexión, también se lista el campo equivalente en el diálogo de Configuración de Perfiles de Bases de Datos.

| Propiedad | Tipo De Dato | Descripción | En el Database profile |
|---------------|--------------|--|------------------------|
| DBMS | String | Tipo de base de datos a la que nos vamos a conectar. | DBMS |
| Database | String | Nombre de la base de datos a la que nos vamos a conectar. | Database Name |
| UserID | String | El nombre o ID del usuario que se conecta a la base de datos. | User ID |
| DBPass | String | Contraseña del usuario que se conecta con la base de datos. | Password |
| Lock | String | Debería servir para fijar los niveles de bloqueo en la base de datos (Isolation Lavel), pero su significado varía según el DBMS. | Isolation Level |
| LogID | String | El nombre o ID con el que nos conectamos al servidor de bases de datos. | Login ID |
| LogPass | String | La contraseña con la que nos conectamos al servidor de bases de datos. | Login Password |
| ServerName | String | Nombre del servidor donde reside la base de datos. | Server Name |
| AutoCommit | Boolean | Si es True, define que PowerBuilder, después de cada sentencia de modificación de datos envía automáticamente un COMMIT y cierra la transacción para iniciar una nueva. Si es False, es el desarrollador el que se debe encargar de hacer los COMMIT y ROLLBACK. | AutoCommit Mode |
| DBParm | String | Contiene los parámetros específicos para conexión según el DBMS. | DBPARM |
| SQLReturnData | String | Según el DBMS. debe devolver datos específicos al cliente como respuesta a las sentencias que este envía. | -- |
| SQLCode | Long | Valor de retorno de la base de datos que indica el éxito o el fracaso de la operación SQL más reciente llevada a cabo. | -- |

| Propiedad | Tipo De Dato | Descripción | En el Database profile |
|------------|--------------|--|------------------------|
| SQLNRows | Long | Número de filas afectadas por la operación SQL mas reciente. | -- |
| SQLDBCode | Long | Código de error de la base de datos. | -- |
| SQLErrText | String | Mensaje que describe el error ocurrido en la base de datos, y corresponde al código obtenido en SQLDBCode. | -- |

Sentencia: Connect

Sintaxis

```
CONNECT {USING TransactionObject} ;
```

Conecta un objeto de transacciones con una base de datos, el parámetro TransactionObject es obligatorio cuando no se esta utilizando el objeto de transacciones por defecto.

Sentencia: Disconnect

Sintaxis

```
DISCONNECT {USING TransactionObject} ;
```

Desconecta un objeto de transacciones de una base de datos, el parámetro TransactionObject es obligatorio cuando no se esta utilizando el objeto de transacciones por defecto.

Ejercicio 13

Crear el script que permita la conexión y desconexión de una aplicación con la base de datos SistNotas.DB.

Para el desarrollo de este ejercicio utilizaremos la aplicación creada en la sesión anterior.

Programar la conexión

Script - Open For SistNotas

```
// Inicializando el Objeto de Transacciones
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "ConnectString='DSN=SistNotas;UID=dba;PWD=sql'"

// Realizando la conexión
Connect ;

// Verificando la Conexión
If sqlca.sqlcode <> 0 Then
    Beep(2)
    MessageBox( "Error en conexión", &
        "Número de error: " + string(sqlca.sqlDBCode) + char(13) + &
        "Mensaje: " + sqlca.sqlErrText + char(13) + Char(13) + &
        "Programa Abortado" )
    Halt
End If
MessageBox( "Mensaje","Conexión Conforme" )
Open( w_frame )
```

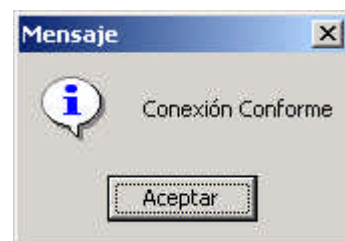
Programar la desconexión

Script - Close for SistNotas

```
// Desconexión de la base de Datos
Disconnect ;
```

Probar la aplicación

Ahora puede ejecutar la aplicación, y si no ha cometido ningún error, debe tener el siguiente mensaje.



En www.LibrosDigitales.NET encontrará la solución de este ejercicio. La librería es SistNotas02.PBL.

El DataWindows

Objetivos

En este primer módulo se desarrollaran los siguientes puntos:

- ✓ Introducción a los DataWindows
- ✓ Estilos de Presentación para un DataWindow
- ✓ Fuentes de Datos para un DataWindow
- ✓ Estilos de Edición de las columnas
- ✓ Propiedades de Actualización de un DataWindow
- ✓ Acciones Predeterminadas para los botones de un DataWindow
- ✓ Funciones del Control DataWindow
- ✓ Validación de Datos
- ✓ Completar la aplicación

Introducción de los DataWindows

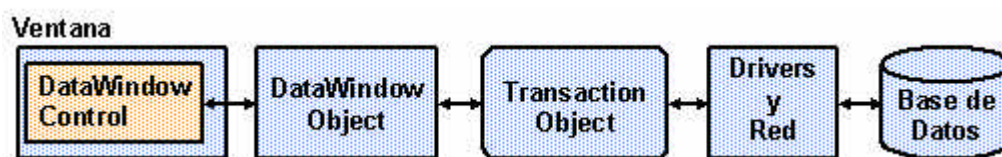
El objeto DataWindow se utiliza para recuperar, presentar, y manipular datos desde una base de datos relacional u otra fuente de datos.

En el Objeto DataWindow es sin duda alguna el principal objeto de PowerBuilder, y el más utilizado, ya que con este objeto podemos realizar múltiples tareas, como por ejemplo:

- Presentación de datos en diferentes formatos.
- Flexibilidad de tratamiento de datos, según convenga; fila por fila, o varias filas a la vez.
- Una misma interfaz y entorno de desarrollo de objetos de acceso a datos e informes.
- Reusabilidad del código generado mediante el uso del mecanismo de herencia.
- Gestión automática de bloqueos y transacciones.
- Generación gráfica de las sentencias de Fuente de Datos.
- Modificación dinámica de todos los aspectos (visuales y de comportamiento) del objeto, en tiempo de ejecución.
- Etc.

Esquema de Funcionamiento

El DataWindow es una clase de acceso y manipulación de datos de la Base de Datos. Trabaja estrechamente con la clase Transaction para desempeñar su trabajo, la cual, a su vez, se apoya en los drivers correspondientes para acceder a la base de datos. Este modo de funcionamiento le permite que sea en la mayoría de los casos portable entre diferentes bases de datos, debido a que representa un nivel de abstracción por encima de los diferentes drivers.

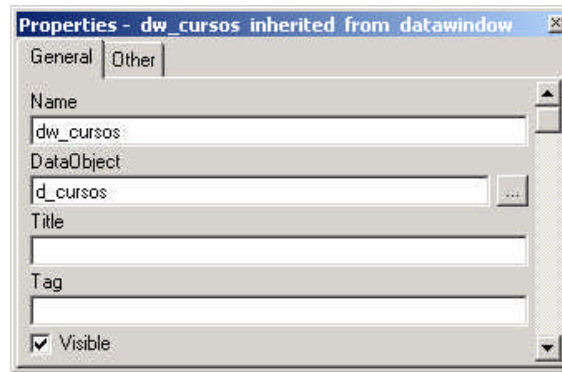


Como podemos apreciar en el gráfico anterior, existe un **Control DataWindow**, que se ubica en la ventana, y un **Objeto DataWindow**; es en el objeto donde definimos la fuente de datos, el estilo de presentación, las propiedades de edición, etc. Por otro lado es el **Control DataWindow** el que tiene todas las funciones que nos permiten tener todo el control sobre los datos que se recuperan.

Podemos afirmar que el Control DataWindow y el Objeto DataWindow forman una pareja inseparable para poder realizar su trabajo, cada uno en forma independiente no nos sirve, también es necesario resaltar que el prefijo para dar nombre a un Objeto DataWindow es **dw_** y para el Control DataWindow es **dw_**.

Para realizar la asociación entre un Objeto DataWindow y un Control DataWindow tenemos dos alternativas:

1. Definir el nombre del Objeto DataWindow como propiedad del Control.

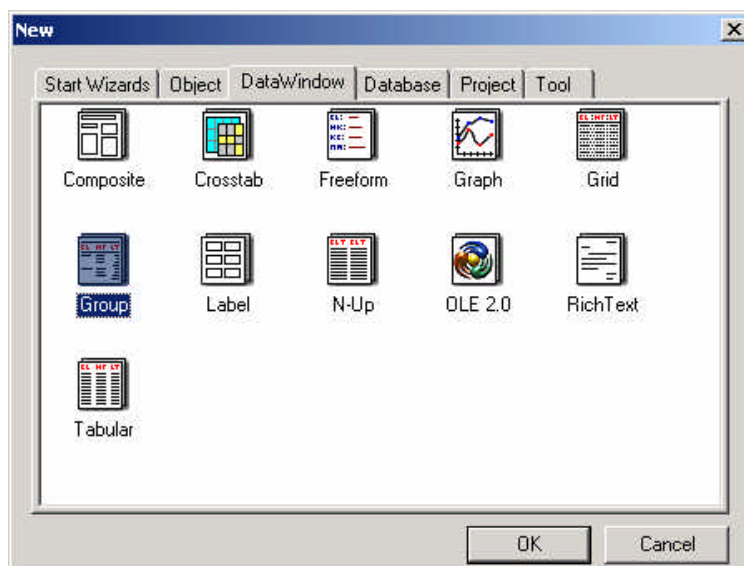


2. Definir el nombre del Objeto DataWindow en tiempo de ejecución, como se puede ver a continuación:

```
dw_curso.DataObject = "d_curso" // Asociación del objeto con el control  
dw_curso.SetTransObject( SQLCA ) // Asociación con el objeto de transacciones  
dw_curso.Retrieve() // Recuperación de datos
```

Estilos de Presentación para un DataWindow

El DataWindow tiene varios estilos de presentación, algunos orientados a la manipulación de datos y otros a la creación de informes, tal como se muestra en el siguiente gráfico.



Fuentes de Datos para un DataWindow

Los datos que se cargan en el DataWindow provienen de una fuente de datos, los tipos que tenemos a disposición los podemos ver en el siguiente gráfico.



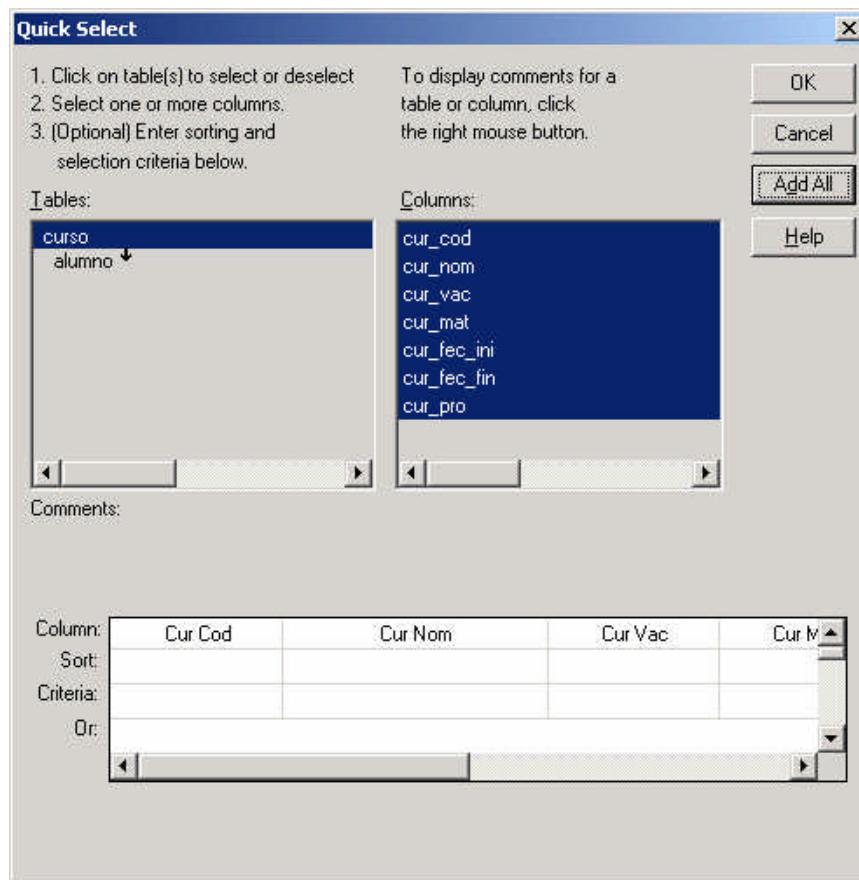
Ejercicio 01

Crear un Objeto DataWindow de nombre **d_man_curso**, con las siguientes especificaciones:

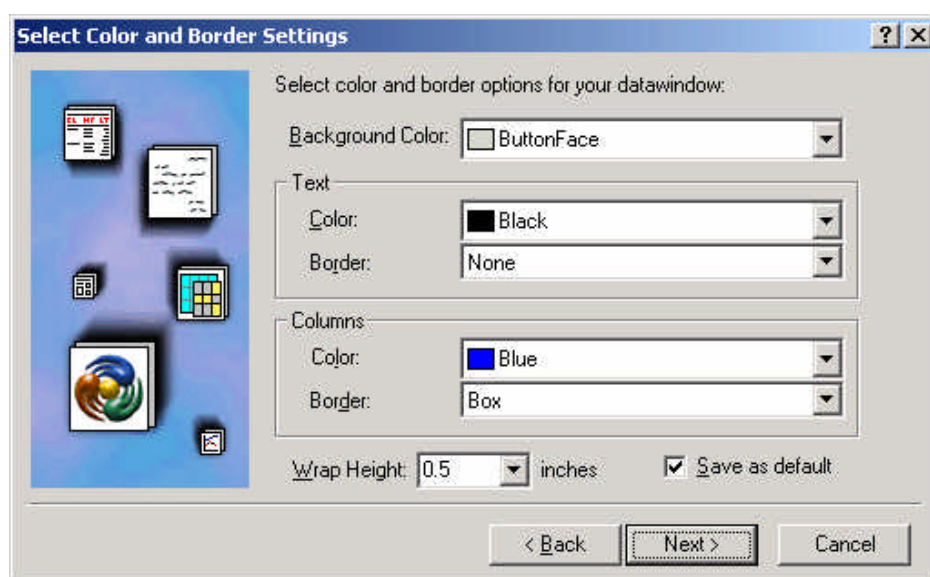
- Estilo: FreeForm
- Fuente de Datos: Quick Select
- Tabla: Curso
- Columnas: Todas

Los pasos que debe seguir son:

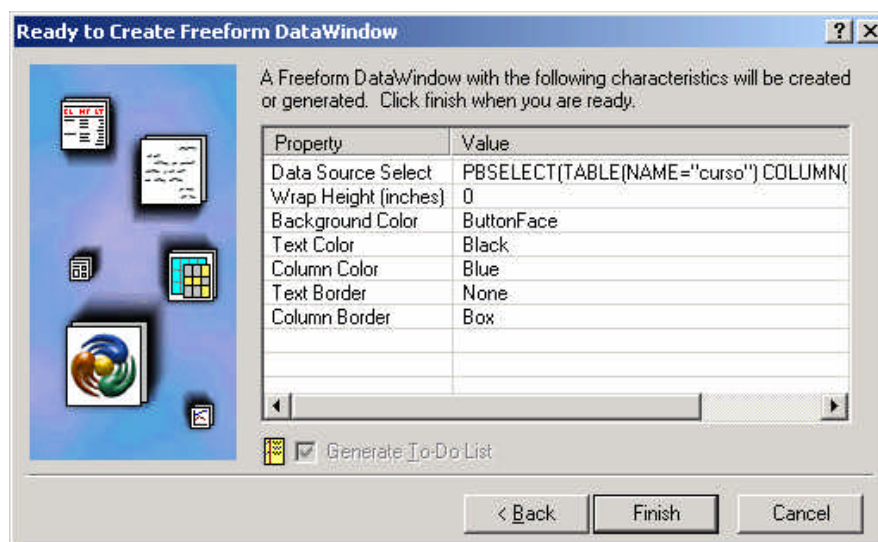
1. En la ficha **DataWindow** del diálogo **New**, ilumine el icono **FreeForm** y luego haga clic en el botón **Ok**.
2. En el diálogo **Choose Data Source for Freeform DataWindow**, ilumine el icono **Quick Select** y luego haga clic en el botón **Next**.
3. En el diálogo **Quick Select**, seleccione la tabla **Curso** y todas sus columnas, y luego haga clic en el botón **Ok**.



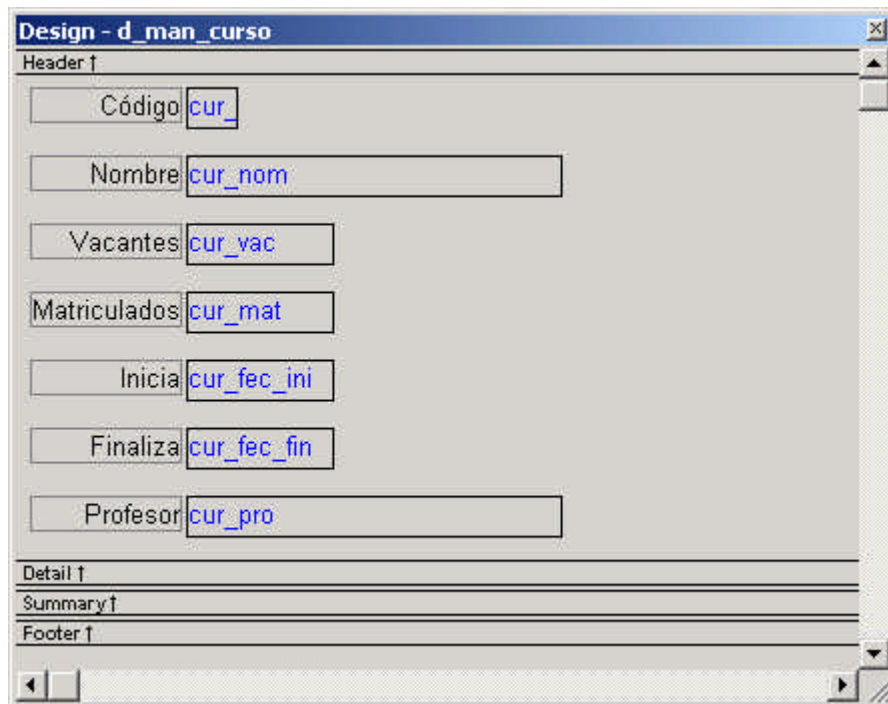
- En el diálogo **Select Color and Border Settings**, seleccione las propiedades según su criterio ó de lo contrario asuma los valores por defecto, luego haga clic en el botón Next.



- En el diálogo **Ready to Create FreeForm DataWindow**, haga clic en el botón **Finish**.



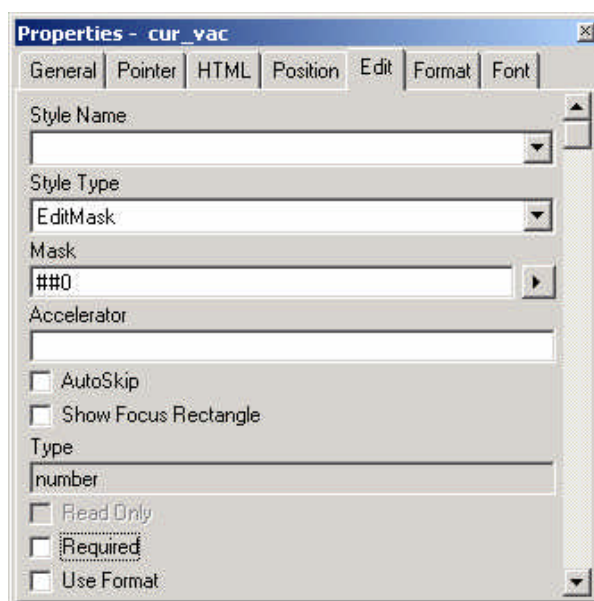
6. Ahora que ya tenemos el Objeto DataWindow, debe proceder a grabarlo.



Estilos de Edición de las columnas

Tenemos varios estilos de edición para las columnas, estos estilos permiten uniformizar los datos que ingresan los usuarios, así como también la selección de valor correctos provenientes de una lista, que bien podría ser creada a partir de los valores de una tabla, como por ejemplo la selección del código de un producto, la selección del código de un curso, etc.

La selección de un estilo de edición se realiza en ficha Edit del panel de propiedades, tal como se muestra en el siguiente grafico.



Ejercicio 02

Según el siguiente cuadro establezca las propiedades de edición de las columnas.

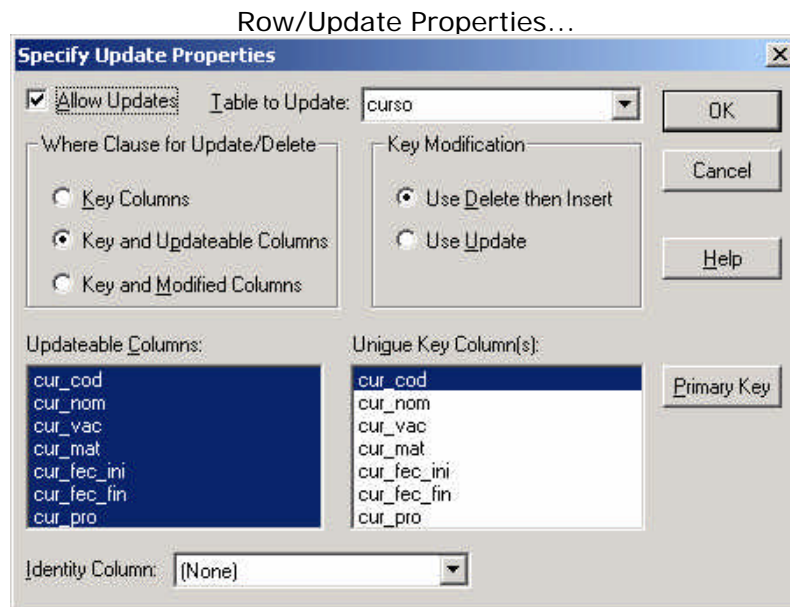
| Columna | Propiedad | Valor |
|-------------|--|--------------------------------------|
| Cur_cod | Style Type Case | Edit Upper (1) |
| Cur_nom | Style Type Case Alignment | Edit Upper (1) Left (0) |
| Cur_vac | Style Type Mask Spin Control Spin Increment Spin Min Spin Max | EditMask ## Sí 1 1 40 |
| Cur_mat | Style Type Mask Spin Control Spin Increment Spin Min Spin Max | EditMask #0 Sí 1 0 40 |
| Cur_fec_ini | Style Type Mask Spin Control Spin Increment | EditMask dd/mm/yyyy Sí 1 |
| Cur_fec_fin | Style Type Mask Spin Control Spin Increment | EditMask dd/mm/yyyy Sí 1 |
| Cur_pro | Style Type Case Alignment | Edit Upper (1) Left (0) |

Propiedades de Actualización de un DataWindow

Por defecto el objeto DataWindow solo puede actualizar una tabla. Para indicar las propiedades de actualización ejecute el comando **Update Properties** del menú **Row**, y tendremos acceso al diálogo **Specify Update Properties**, es en este diálogo donde indicaremos las características de actualización que tendrá el DataWindow.

Ejercicio 03

Establezca las propiedades de actualización de la tabla Curso como se muestra en el siguiente gráfico.



Acciones Predeterminadas para los botones de un DataWindow

Dentro del objeto DataWindow podemos insertar una serie de controles. La introducción de algunos estos controles se debe principalmente a la posibilidad de ejecución de un DataWindow de manera independiente en formato Plung-In dentro de un Visualizador de Páginas HTML. Su funcionalidad se puede emplear también en un entorno de aplicaciones Cliente/Servidor.



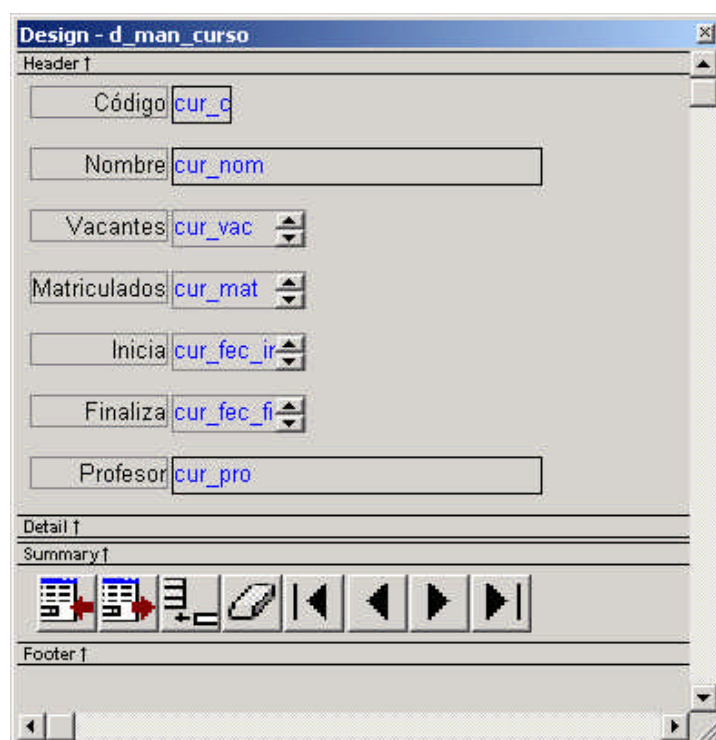
En esta oportunidad nos ocuparemos del control Button. Usted puede colocar este control en cualquier posición del DataWindow, y luego se puede acceder a el como si se tratara de cualquier columna.

El control Button permite la ejecución de una acción, esta puede una acción predeterminada ó definida por el usuario. Veamos las principales propiedades de este control.

| Propiedad | Descripción |
|-------------------------|---|
| Action | La acción que el usuario le asigna al botón. |
| DefaultPicture | Determina si se utiliza la imagen por defecto si se ha seleccionado una acción predeterminada. |
| Name | El nombre del botón. |
| PictureName | Indica la ubicación y el nombre del archivo que contiene la imagen que se mostrará en el botón. |
| SuppressEventProcessing | Determina si se disparan o no los eventos ButtonClicked y ButtonClicking para un botón en particular. |
| Text | El texto que muestra el botón. |

Ejercicio 04

En la banda Footer del DataWindow d_man_curso defina los siguientes botones:

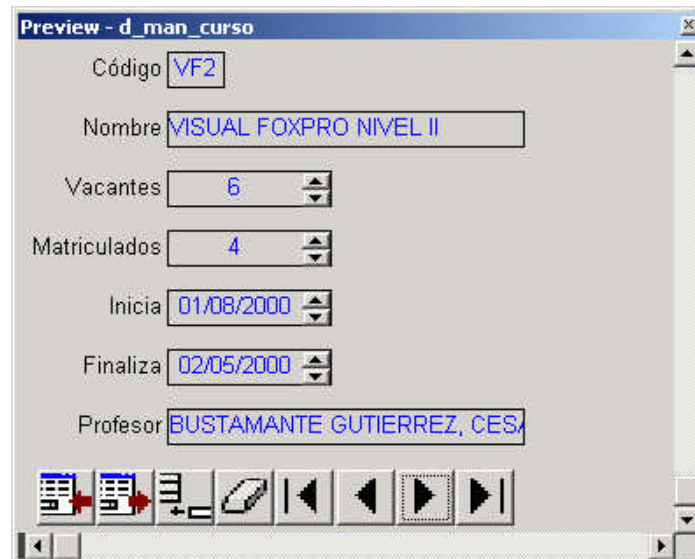


El siguiente cuadro le servirá de guía para establecer las propiedades de los botones, para una mejor comprensión se han enumerado los botones de izquierda a derecha. Los valores de las propiedades se deben establecer en la Ventana de Propiedades. El valor de la propiedad Name es el que toma por defecto.

| Número | Propiedad | Valor |
|--------|--|---------------------------------|
| 1 | Text Action Action Default Picture | <Blanco> Retrieve (2) Si |
| 2 | Text Action Action Default Picture | <Blanco> Update (13) Si |
| 3 | Text Action Action Default Picture | <Blanco> InsertRow(12) Si |
| 4 | Text Action Action Default Picture | <Blanco> DeleteRow(10) |
| 5 | Text Action Action Default Picture | <Blanco> PageFirst(6) Si |
| 6 | Text Action Action Default Picture | <Blanco> PagePrior(5) Si |

| Número | Propiedad | Valor |
|--------|--|-------------------------------|
| 7 | Text Action Action Default Picture | <Blanco> PageNext(4) Si |
| 8 | Text Action Action Default Picture | <Blanco> PageLast(7) Si |

Luego de grabar los cambios podemos realizar una vista preliminar.



Ejercicio 05

Creación de la ventana de mantenimiento de cursos, debe seguir los siguientes pasos:

1. Crear una nueva ventana, asígnele el nombre **w_man_curso** y un título adecuado.
2. Agregar un Control DataWindow y asígnele el nombre **dw_man_curso** y asócielo con el Objeto DataWindow **d_man_curso** (en el panel de propiedades).
3. codifique el siguiente script en el evento **open** de la ventana:

Script - Open for w_man_curso

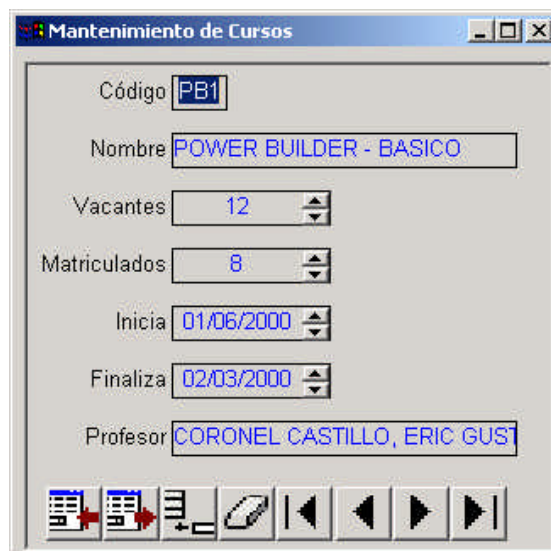
```
dw_man_curso.SetTransObject( SQLCA )  
dw_man_curso.Retrieve()
```

4. Grabar la ventana.
5. Programar el comando **Proceso/Curso** del menú marco.

Script - clicked for m_pro_curso

```
OpenSheet( w_man_curso,w_frame,4,Original! )
```

6. Ahora puede usted ejecutar la aplicación y realizar la prueba respectiva.



Funciones del Control DataWindow

Funciones de Desplazamiento

1. ScrollNextRow()
2. ScrollPriorRow()
3. ScrollToRow (long row)
4. ScrollNextPage()
5. ScrollPriorPage()

Funciones de Mantenimiento

6. SetTransObject (transaction transaction)
7. Retrieve ({ , any argument, any argument . . . })
8. InsertRow (long row)
9. DeleteRow (long row)
10. Update ({ boolean accept { , boolean resetflag } })
11. DeletedCount()
12. ModifiedCount ()

Otras Funciones

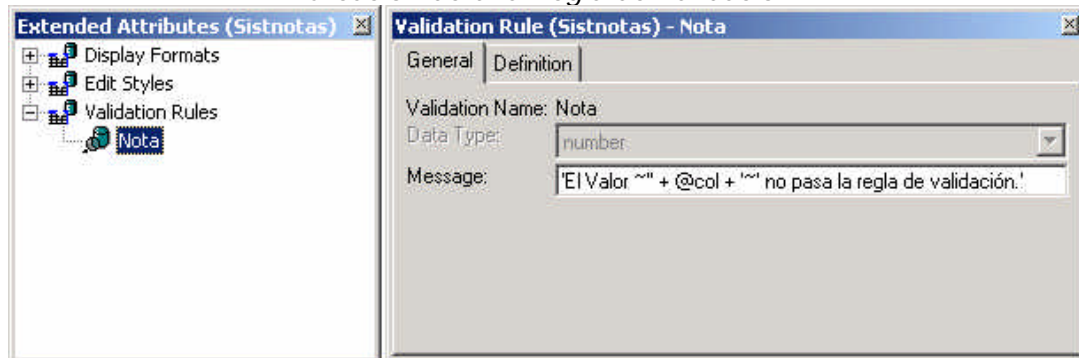
13. SetFocus()
14. GetRow()
15. GetColumn()
16. GetColumnName()
17. GetItemDate (long row, integer column { , DWBuffer dwbuffer, boolean originalvalue })
18. GetItemDate (long row, string column { , DWBuffer dwbuffer, boolean originalvalue })
19. GetItemDateTime (long row, integer column { , DWBuffer dwbuffer, boolean originalvalue })

20. GetItemDateTime (long row, string column { , DWBuffer dwbuffer, boolean originalvalue })
21. GetItemDecimal (long row, integer column { , DWBuffer dwbuffer, boolean originalvalue })
22. GetItemDecimal (long row, string column { , DWBuffer dwbuffer, boolean originalvalue })
23. GetItemNumber (long row, integer column { , DWBuffer dwbuffer, boolean originalvalue })
24. GetItemNumber (long row, string column { , DWBuffer dwbuffer, boolean originalvalue })
25. GetItemStatus (long row, integer column, DWBuffer dwbuffer)
26. GetItemStatus (long row, string column, DWBuffer dwbuffer)
27. GetItemString (long row, integer column { , DWBuffer dwbuffer, boolean originalvalue })
28. GetItemString (long row, string column { , DWBuffer dwbuffer, boolean originalvalue })
29. GetItemTime (long row, integer column { , DWBuffer dwbuffer, boolean originalvalue })
30. GetItemTime (long row, string column { , DWBuffer dwbuffer, boolean originalvalue })
31. SelectRow (long row, boolean select)
32. SetColumn(NúmeroColumna | NombreColumna)
33. SetRow (long row)
34. SetRowFocusIndicator (RowFocusInd focusindicator { , integer xlocation { , integer ylocation } })
35. Reset()
36. RowCount()

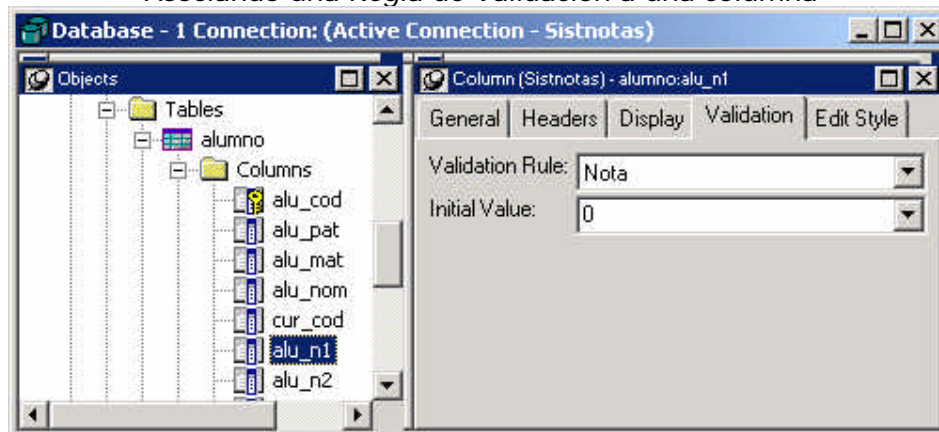
Validación de Datos

La introducción interactiva de datos en un DataWindow se realiza en una fila, columna a columna. El usuario introduce los valores apropiados para cada campo y es responsabilidad del desarrollador crear las reglas de validación necesarias para aceptar o rechazar el valor introducido. En el Database Painter es posible crear reglas de validación genéricas y luego asociarlas a varias columnas de la estructura de datos, como se ilustra en los siguientes gráficos..

Creación de una Regla de Validación



Asociando una Regla de Validación a una columna



Lo que veremos a continuación es cuando se aplica una regla y como se aceptan o rechazan los valores.

Cuando el usuario se encuentra con el cursor en un campo, ese campo tiene dos celdas: un de edición y otra de almacenamiento del dato dentro del buffer primario. Cuando el usuario entra en ese campo, los valores de ambas celdas coinciden. Si empieza a introducir datos en la celda, lo está haciendo en la celda de edición y es en ese momento cuando empieza a ser diferente su contenido de la celda del buffer.

La validación del dato introducido se realiza en cualquiera de los siguientes casos:

1. El usuario pulsa tecla **Enter**.
2. El usuario pulsa la tecla **Tab** y se mueve dentro de otra columna del objeto DataWindow.
3. El usuario pulsa con el botón del Mouse otra columna del objeto DataWindow.
4. Desde el código se ejecuta la función AcceptText().

Cada una de estas acciones provoca que se intente pasar el valor de la celda de edición a la celda del buffer primario. Durante este traspaso del valor de un sitio a otro se realiza la validación del valor de la celda de edición. Cada vez que se ejecuta una de las cuatro acciones anteriores y el usuario ha introducido un valor dentro de la celda de edición sucede esta secuencia de preguntas.

¿Es correcto el tipo de dato introducido dentro de la celda?

Esta pregunta la hace directamente PowerBuilder. El usuario refuerza la validación de tipos de datos, simplemente asignando, durante el proceso de creación de un DataWindow, una columna de base de datos a una columna del objeto. Si el tipo de dato no es el esperado, Sucede el evento **ItemError** del control DataWindow.

¿Satisface el valor la regla de validación asociada a la celda?

La regla de validación se puede asociar a la celda de dos maneras: desde el Painter DatWindow o desde el Painter Database mediante la asociación de un Atributo Extendido de tipo validación a la columna de una tabla. Si existe la regla, se evalúa el valor de acuerdo a la misma. Si no cumple, sucede el evento **ItemError**. Si la regla no existe, se salta este paso.

El evento **ItemError** nos ofrece la posibilidad de tratar el error. Existen valores de retorno asociados a este evento que nos brindan toda la gama de comportamientos que puede requerir el tratamiento de un error de edición.

Valores de retorno del evento ItemError

| Valor de Retorno | Comportamiento |
|------------------|--|
| 0 | Rechaza el valor introducido y presenta el mensaje de error. |
| 1 | Rechaza el valor y no presenta el mensaje. |
| 2 | Acepta el valor erróneo. |
| 3 | Rechaza el valor pero permite el cambio de la posición del cursor. |

¿Existe realmente un valor dentro de la celda de edición? ¿Ha cambiado el valor inicial?

El DataWindow es este punto comprueba el contenido de las dos celdas. En caso de que sean iguales, simplemente abandona el proceso de validación. Si no lo son se prosigue con la validación.

¿Existe código en el evento ItemChanges?

Este evento sucede dentro de un control DataWindow cada vez que el foco pasa de una celda a otra. Si las validaciones que se desean aplicar a la columna son lo suficientemente complicadas como para que las reglas de validación tradicional no sean suficientes para su implementación, se realiza la codificación en este evento. Esto sucede frecuentemente para validaciones cruzadas contra valores que no se encuentran en la misma fila del objeto DataWindow. Si el valor no pasa las validaciones, sucede el evento *ItemError*.

Con el evento **ItemChanged** sucede lo mismo que con el **ItemError**; nos permite modificar el comportamiento inicial en condiciones de error según los valores de la tabla siguiente.

Valores de retorno del evento ItemChanged

| Valor de Retorno | Comportamiento |
|------------------|--|
| 0 | Acepta el dato. |
| 1 | Rechaza el dato y provoca un evento ItemError. |
| 2 | Rechaza el valor pero permite el cambio de la posición del cursor. |

Las estrategias de validación no acaban aquí. Aparte de las reglas de validación definidas dentro del painter DataWindow, tenemos la posibilidad de definir Tablas de Código. Estas tablas de código nos permiten introducir un número pequeño de valores dentro de algunos estilos de edición como puede ser el CheckBox, los Botones de Radio y Listas Desplegables. Estos valores permiten que el usuario, en vez de introducir un dato, pueda seleccionar uno de los datos que se le ofrece, minimizando la posibilidad de error.

Si la tabla de códigos es a su vez una tabla de la base de datos se puede usar el estilo de edición DropDownDataWindow, que permite enlazar un objeto DataWindow con otro objeto DataWindow desplegable.

Ejercicio 06

Realice las modificaciones necesarias en la ventana **w_man_curso** para que tenga la apariencia que se muestra a continuación.

Mantenimiento de Cursos

Fila 1 de 12

Código: PB1

Nombre: POWER BUILDER - BASICO

Vacantes: 12

Matriculados: 8

Inicia: 01/06/2000

Finaliza: 02/03/2000

Profesor: CORONEL CASTILLO, ERIC GUST

Botones: Siguiete, Anterior, Recuperar, Añadir, Grabar, Cerrar

Ejercicio 07

Desarrolle una ventana para la matricula de nuevos alumnos.

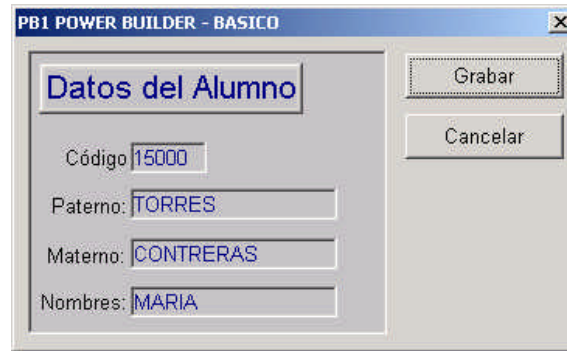
En el diálogo Seleccione Curso solo se debe mostrar los cursos que tienen vacantes, después de seleccionar el curso, debe hacer clic en el botón Matricular

Seleccione Curso

| Código | Nombre | Vacantes | Matriculados | Inicia |
|--------|--------------------------------|----------|--------------|------------|
| AU1 | AUTOCAD NIVEL I | 17 | 3 | 06/01/2000 |
| AU2 | AUTOCAD NIVEL II | 7 | 3 | 06/01/2000 |
| AU3 | AUTOCAD NIVEL III | 7 | 0 | 06/01/2000 |
| BC1 | BORLAND C++ NIVEL I | 20 | 3 | 06/01/2000 |
| BC2 | BORLAND C++ NIVEL II | 9 | 1 | 08/01/2000 |
| PB1 | POWER BUILDER - BASICO | 12 | 8 | 06/01/2000 |
| PB2 | POWER BUILDER - AVANZADO | 8 | 2 | 06/01/2000 |
| PB3 | POWER BUILDER - CLIENTE/SERVID | 8 | 2 | 06/01/2000 |

Botones: Matricular, Cancelar

Después de hacer clic en el botón Matricular aparece un diálogo modal donde se debe ingresar los datos del alumno.



Después de ingresar los datos del alumno, acepta la matricula haciendo clic en el botón Grabar, ó la cancela haciendo clic en el botón Cancelar.

Ejercicio 08

Desarrolle una ventana para el ingreso de notas.



Los ejemplos los puede bajar de www.LibrosDigitales.NET.

Próxima Entrega
Edición 1.1
Dentro de una semana

Prohibida la reproducción total o parcial
Sin nuestro consentimiento
Derechos de copia reservados
www.LibrosDigitales.NET