

Manual de Oracle Forms 6i

Red Software House - Publiguías



Autor: Francisco Espinoza R.
Fecha: Marzo 2003.

INDICE GENERAL

1-INTRODUCCION A FORMS	Pag 2.
2-FORMS	Pag 2.
3-ELEMENTOS	Pag 3.
3.1 OBJETOS Y PROPIEDADES	Pag 3.
3.2 TRIGGER	Pag 4.
4-BUILT IN	Pag 4.
5-ELEMENTOS DE DISEÑO	Pag 5.
5.1-NAVEGADOR	Pag 5.
5.2-BLOQUES DE DATOS	Pag 10.
5.3-TIPOS DE ÍTEMS MAS UTILIZADOS	Pag 15.
6-CANVAS Y VENTANAS	Pag 20.
6.1-CANVAS DE CONTENIDO	Pag 21.
6.2-TRABAJAR CON VARIOS CANVAS DE CONTENIDO	Pag 24.
6.3-CANVAS APILADOS	Pag 28.
6.4-BARRAS DE HERRAMIENTAS	Pag 32.
7-BUILT-IN PARA CONEXION A LA BBDD	Pag 33.
8-VARIABLES DEL SISTEMA	Pag 35.
9-FUNCIONALIDAD	Pag 37.
9.1-CHECK BOX	Pag 37.
9.2-RADIO BUTTON	Pag 39.
9.3-LIST ITEM	Pag 42.
9.4-LOVS Y RECORD GROUPS	Pag 49.
9.5-ALERTAS	Pag 57.
9.6-TIMERS	Pag 60.
9.7-EDITORES	Pag 64.
10-COMUNICACION ENTRE DIVERSOS Elementos	Pag 65.
10.1-PARAMETROS_1	Pag 65.
10.2-RELACION ENTRE BLOQUES	Pag 68.
10.3-COMUNICACION ENTRE FORMS	Pag 72.
10.3.1-OPEN_FORM	Pag 72.
10.3.2-NEW_FORM	Pag 73.
10.3.3-CALL_FORM	Pag 74.
10.3.4-RUN_PRODUCT	Pag 75.
10.4-PARAMETROS_2	Pag 76.
11-CREACION DE MENUS	Pag 80.
12-ANEXOS	Pag 84.
12.1-IDENTIFICADOR INTERNO	Pag 84.
12.2-VALIDACIONES	Pag 85.

DEVELOPER 2000/FORMS

1-INTRODUCCION A FORMS

Developer 2000 es una herramienta integrada para el desarrollo de aplicaciones visuales que precisen acceder a bases de datos desde plataformas Windows (aunque se están desarrollando también nuevas versiones para otros entornos como Linux).

Su funcionamiento esta optimizado para bases de datos relacionales Oracle pero también puede utilizarse con otros gestores como Acces y Sql Server de Microsoft o DB2 de IBM y otros mediante el uso de controladores ODBC, su funcionamiento en estos otros gestores es mas lento e inestable.

Developer 2000 esta compuesto de cuatro aplicaciones básicas, a saber:

- FORMS. Genera pantallas y trata dichas pantallas.
- REPORTS. Creacion de informes sobre una tabla.
- GRAPHICS. Genera gráficos estadísticos partiendo de los valores contenidos en una tabla.
- BOOK. Utilidad para generar documentación sobre las aplicaciones.

2-FORMS

Herramienta cliente-servidor que permite modificar, crear y borrar datos en la BBDD mediante la interacción del lenguaje PL/SQL y la programación visual orientada a eventos.

Forms se compone de tres programas principales:

- Forms Designer (a partir de Forms 5.0 se denomina Forms Builder): crea las pantallas y da funcionalidad a las mismas.
- Forms Generate: convierte el fichero designer en un fichero seudoejecutable
- Forms Runform: ejecuta el seudoejecutable

Es importante saber que Forms no genera ficheros *.exe.

A pesar de ser ficheros independientes y de poder generarse por separado, nos es más cómodo realizar las tres fases desde Forms Designer, dado que este nos lo permite.

Designer genera tres tipos de ficheros principales sin los que una aplicación nunca podrá existir en Forms:

- *.fmb: incluye todas las partes que intervienen en el desarrollo, pantallas, procedimientos, funcionalidad, etc.

- *.fmt: contiene el fmb traducido a lenguaje C.

- *.fmx: seudoejecutable

Otro tipo de ficheros que también genera Forms y que veremos mas adelante en detalle son los menús y las librerías:

Menús:

- *.mmb: fichero fuente visual de un menú

- *.mmt: fichero fuente en modo texto del menú

- *.mmx: fichero seudoejecutable del menú

Librerías:

- *.pll: código fuente

- *.pld: código fuente en modo texto

- *.lib: ejecutable

3-ELEMENTOS

3.1 OBJETOS Y PROPIEDADES:

Objeto es todo aquello que se pueda insertar en una pantalla y tenga propiedades: cuadros de texto, botones, etc.

Cuando se genera un objeto automáticamente se crean sus propiedades que tomaran un valor por defecto inicial dado por Forms, dichas propiedades se podrán modificar en tiempo de diseño o en tiempo de ejecución dependiendo del objeto y de la propiedad, y lo que es más importante dichas propiedades se comportaran como variables de tipo global en cuanto a su contenido, es decir si son modificadas mantendrán el valor asignado a lo largo del tiempo de ejecución o hasta que su valor sea nuevamente alterado.

Los objetos se dividen en:

- a) Items: interfaz de comunicación entre el usuario y la aplicación, textos, botones, listas, etc.
- b) Bloques: agrupaciones de ítems. Estos bloques pueden estar asociados a tablas o no, es mas todo ítem debe estar asociado a un bloque de tal modo que será necesario crear al menos un bloque por si solo existe un ítem en la aplicación.

3.2 TRIGGER

Permiten dar funcionalidad a los eventos (instrucciones asociadas a un ítem cuando se produce un evento). No deben confundirse con los eventos de forma aislada, un trigger es un conjunto formado por un evento y unas instrucciones, es un concepto algo distinto del simple concepto de evento que existe en otros entornos de desarrollo visuales, debe ser tratado de una forma más conceptual que practica, su dominio requiere un poco de experiencia. No obstante podemos encontrar referencias a un Trigger solamente como disparador o evento.

Los triggers principales en Forms son:

- key: apenas utilizado en Forms, son ejecutados al pulsar una tecla o combinación de teclas, principalmente se usan en entornos de modo texto.
- when: él más utilizado, son desencadenados por múltiples eventos
- on: interviene en acciones por defecto del sistema, ON INSERT, ON DELETE,..
- pre: se disparan antes de ejecutar el evento correspondiente. PRE INSERT, por ejemplo para comprobar condiciones específicas de inserción en función de unos valores.
- post: misma funcionalidad que el anterior pero a la inversa temporalmente POST INSERT, POST COMMIT, etc..

4-BUILT IN

Son funciones y procedimientos empaquetados dentro de Developer 2000, son proporcionados por Oracle en forma binaria y por tanto no se dispone de su código fuente. Los Built-in se encargan de realizar tareas que faciliten el uso de las herramientas de Forms, fundamentalmente de los ítems, para así descargar al

programador de trabajo, algunos son muy complejos y potentes por lo que supone quitar de los programas bastantes miles de líneas de código que de otra manera deberíamos escribir nosotros. Su número dependiendo de la versión de Forms es de unos cientocincuenta. Muchos triggers adquieren su potencia debido a estos Built-in.

5-ELEMENTOS DE DISEÑO

5.1-NAVEGADOR:

En el visualizaremos tanto los objetos efectivamente instanciados o incluidos en nuestra aplicación como los que tenemos a nuestra disposición para ser incluidos (lo que podríamos denominar para entendernos; clases)

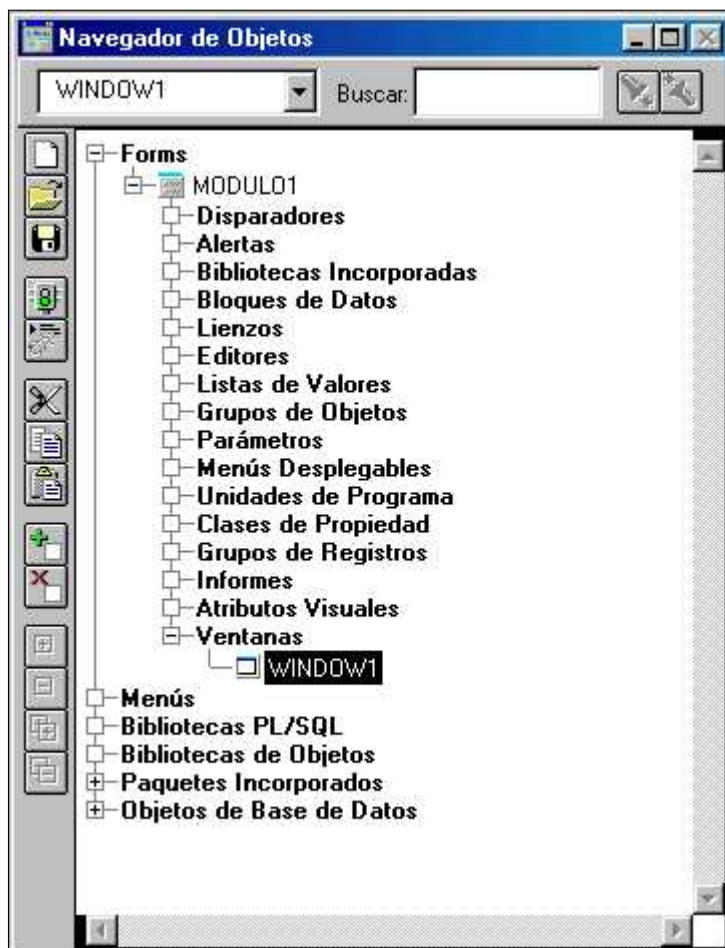


Imagen 1.

Como se puede apreciar en la Imagen 1 el navegador despliega los componentes que intervienen en toda aplicación desarrollada con Forms, estos son los Módulos, Menús, Bibliotecas, Paquetes Incorporados y Objetos de la Base de Datos

- Modulo: cada modulo puede contener una aplicación autosuficiente pero podremos trabajar con aplicaciones que incluyan uno o mas módulos
- Menús: menús creados por el programador.
- Bibliotecas: funciones y procedimientos creados por el programador.
- Paquetes incorporados: funciones y procedimientos almacenados en la BBDD accesibles en todo momento para el programador, por ejemplo el paquete DBMS_OUTPUT tiene una serie de funciones asociadas, entre ellas PUTLINE, que permite imprimir datos por pantalla. Este componente incluye todas las funciones y controles integrados por Oracle en Forms.
- Objetos de la base de datos: aquí encontramos una imagen de las tablas y vistas a las que tenemos acceso, además de otros objetos de la base de datos a la que estemos accediendo.

Pasamos a describir el contenido de el componente MODULO1 (nombre asignado al modulo por defecto al entrar en un Forms “vacío”, lo habitual es que los programadores asignen al modulo el nombre del fichero fmb que va a contener la aplicación.)

Como se comenta en la descripción del modulo, cada modulo puede contener una aplicación autosuficiente pero podremos trabajar con aplicaciones que incluyan uno o mas módulos, cuantos mas módulos incluyamos mas difícil será trabajar con ellos dado que el numero de objetos en el navegador aumenta rápidamente.

- Disparadores: aquí se encuentran todos los eventos posibles en una aplicación Forms, incluidos eventos propios de Forms y eventos Windows (como la carga de un form en memoria, la pulsación de teclas o el movimiento del ratón), si quisiésemos acceder de forma más compleja al API de Windows deberíamos utilizar otras librerías y funciones cuya explicación esta fuera del propósito de esta guía , en cualquier caso el programador puede acudir a la extensa documentación que existe sobre el tema en diversas publicaciones.
- Alertas: avisos predefinidos por la aplicación como mensajes de error o advertencias.

- Bibliotecas incorporadas: funciones y procedimientos locales al modulo y por tanto solo accesibles por este.
- Bloques de datos: bloques lógicos de ítems
- Vistas lienzo (canvas): todas las “paginas” de la aplicación.
- Editores: sirven para poder escribir comentarios en una ítem de la aplicación.
- Listas de valores: listas que permitirán la elección de un campo dentro de las mismas.
- Grupos de objetos: nos permitirán heredar propiedades, agregar nuevas, tener objetos hijos, etc.
- Parámetros: valores pasados a la aplicación con algún fin, por ejemplo en función de un departamento realizar un informe u otro.
- Unidades de programa, funciones y procedimientos creados por el usuario
- Clases de propiedad: conjunto de propiedades para crear objetos idénticos (varios botones iguales por ejemplo)
- Atributos visuales: es idéntico a clases de propiedad pero con atributos gráficos
- Ventanas: ventanas del modulo
- Grupo de registros: cursores internos creados por Forms



*Podemos establecer una relación entre los diversos componentes de una aplicación.



- 1 aplicación tiene 1 o varios Módulos
- 1 Modulo tiene 1 o varias Ventanas
- 1 Ventana tiene 1 o varios Lienzos o Canvas
- 1 Canvas tiene uno o varios Bloques
- 1 Bloque tiene uno o varios Items

Es importante saber que los objetos visuales se distinguirán de las variables del código PL/SQL por un prefijo que será el símbolo gramatical ‘:’ de tal modo que para referenciar a un ítem en código PL/SQL se antepondrá dicho símbolo. El lector habrá advertido que los objetos visuales son tratados como una variable mas en el código ya que en el fondo dichos objetos no son mas que eso, variables desde el punto de vista del programador de Forms.

Las variables PL/SQL son locales al bloque en el que se declaran y para declarar una variable global antepondremos el prefijo ‘:GLOBAL.NOMBRE_VARIABLE’, las variables globales se declaran por sintaxis exclusivamente y no por ubicación y sintaxis como si se puede hacer en lenguaje C.

Vamos a crear un simple ejemplo en el que aplicaremos gran parte de lo visto hasta este momento, como es habitual utilizaremos el famoso “Hola mundo”.

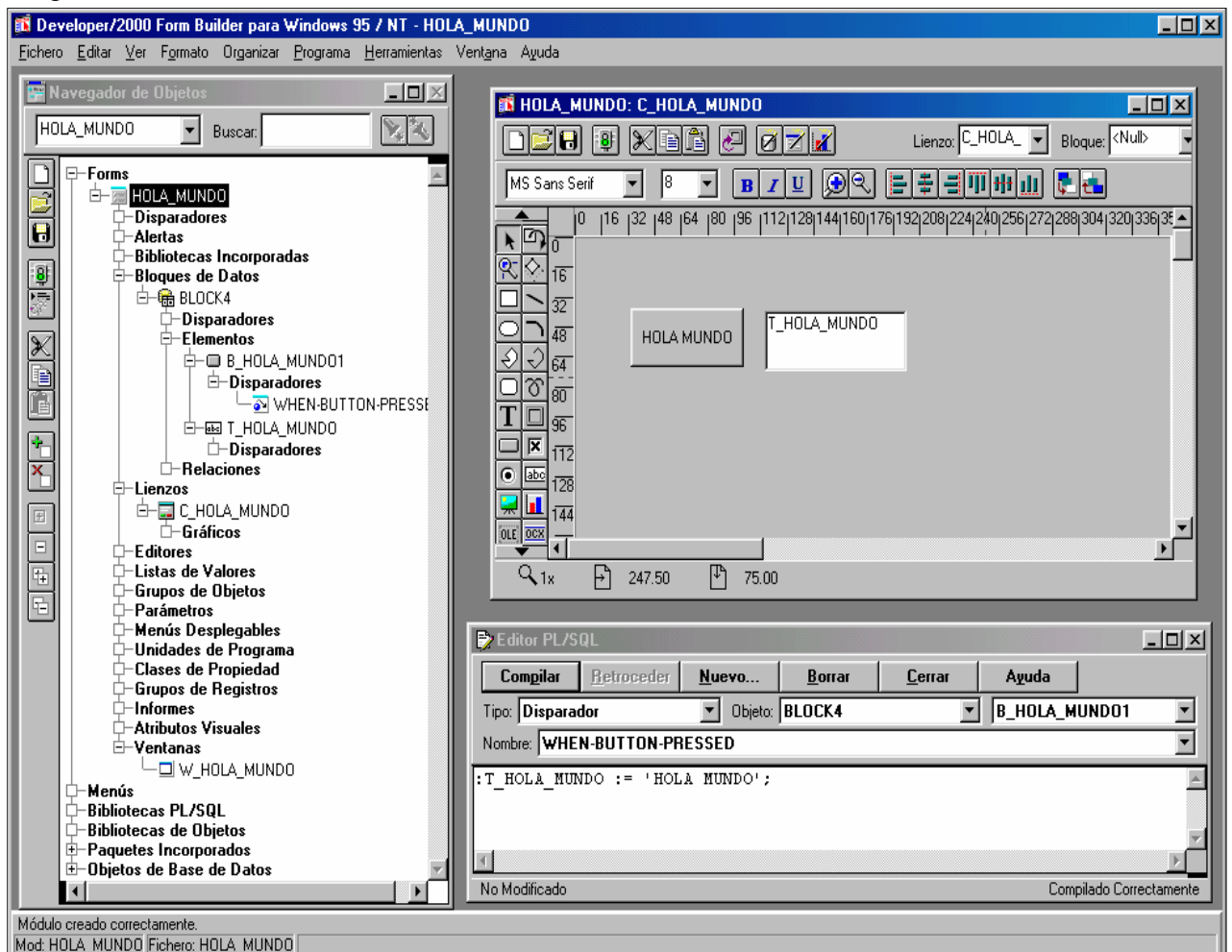
- 1-En primer lugar en vista lienzo crearemos un nuevo formulario o canvas, para ello pulsaremos sobre el elemento Lienzo del modulo HOLA_MUNDO y después sobre el botón  que permite crear o eliminar elementos usaremos + para crear y X para eliminar,  después pulse dos veces sobre el para visualizarlo.

- 2-En dicho canvas insertaremos un botón  y un Text Item , a los que cambiaremos nombre y etiquetas accediendo a sus propiedades con el menu contextual al pulsar el botón derecho del ratón sobre ellos.

- 3-Con el botón derecho del ratón pulsaremos sobre el botón desplegando así de nuevo el menú contextual e invocaremos el editor de PL/SQL, pulse después Nuevo para escoger el trigger si no aparece la ventana de triggers.

- 4-Escribiremos asociando al evento WHEN_BUTTON_PRESSED, la siguiente línea de código :T_HOLA_MUNDO := ‘HOLA MUNDO’

Imagen 2.



70

Una vez hecho esto debemos compilar el código escrito utilizaremos el botón creado para ello en el editor PL/SQL, si no se producen errores podremos pasar a ejecutarlo, utilizando para ello el botón que tiene dibujado en su interior un semáforo


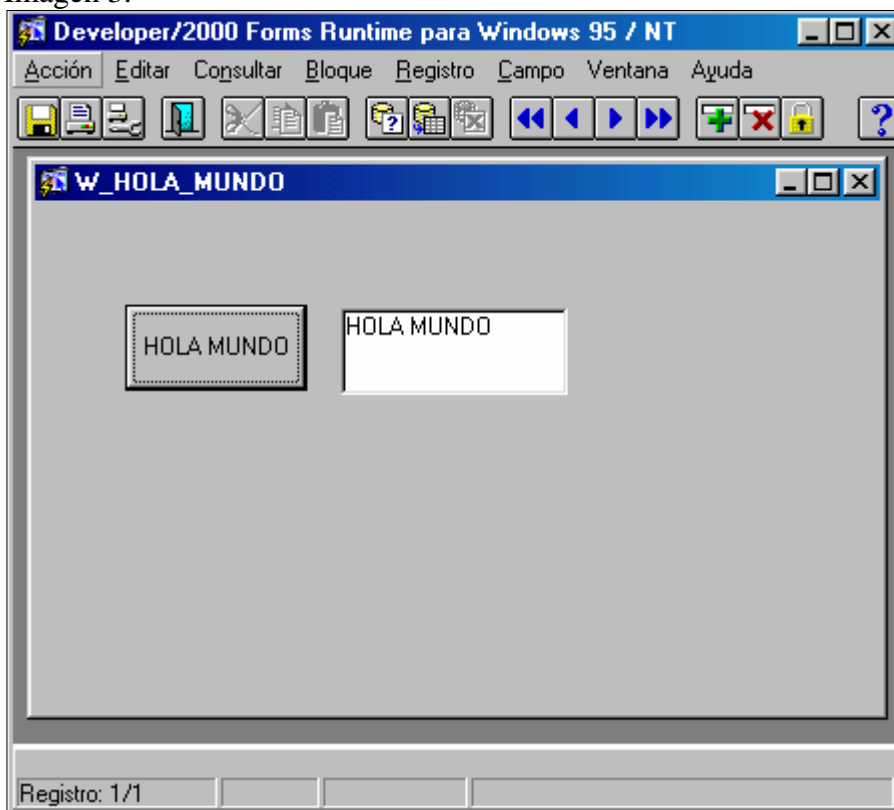
, bien desde el navegador de objetos o desde el editor de diseño. Dicha compilación generara un fichero HOLA_MUNDO.fmx que estará alojado en el mismo directorio que el fichero HOLA_MUNDO.fmb y generara el programa delante nuestro en mas o menos tiempo dependiendo del tamaño del mismo y de los accesos que deba hacer a la BBDD, el aspecto que presentara el programa será similar a la Imagen 3 dependiendo de la versión de FORMS que utilicemos.

Imagen 3.



Como podemos comprobar, ,en este sencillo ejemplo hemos aplicado mucho de lo que hemos visto, repasemos lo que hemos hecho hasta el momento.

En primer lugar creamos un modulo al que llamamos HOLA_MUNDO a partir de aquí es de donde surge toda la aplicación, al generar después un CANVAS o LIENZO hemos preparado el entorno de trabajo en el que hemos ido insertando controles y código asociado a esos controles, y algo que es mas importante hemos generado un Bloque de Datos sin que nos hayamos dado cuenta ya que Forms lo hizo por nosotros, si examinamos la Imagen 2 veremos que se ha creado un elemento llamado BLOCK4 (el cual no he cambiado de nombre para que el lector pueda ver que

es generado automáticamente) y que este incluye todos los controles insertados dentro de la ventana final y que dichos controles llevan asociados o no determinados eventos, esto resulta de vital importancia en Forms pues la aplicación comenzara a ejecutarse a partir del primer bloque de no modificar nosotros esta disposición por defecto de Forms, además como veremos mas adelante podremos utilizar bloques con controles que harán referencia a columnas tablas de la BBDD, lo que en definitiva es la misión de Forms.

Otro punto importante a destacar es la relación entre el Lienzo y la Ventana, no debemos confundir ambas cosas pues son muy distintas, habrá observado que el resultado de la ejecución del modulo no es “normal” comparado con otras aplicaciones Windows, ya que contiene una ventana principal y en su interior otra ventana, pues bien el Lienzo C_HOLA_MUNDO es la ventana principal y la ventana interior es W_HOLA_MUNDO, por tanto solo podrá haber un lienzo principal en ejecución y una o varias ventanas en su interior pero nunca dos canvas podrán aparecer como ventanas principales a la vez.

Este pequeño ejemplo solo es una introducción a Forms iremos profundizando mas en cada uno de los elementos de una aplicación.

5.2-BLOQUES DE DATOS:

Los bloques de datos son extremadamente importantes en Forms, analicemos ahora sus características detalladamente.

Un bloque de datos es un conjunto de ítems y puede ser de dos tipos:

- a) Bloque de control: contendrá ítems no relacionados con la BBDD
- b) Bloque tabla: contendrá ítems si relacionados con la BBDD


Dependiendo de la versión que utilice de Forms los pasos para crear un bloque bien de tabla o de control pueden variar, pero siempre comenzaran del mismo modo, pulsando en el navegador sobre Bloque de Datos y a continuación sobre el botón . , Ahora deberá aparecer una ventana que le indicara si quiere crear un bloque tabla o un bloque de control, si crea el bloque de control simplemente vera como aparece un nuevo bloque en el navegador con sus elementos por defecto, si escoge un bloque tabla deberá tomar decisiones, vamos a detenernos en ellas pues son sumamente importantes para Forms.

Imagen 4.

The screenshot shows the 'M_BLOQUES: New Block Options' dialog box with the 'General' tab selected. The fields are as follows:

Field	Value
Base Table:	EMPLEADO
Block Name:	EMPLEADO
Canvas:	C_BLOQUES
Sequence ID:	1

Buttons: Select... (next to Base Table and Canvas), OK, Cancel.

Como puede ver en la Imagen 4 hemos seleccionado la tabla EMPLEADO , a continuación seleccionamos los campos que nos interesa que formen parte del bloque, serán todos aquellos anteceditos del signo +.

Imagen 5.

The screenshot shows the 'M_BLOQUES: New Block Options' dialog box with the 'Items' tab selected. The 'Select Columns...' button is visible. The list of columns includes:

- + COD_EMPLEADO
- + NOMBRE
- + APELLIDO1
- + APELLIDO2
- + DIRECCION
- + NIF
- + TLF_DOM
- + TLF_CONTACTO

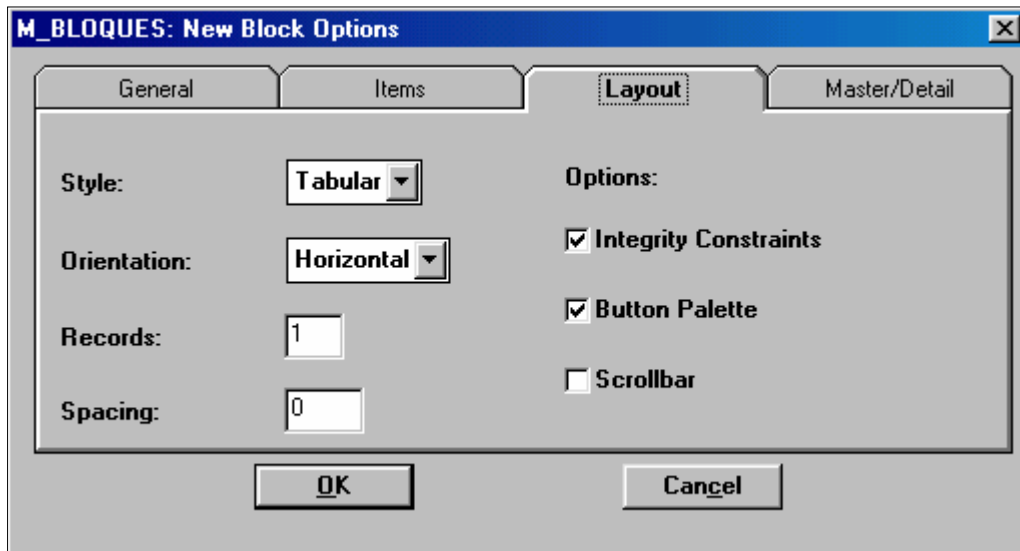
The 'Item Options' section shows:

Field	Value
Label:	Cod Empleado
Width:	54
Type:	Text Item
Include:	<input checked="" type="checkbox"/>

Buttons: OK, Cancel.

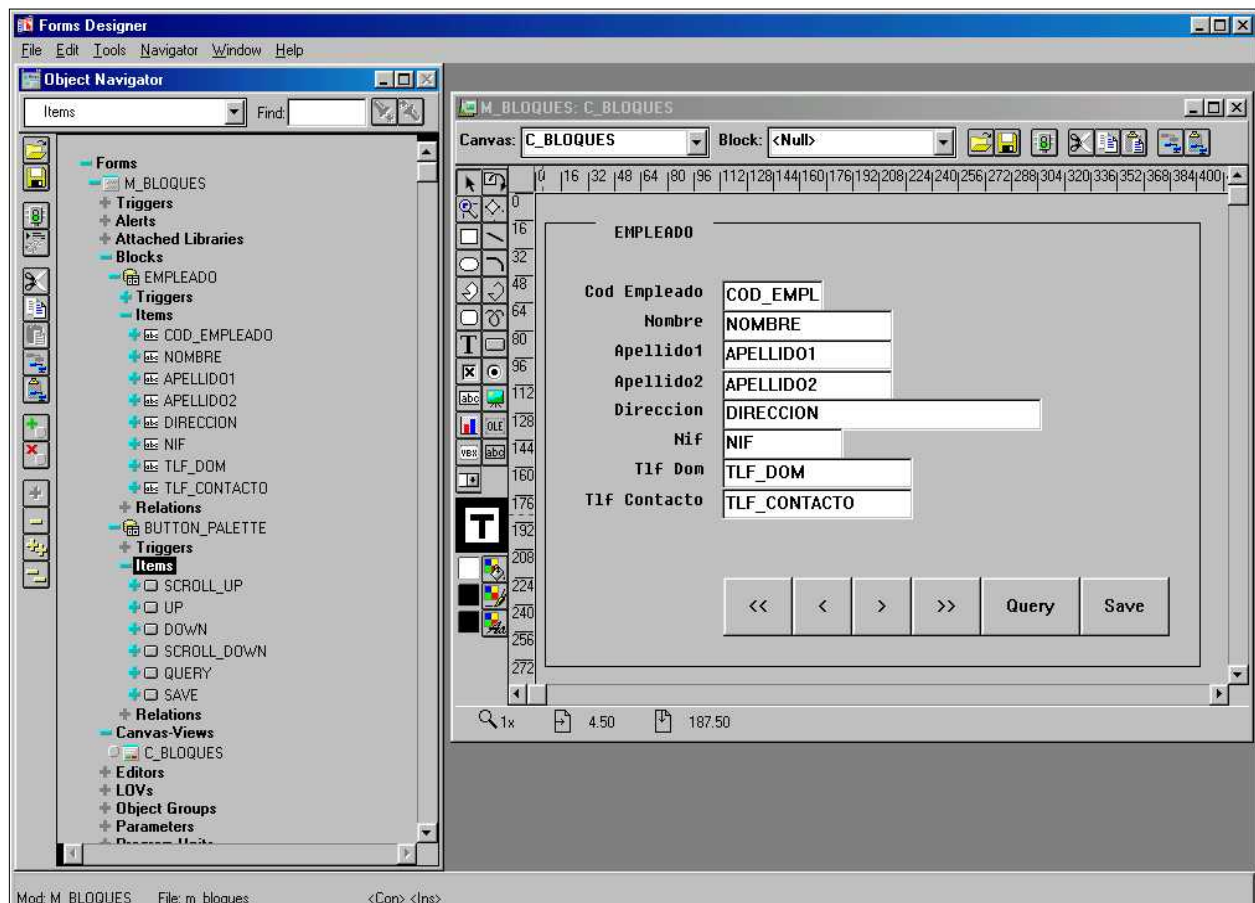
Observe en la Imagen 5 que puede modificar la etiqueta que aparecerá identificando al campo y lo que es mas importante puede ver que dichos campos aparecerán dentro de un Text Item, podríamos haber seleccionado otro tipo de control para que por ejemplo el usuario no pudiese escribir en su interior, dado que los Text Item admiten la modificación de su contenido.

Imagen 6.



Por ultimo debemos establecer la apariencia del bloque en el canvas, si queremos que se respete la integridad referencial y si deseamos un cuadro de botones o una barra de desplazamiento, pulsaremos aceptar y el bloque tabla será incluido en el canvas y por supuesto en el navegador, donde nos detendremos a analizar diversos temas. El apartado maestro-detalle lo veremos mas adelante.

Imagen 7.



Como podemos ver en la Imagen 7 Forms ha generado dos bloques dentro del modulo M_BLOQUES, uno de ellos se llama EMPLEADO y es el que contiene los

TEXT_ITEM correspondientes a los campos asociados con la BBDD y el otro es el denominado BUTTON_PALETTE en el que se han incluido todos los botones generados por Forms para el avance entre los registros y el “salvado” de registros nuevos que incluyamos, es muy importante que entendamos porque Forms ha separado ambos tipos de controles en dos bloques, intentare explicarlo de forma mas detallada, existen dos razones principales:

- a) Todo bloque tabla debe contener única y exclusivamente aquellos controles relacionados directamente con la BBDD, de lo contrario Forms producirá un error, observe el lector como Forms ha separado todos los controles creando para ellos dos bloques; EMPLEADO y BUTTON_PALETTE, esto es debido a que Forms intentara incluir datos para su visualización dentro de todos los controles de un bloque tabla, cuando no pudiese realizarlo por ejemplo en un botón por razones obvias la aplicación produciría un error en tiempo de ejecución, una ubicación de controles en Forms como la de la Imagen 8 dentro de un bloque tabla produciría un error .

Imagen 8.



- b) En segundo lugar es de vital importancia que sepamos que cuando la aplicación se cargue en memoria para su ejecución Forms utilizara un procedimiento descendente para visualizar los canvas, de tal modo que el primero que se podrá ver será aquel que contenga el primer ítem que aparezca en el primer bloque, en este caso al no haber mas canvas que C_BLOQUES y dado que el primer bloque es EMPLEADO y el primer control dentro de este es COD_EMPLEADO que esta incluido dentro de la ventana del canvas C_BLOQUES es lógico pensar que será este canvas el primero en visulizarse, por tanto en FORMS no existe un procedimiento especifico para indicar cual queremos que sea el primer formulario de la aplicación, este se lanzara en función de su ubicación en el navegador (aunque esto es modificable como veremos mas adelante), este es un aspecto criticable de Forms dado que no es obviamente la forma mas adecuada de preparar la aparición de formularios, si nuestra aplicación contase con varios canvas deberíamos prestar especial atención a la ubicación de los controles contenidos en los mismos dentro del navegador de objetos. Generalmente la

norma mas adecuada es separar los bloques de controles en base a su funcionalidad y objetivo dentro de una aplicación aunque esto variará en función de las necesidades del programador.

Por tanto es importantísimo saber donde debemos ubicar nuestros controles dentro del navegador de objetos, a medida que avancemos en Forms iremos descubriendo mas detalles a este respecto, pasemos a ver el resultado de la ejecución de nuestro ejemplo anterior.

The screenshot shows a Windows-style application window titled "Forms 4.5 (Runform)". Inside, there is a sub-window titled "W_BLOQUE". The form is titled "EMPLEADO" and contains several data entry fields with labels to their left:

- Cod Empleado: 1
- Nombre: Eduardo
- Apellido1: Garcia
- Apellido2: Martinez
- Direccion: Juan Alvarez Mendizabal 15
- Nif: 50164448k
- Tlf Dom: 915425889
- Tlf Contacto: (empty field)

Below the fields is a row of six buttons: "<<", "<", ">", ">>", "Query", and "Save". At the bottom left of the window, it says "Count: *3" followed by a small 'v' icon.

Imagen 9.

La salida de la aplicación por pantalla tal y como puede verse en la imagen inmediatamente superior incluye los dos grupos de controles que acabamos de analizar, aquellos que se relacionan directamente con el contenido de columnas de la BBDD y el otro bloque que contenía aquellos controles que no tenían esa relación directa, los cuales en este caso son botones de avance y retroceso y de inserción, Forms ya ha incluido código en estos botones por nosotros y ademas ha colocado unas etiquetas descriptivas a cada uno de los campos.

Otro punto interesante en este momento es observar la disposición de las ventanas, observe como la ventana interior principal contiene el canvas en el que se encuentran los controles (explicaremos en detalle en siguientes capítulos la diferencia entre canvas y ventana). Es fácil olvidar que en el explorador de objetos al final de la lista encontramos dicha ventana a la que hemos llamado W_BLOQUE, si quisiésemos alterar las dimensiones o el aspecto de la ventana interior (dado que la exterior es algo ajeno a la aplicación) deberíamos acceder a ellas pulsando el botón derecho del ratón sobre ella en el navegador y accediendo a sus propiedades o bien haciendo una doble pulsación del botón izquierdo del ratón sobre dicho objeto igualmente en el navegador, trataremos con mas profundidad la posibilidad de que existan varias ventanas e incluso varios canvas en un mismo modulo y su navegación entre ellos en capítulos posteriores.

Una ultima consideración al respecto de este programa, el lector pudiera pensar que la aplicación esta prácticamente terminada dado que presenta un buen aspecto, permite operar sobre la BBDD y nos indica perfectamente sobre que campos estamos trabajando y lo que es mas, no hemos escrito ni una sola línea de código, nada mas lejos de la realidad, si nos paramos a observar ninguno de los campos esta validado con lo que el usuario final podría introducir cualquier valor en ellos, además no nos es posible retroceder ante una posible equivocación. Los bloques tabla son un punto de partida para realizar aplicaciones mucho mas complejas y eficaces y han sido tratados en este capitulo para entender mejor la ubicación de controles dentro del navegador de objetos.

5.3-TIPOS DE ITEMS MAS UTILIZADOS:

Antes de analizar los ítems mas utilizados es importante estudiar el acceso a las propiedades de un ítem sea cual fuere.

En la parte izquierda del editor de diseño encontramos la lista de ítems a nuestra disposición para incluir en nuestros formularios o canvas.

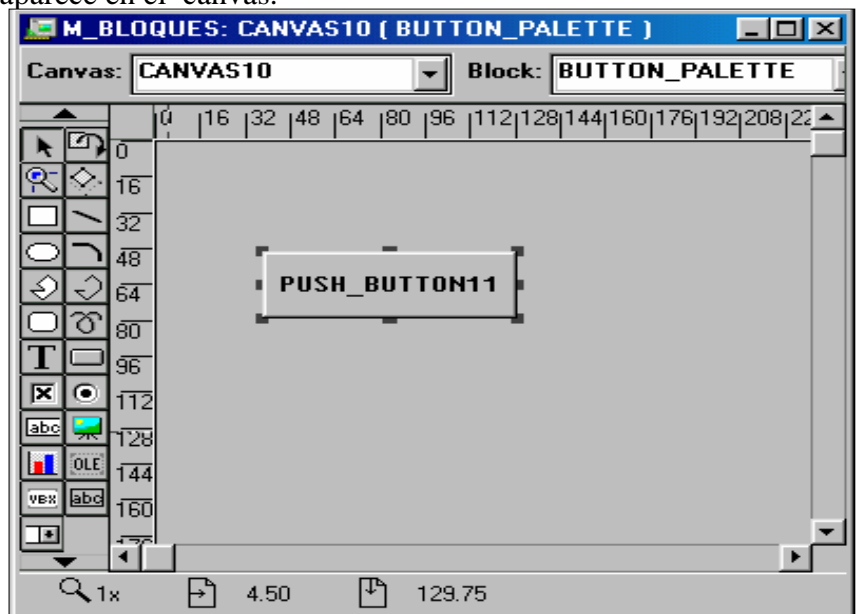
Lista de ítems.



Imagen 10.

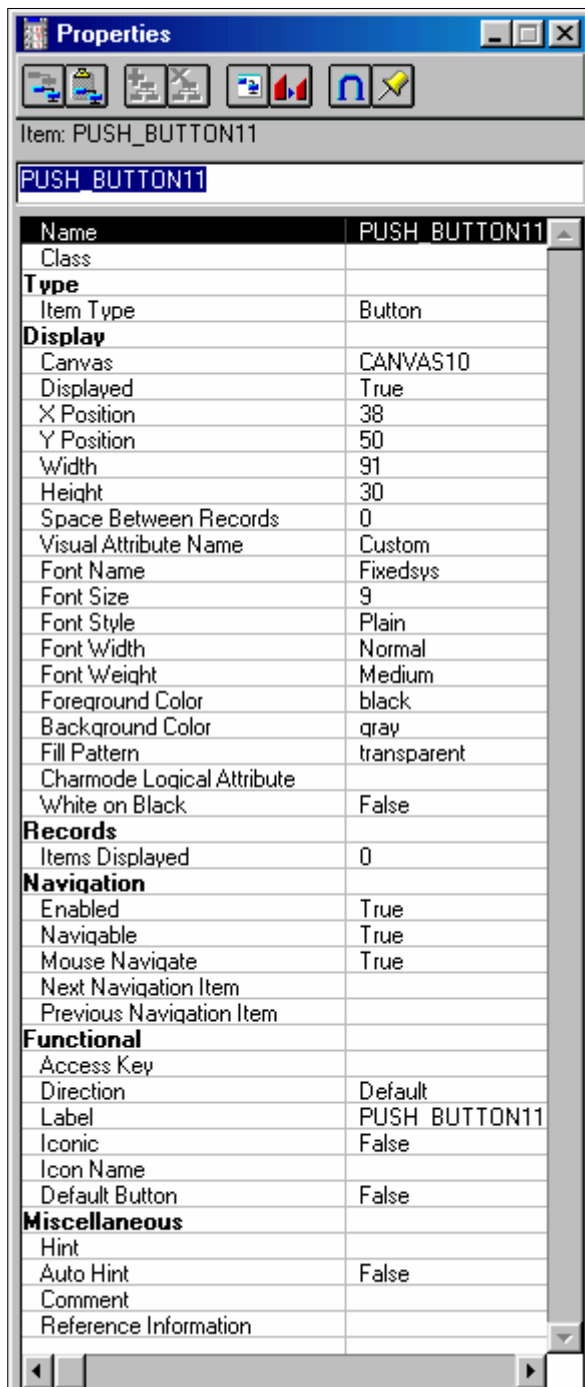
Veamos un ejemplo con un botón:

Una vez insertado el control o ítem que nos interesa, este aparece en el canvas.



En este caso hemos insertado un botón llamado PUSH_BUTTON11. Si pulsamos con el botón derecho sobre el aparecerá un menú contextual en el que podremos acceder a sus propiedades.

Imagen 11.



Observe la Imagen 11, podrá ver como las propiedades están divididas en bloques según su funcionalidad, dichos bloques o grupos de propiedades pueden o no ser comunes a otros ítems y todas pueden variar en numero, unos controles tendrán mas y otras menos, pero en general todas tendrán seis o siete posibles campos de acción sobre el ítem dependiendo del mismo:

- Nombre del ítem: será la etiqueta que defina a la variable.
- Tipo: definirá el tipo de ítem sobre el que se esta trabajando (si es un botón, un cuadro de texto, etc.)
- Display: aquí se gestionara todo lo relativo a la apariencia física del ítem en la aplicación.
- Registros: su función se detalla mas adelante
- Navegación: gestiona el control del foco entre los ítems.
- Funcional: establece diversas acciones para el control.
- Misceláneo: utilidades diversas para el control

-TEXT-ITEM:

Se usa para la entrada y salida de datos. Generalmente será tratado por código PL/SQL como una variable mas (antecediendo para ello el signo ':').

Un dato importante es que no es necesario pulsar intro para que el contenido escrito por el usuario dentro del ítem pase a memoria pues este se va introduciendo carácter a carácter a medida que se escribe.

En tiempo de diseño solo aparecerá el nombre del ítem en su interior nunca podremos visualizar su contenido salvo en tiempo de ejecución.

Propiedades mas importantes:

- Nombre: etiqueta con la que identificaremos a la variable
- Lienzo: lienzo en el que se encuentra ubicado el ítem
- Mostrado: si queremos que este oculto o no
- Posición: X horizontal, Y vertical
- Tipo de dato: especialmente importante dado que podremos especificar que datos va a contener el ítem. Como puede verse en la Imagen 12 el tipo de dato para este TEXT_ITEM de ejemplo seria CHAR, por tanto el ítem admitiría cualquier carácter ASCII como mínimo

Data	
Mirror Item	
Data Type	Char
Maximum Length	30
Fixed Length	False
Required	False
Format Mask	
Range Low Value	
Range High Value	
Default Value	
Copy Value from Item	

Imagen 12.

- Longitud Maxima: solo permitiría un numero máximo de caracteres.
- Longitud fija: obligaría al usuario a introducir un numero mínimo de caracteres
- Necesario: seria obligatorio escribir al menos un carácter
- Valor por defecto: valor con el que arrancaría el ítem en la aplicación
- Máscara de formato: por ejemplo 9999D99 solo permitiría cuatro números enteros y dos decimales
- Rango menor: establece el mínimo valor que puede admitir el ítem
- Rango mayor: establece el máximo valor que puede admitir el ítem.

Otra propiedad importante dentro del apartado Funcional es el salto automático. Que permite salto de línea dentro del ítem siempre que Varias Líneas tenga el valor TRUE.

-DISPLAY ITEM.

Es idéntico al Text Item con la diferencia de que no admite introducir texto, suele utilizarse para pantallas de confirmación o bien cuando necesitamos proteger algún campo.

-LIST-ITEM.

Despliega listas de valores, estas pueden ser de tres tipos:

- a) Text List (no permite insertar datos)



- b) Pop List (no permite insertar datos, también llamada lista desplegable)



- c) Combo Box (a diferencia de las dos anteriores la combo box si permite insertar datos)



Una propiedad muy importante dentro del apartado Funcional es Elemento de lista que me permitirá por ejemplo cargar los registros de un cursor dentro de un List Item.

-BOTONES.

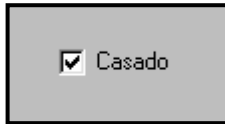
Es el ítem por excelencia, sin lugar a dudas el mas utilizado, y lo haremos fundamentalmente para realizar las operaciones de inserción, confirmación, borrado, etc.

Puede ser de dos tipos: de tipo texto donde el botón llevara un texto escrito en la parte frontal, o icónico, en cuyo caso el botón representara la acción para la que esta diseñado insertando un pequeño dibujo en su interior, para ello la propiedad Icónico debe ser TRUE y en Nombre de Icono debemos especificar el fichero que contiene el icono cuya extensión será '.ico', dicho fichero debe estar en un directorio especificado por Oracle que estará asociado a la variable de entorno TK21-ICON, en entornos Windows debemos acceder al registro para modificar esa variable, en UNIX se encontrara en el fichero de configuración Oracle.

El apartado mas importante dentro de las propiedades de los botones es Funcional.

-CHECKBOX.

Se utiliza para valores o campos que solo aceptan dos opciones, por ejemplo:

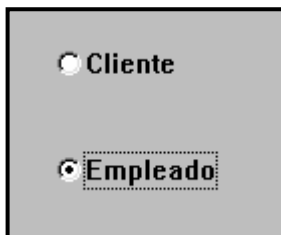


El apartado mas importante dentro de las propiedades es funcional, para localizar mejor estas variables se antepone el prefijo CH o CK en el identificador

-RADIO BUTTON.

Solo podrá admitir dos valores, activado o no activado, aunque es similar al Check Box, la diferencia fundamental es que el Radio Button no aparecerá solo sino junto a otros Radio Button y solamente uno de ellos podrá estar activado o seleccionado y lo que es mas, siempre habrá uno activado, se trata por tanto de opciones excluyentes entre si.

Su prefijo identificador suele ser RB.

**-CHAR ITEM.**

Inserta gráficos creados por ORACLE GRAPHICS.

-IMAGE ITEM.

Inserta gráficos de todo tipo (archivos bmp, jpg, tiff, etc..)

-OLE.

Permite “embeber” ficheros de otras aplicaciones de terceros como Microsoft Word, Microsoft Excell, etc

-VBX.

Permite a las aplicaciones Forms utilizar controles Visual Basic, generalmente realizados por terceras partes.

-TEXTO.

Nos permite introducir texto en el lienzo como si de un editor cualquiera se tratase para así crear aplicaciones con indicaciones, títulos, etc. Puede ver su efecto final en la Imagen 13.

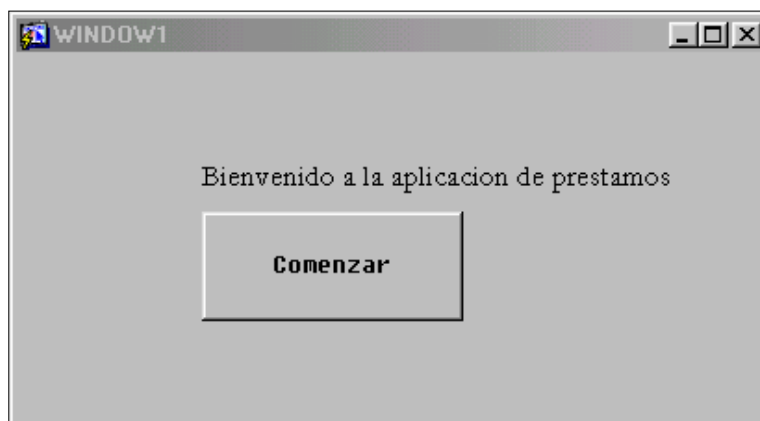


Imagen 13.

6-CANVAS Y VENTANAS:

Toda aplicación Forms como hemos podido comprobar parte de un canvas o lienzo, es imposible hacer nada sin este elemento y al menos un control, pasemos ahora a analizar mas en detalle este elemento.

Existen tres tipos de Canvas:

- a) Canvas de contenido: son los que Forms genera por defecto, ocupan como hemos visto toda la ventana principal y en un momento determinado el foco solo puede estar sobre un canvas de contenido en el caso de que hubiese mas de uno.
- b) Canvas apilados: son utilizados para poder ser visualizados dentro de un canvas de contenido. Por norma general son mas pequeños que los canvas de contenido y se utilizan para no tener que repetir una zona idéntica que deba estar presente dentro de dos o mas canvas de contenido (lo que nos obligaría a duplicar campos, ítems, etc..)

Es muy importante saber que un canvas apilado no puede situarse en el navegador de objetos antes del canvas de contenido y por regla general se coloca el ultimo.

Por otro lado no debemos invadir el área de un canvas apilado con un control o ítem del canvas de contenido pues existe la posibilidad de que el canvas apilado no aparezca

c) Barras de herramientas: pueden ser horizontales o verticales

Para crear un canvas apilado o una barra de herramientas acudiremos a las propiedades del canvas y en la propiedad Tipo elegiremos el tipo de canvas que deseamos crear.

Debemos recordar también que el orden de aparición de los canvas esta determinado por el primer ítem establecido en el primer bloque de datos del navegador de objetos como ya comentamos en capítulos anteriores.

*Un pequeño truco para forzar la aparición de un canvas es utilizar la función GO_ITEM('OBJETO') en el primer trigger que se ejecuta al arrancar una aplicación Forms, este es WHEN_NEW_FORM_INSTANCE, si llamamos a un ítem situado en un canvas utilizando este disparador será ese canvas el visualizado en primer lugar, esto tiene un pequeño inconveniente, en función de cómo se hayan colocado los ítems en el primer bloque el resultado puede no ser el esperado, debe analizarse detenidamente la ubicación de controles en el navegador de objetos pues sino ese GO_ITEM puede ser solapado, trataremos este tema en breve.

Una vez que hemos visto los tipos de canvas que podemos utilizar pasaremos a analizar como gestiona Oracle Forms el entorno visual de una aplicación. Si el lector ha trabajado ya con otros entornos visuales de desarrollo como bien podrían ser Visual Basic de Microsoft o Delphi de Borland, es importante destacar que Forms tiene un funcionamiento bastante distinto a estas aplicaciones en cuanto al programador se refiere pues el usuario final apenas notara diferencia entre unas y otras. Oracle ha optado por un sistema de navegación entre pantallas sin duda complejo y que requiere un análisis previo considerable, comencemos por los canvas de contenido.

6.1-CANVAS DE CONTENIDO:

Ninguna aplicación Forms puede existir sin dos elementos básicos; un canvas y una ventana. A tenor de lo visto hasta el momento podría pensarse que ambas cosas son similares pero no es así, se trata de elementos bien distintos, intentemos comprenderlo mejor haciendo un símil con un pintor, un lienzo y un marco.

LIENZO/CANVAS



Imagen 14.

MARCO/VENTANA

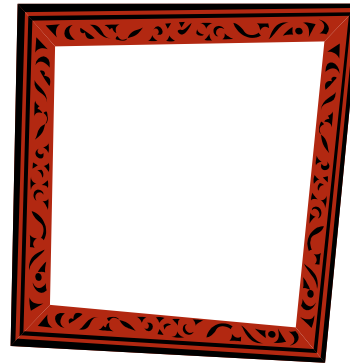


Imagen 15.

Imaginemos que terminado el cuadro el pintor se da cuenta de que ha conseguido retratar mucho mejor a la mujer de azul que a la mujer de rojo y decide que solo va a aparecer en el cuadro final la primera, para ello corta el lienzo por la mitad y ajusta el marco a este lienzo, de tal modo que el cuadro final presenta este aspecto

CUADRO FINAL/VISION DEL USUARIO



Imagen 16.

Sustituya ahora la palabra lienzo por canvas y marco por ventana y habrá descubierto como actúa Forms. El canvas es simplemente el entorno de trabajo donde se insertan los controles (text ítem, botones, check box, etc...) , lo que en este caso y con mucha imaginación eran las modelos del pintor, y por otro lado tenemos la ventana, que para el pintor era un marco, podemos ver entonces que la apariencia final en cuanto a dimensiones de la aplicación va estar directamente relacionada con el tamaño de “salida” que hayamos especificado para la ventana, podríamos decir que la ventana busca el área del canvas que se le ha indicado visualizar, si ambas no coinciden o la ventana es mas pequeña que el área de trabajo del canvas pueden no aparecer todos los

elementos o controles que insertamos en el canvas. Si nuestro canvas mide 100x100 y la ventana asociada al canvas es de 80x80 no podremos visualizar todo el área de trabajo del canvas, pudiendo aparecer una pantalla en ejecución con el aspecto de la Imagen 17.

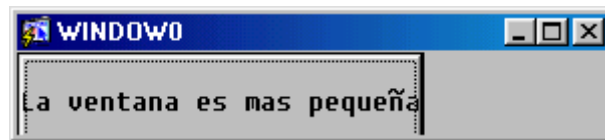


Imagen 17.

Por tanto es de suma importancia entender que la apariencia final de la aplicación va a depender de la interacción entre los canvas y las ventanas a las que estén asociados, veamos que pasos hay que dar para asociar correctamente un canvas de contenido y una ventana en una aplicación muy simple, retomemos a nuestro viejo amigo “Hola Mundo”.

Creamos un canvas con un botón y un cuadro de texto que permita escribir la cadena Hola Mundo en su interior al pulsar el botón, dimensionamos el canvas en la paleta de propiedades a 200x200

Imagen 18.

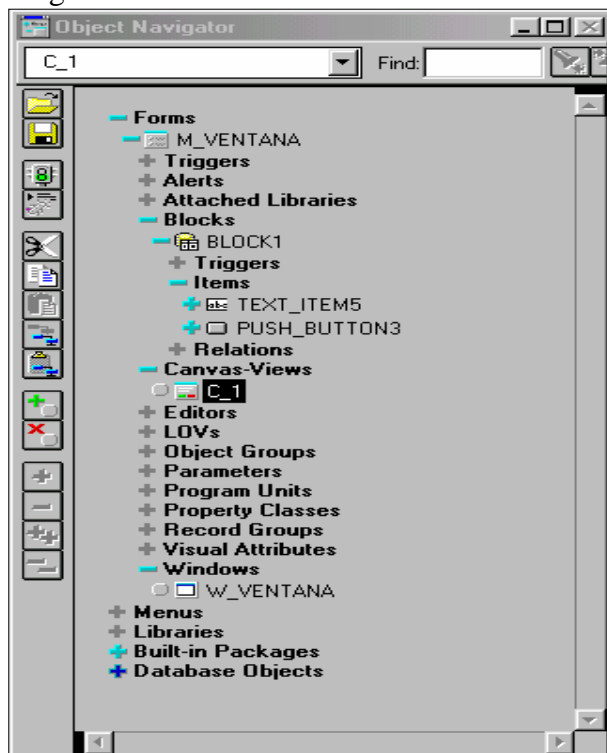


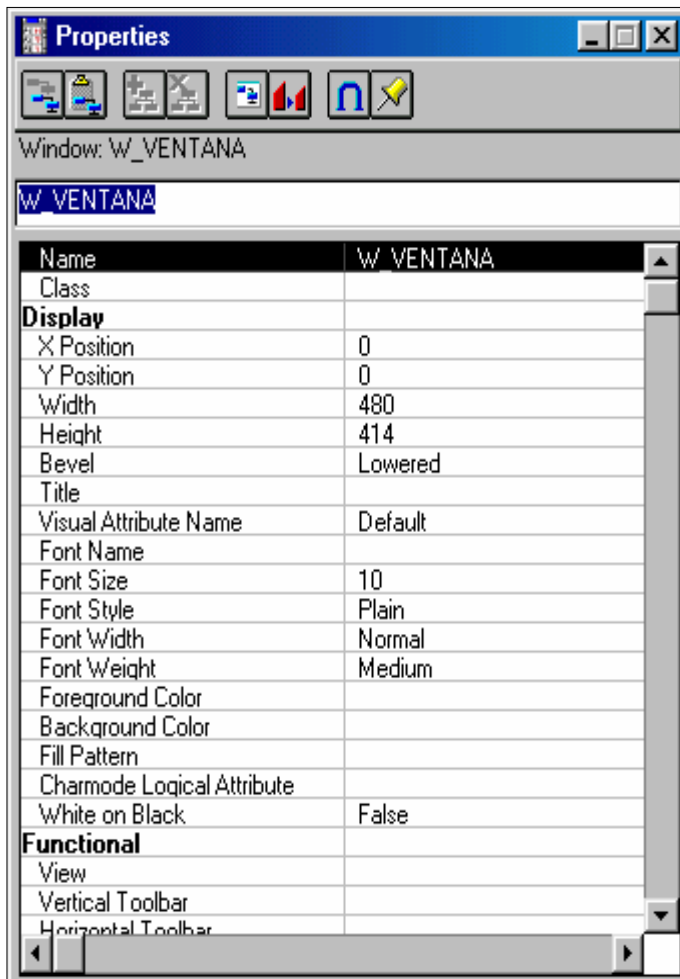
Imagen 19.



Como puede ver en la Imagen 19 dentro de las propiedades del canvas hemos dimensionado el canvas a 200x200 (también podríamos haberlo hecho con el ratón en el editor de diseño) y en el apartado Funcional hemos asociado este canvas con la ventana W_VENTANA donde además le indicamos que dicha ventana posicione su esquina

superior izquierda en las coordenadas $X=0$ e $Y=0$ del canvas, si tenemos en cuenta las dimensiones de la ventana tal y como pueden apreciarse en la Imagen 20, podremos comprobar como sin lugar a dudas la ventana podrá visualizar toda el área del canvas sobradamente pues es de 480 de ancho por 414 de alto.

Imagen 20.



Si observa ahora el resultado de la compilación vera como la pantalla resultante es la especificada en el apartado funcional del canvas para la ventana asociada a este.

Este sistema de trabajo nos lleva a tener que analizar siempre el tamaño de canvas y ventana, una solución frecuente suele ser asignar a ambos el mismo tamaño con lo que nos aseguramos que todo se va a visualizar.

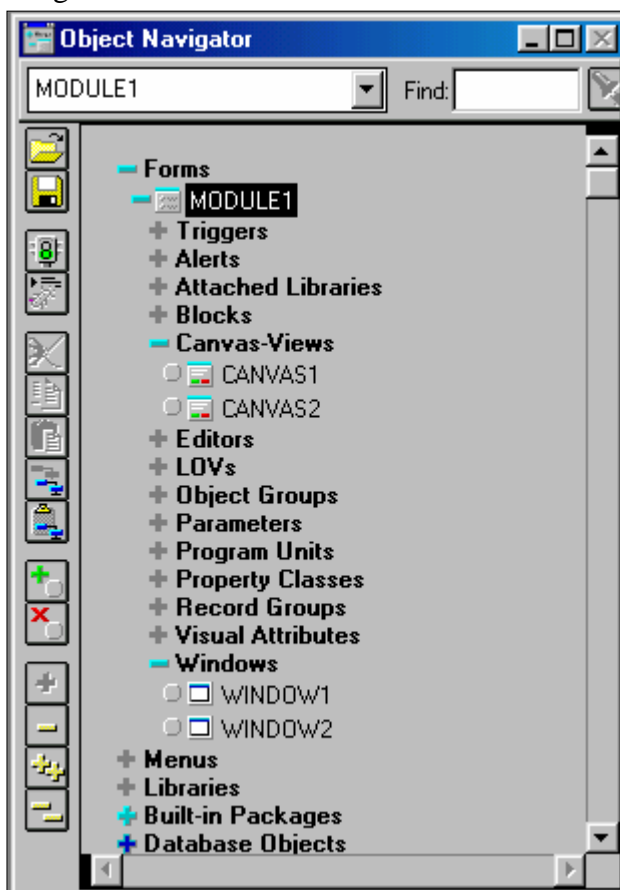
6.2-TRABAJAR CON VARIOS CANVAS DE CONTENIDO:

Imagine ahora que su aplicación crece y que precisa como suele ser habitual de mas de una ventana (ventana desde el punto de vista del usuario final), en este punto las opciones empiezan a ser varias, si decidimos crear la típica aplicación de “ventana tras

La aplicación que acabamos de desarrollar es como habrá podido apreciar muy limitada en cuanto a la navegación entre ventanas se refiere, que ocurriría si usted quisiese mostrar al mismo tiempo dos ventanas a un usuario, con una sola ventana para varios canvas es imposible, o se ve un canvas u otro pero nunca dos a la vez, la solución a este problema es crear mas de una ventana en tiempo de diseño, veamos como hacerlo.

Crearemos una aplicación similar a la anterior en cuanto a funcionalidad pero distinta en el paso de un canvas a otro pues estos van a estar contenidos en ventanas distintas. En primer lugar cree dos canvas y dos ventanas y asócielos respectivamente como se muestra en la Imagen 22.

Imagen 22.



A continuación dimensione canvas y ventanas a 200x200

Ahora insertaremos un botón en cada canvas que llame al botón del otro canvas, para ello usaremos de nuevo el BUILT IN 'GO_ITEM' .

Antes de ejecutar su aplicación repare en la posición de aparición de las ventanas tal y como se muestra en las imágenes 23 y 24.

Imagen 23.

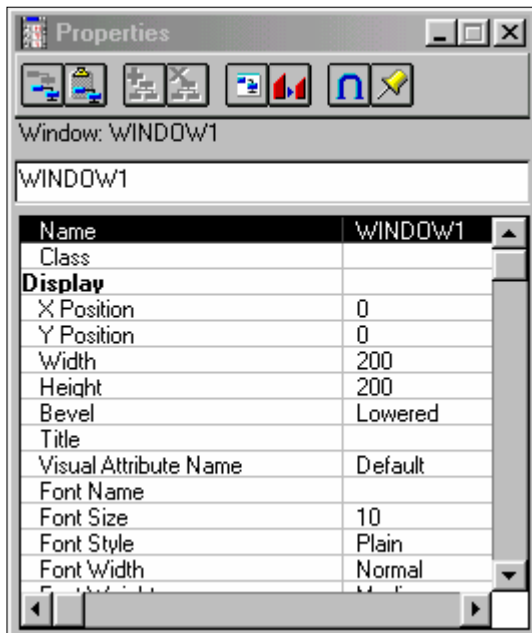
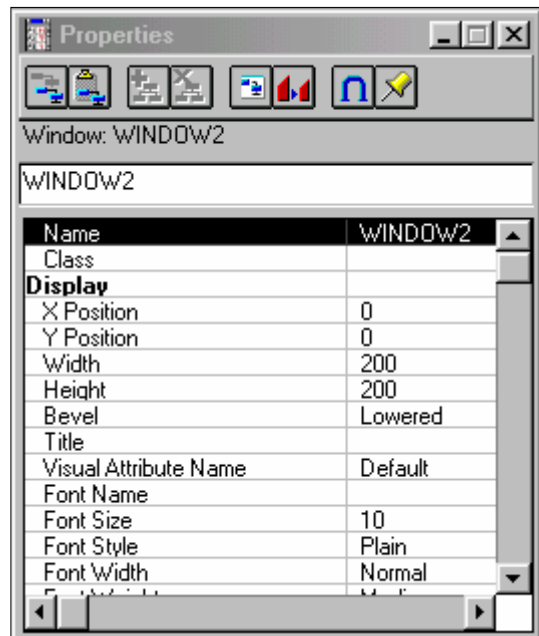


Imagen 24.



Puede ver que la posición de aparición de ambas ventanas en Windows toma la mismas coordenadas, algo en lo que hasta ahora no habíamos reparado, para que ambas ventanas no se solapen deberá desplazar la posición de visualización inicial de alguna de las dos, situando la ventana WINDOW2 en la posición Y = 250 debería ser suficiente, haga los cambios y después compile la aplicación, no olvide llamar desde un control a otro con GO_ITEM y recuerde que el primer canvas visualizado será el que tenga el primer ítem dentro del primer bloque de datos en el navegador de objetos en este caso CANVAS1.

El resultado de la ejecución debería mostrar en primer lugar la ventana WINDOW1 y al pulsar el botón de dicha ventana, debería aparecer la ventana WINDOW2 cuyo botón debería enviar el foco a la ventana uno, si quisiese hacer modales dichas ventanas acceda a la propiedad MODAL dentro del apartado FUNCIONAL en las propiedades de las ventanas.

Existe una forma de conseguir que aparezca mas de una ventana al inicio de la aplicación y de nuevo esta relacionado con la posición de los objetos en el navegador de objetos, si observa el primer elemento dentro del navegador podrá ver que lo primero que encontramos después del Modulo inicial de la aplicación es un trigger o disparador, instancie un nuevo trigger en este punto llamado WHEN_NEW_FORM_INSTANCE y escriba en el una llamada al botón que se encuentra en la ventana WINDOW2, podrá comprobar al ejecutar el programa que aparecen las dos ventanas al mismo tiempo y lo que es mas importante el foco ha quedado sobre la ventana dos, esto es debido a la

secuencia de carga y aparición de los canvas en memoria y en pantalla, Forms va cargando los canvas desde el primero que este asociado al primer objeto del primer

bloque (y ese es el que mantiene el foco) hasta el ultimo, pero solo muestra el primero que cargo en memoria, por tanto si no incluimos el trigger `WHEN_NEW_FORM_INSTANCE` esta secuencia se ve inalterada, si lo incluimos cargara primero el canvas indicado en el trigger y a continuación el primer canvas asociado al primer objeto del primer bloque, pero no mas, no es posible por tanto cargar mas de dos formularios iniciales o canvas de contenido en una aplicación Forms.

6.3-CANVAS APILADOS:

Hasta ahora hemos tratado con canvas de contenido los cuales son los canvas comúnmente mas utilizados, no obstante existe una situación en la que hacerlo todo con canvas de contenido seria cuando menos bastante laborioso, utilicemos un ejemplo.

Imagine que tiene una aplicación con tres canvas de contenido sobre la misma ventana, el primero le da la bienvenida a la aplicación, el segundo le presenta una ventana con una serie de cajas de texto y botones para poder insertar datos en una tabla de una supuesta base de datos para la gestión de empleados y la tercera ventana le permite confirmar la inserción del nuevo empleado que tecleo en la ventana anterior, una aplicación de este tipo podría tener el aspecto de las imágenes 25, 26 y 27

Imagen 25.

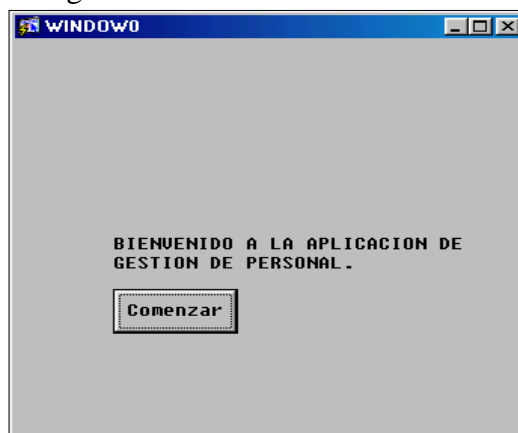


Imagen 26.

Una ventana de aplicación con un título azul que dice 'WINDOW0'. El fondo es gris. Hay cuatro campos de texto con etiquetas a la izquierda: 'Codigo de empleado.', 'Nombre y apellidos', 'Fecha ingreso' y 'Sueldo anual'. A la derecha de los campos 'Fecha ingreso' y 'Sueldo anual' hay un botón rectangular con el texto 'Insertar'.

Imagen 27.

Una ventana de aplicación con un título azul que dice 'WINDOW0'. El fondo es gris. Hay cuatro campos de texto con etiquetas a la izquierda: 'Codigo de empleado.', 'Nombre y apellidos', 'Fecha ingreso' y 'Sueldo anual'. A la derecha de los campos 'Fecha ingreso' y 'Sueldo anual' hay dos botones rectangulares: 'Confirmar' (arriba) y 'Volver' (abajo).

Tanto en la pantalla de inserción como en la de confirmación existen zonas que se repiten de forma idéntica, esta es la situación perfecta para un canvas apilado, con este tipo de canvas podemos diseñar de forma aislada zonas de uno o mas canvas que se repitan de forma “clonica”, un canvas apilado actúa como un canvas invisible en el sentido de que no tendrá ningún tipo de señal que lo identifique en la aplicación final, en este caso haremos un canvas apilado con la zona de los canvas que comprende los campos de Código de empleado y Nombre y Apellidos de tal forma que no tendremos que dibujar dos veces los mismo campos con nombres de variable distintos en los dos canvas, evitando así las complicaciones que esto conlleva como por ejemplo el hecho de tener que mover por código la información contenida en esos campos cuando el usuario pulsase el botón “Insertar” para que apareciesen en la pantalla siguiente, lo cual aumenta la posibilidad de cometer errores o simplemente nos da mas trabajo, trabajo que se puede evitar, veamos como.

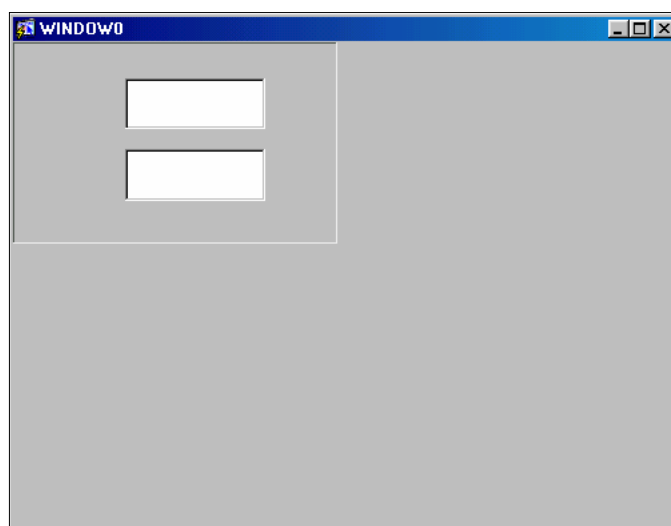
En primer lugar para diseñar un canvas apilado hay que crear un nuevo canvas teniendo muy en cuenta dos cosas:

- a) Un canvas apilado debe diseñarse dentro de un área exclusiva para el
- b) Ningún objeto de ningún otro canvas puede situarse encima de un canvas apilado, en caso contrario no se mostrara

Estudiemos el tema mas en detalle

Creemos un canvas en el navegador de objetos que llamaremos C_APILADO, accediendo a sus propiedades en el apartado Tipo indicaremos que el canvas es apilado, redimensionamos el canvas a 180 de ancho por 110 de alto para después acceder al apartado Vista Apilada del mismo canvas, es aquí donde establecemos el máximo tamaño que tendrá el canvas apilado en la ventana, y por tanto será este área en la que en su interior solo podrá haber objetos del canvas apilado, normalmente esta zona suele tener las mismas dimensiones que el canvas apilado, en este caso 180 x 110, pruebe a insertar un par de controles y compile la aplicación, el resultado obtenido será parecido al de la Imagen 27..

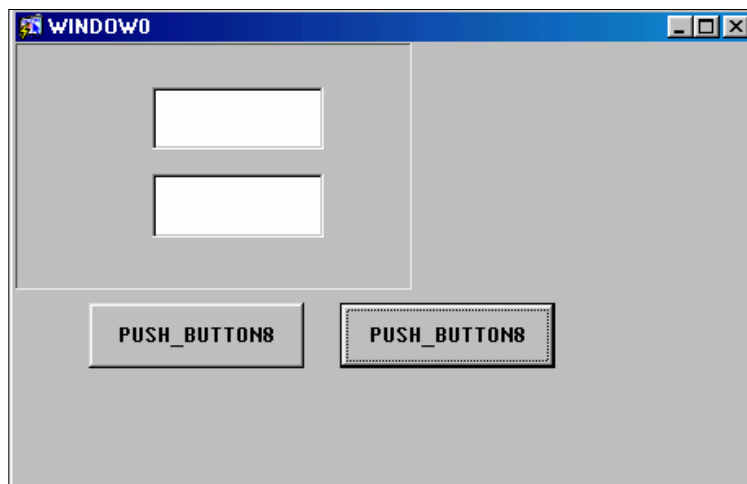
Imagen 27.



El área tomada por el canvas apilado se muestra hundida (para producir este efecto acceda a las propiedades y seleccione Hundido en Bisel) y este área es la que no debe ser invadida por ningún otro canvas que vaya a aparecer en la ventana WINDOW0, en caso contrario el canvas apilado no aparecerá.

A continuación cree un canvas de contenido asociado a la misma ventana e inserte uno o dos botones, el tamaño de este canvas nos es indiferente, puede ser mayor o menos que el apilado, comenzar en la posición 0,0 o no, puede tener 2 o 20 controles pero recuerde, no invada el área del canvas apilado, después compile y observe el resultado, debería ser algo similar a lo mostrado en la figura 28.

Imagen 28.

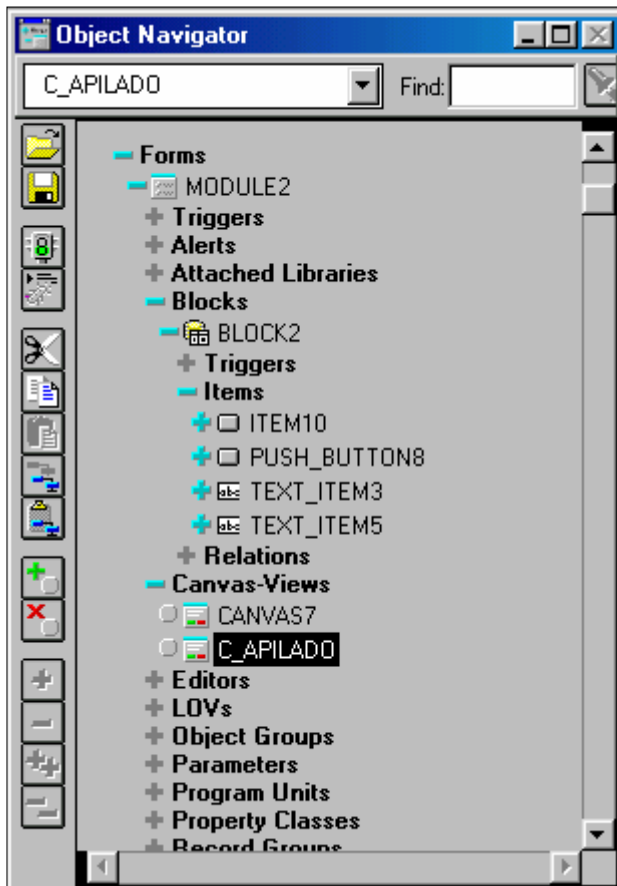


Como puede observar los dos canvas aparecen al mismo tiempo en la misma ventana, cosa que era imposible con dos canvas de contenido, podría usted seguir creando canvas de contenido asociados a la misma ventana sin ningún problema siempre y cuando no invadiese el área del canvas apilado como ya comentamos.

Existe otro detalle muy importante, como siempre nos vamos a referir al orden de colocación de los objetos en el navegador, si colocase usted algún control del canvas apilado como primer ítem del primer bloque ocurriría lo que hemos visto en varias ocasiones, los canvas posteriores al canvas apilado se verían solapados, pero además ahora debemos tener otro factor muy en cuenta y es la ubicación ya no de los controles en el navegador, sino de los propios canvas, si se coloca un canvas apilado delante de uno de los canvas de contenido sobre los que debe aparecer o se coloca el canvas apilado el primero en la lista de canvas del navegador, el canvas apilado no aparecerá, esto también es debido al orden de aparición que realiza Forms sobre los canvas, debido a ello es costumbre entre los programadores de Forms colocar los canvas apilados al final de la lista de canvas o bien después de todos los canvas de contenido sobre los que

tiene que aparecer, la ubicación elegida para el ejemplo anterior se muestra en la Imagen 29.

Imagen 29.



En este navegador puede ver un canvas de contenido llamado CANVAS7 y un canvas apilado llamado C_APILADO, los text ítem corresponden al canvas apilado y los botones al canvas de contenido, cree una aplicación similar a esta y experimente cambiando de lugar los controles y canvas para ver por usted mismo los resultados que se producen.

Un ultimo dato sobre los canvas apilados, una ventana puede contener mas de un canvas apilado, la forma de creación es idéntica al anterior y las reglas que rigen para su colocación son las mismas que hemos visto hasta ahora, lógicamente cuantos mas canvas apilados mas se restringe la inserción de controles de otros canvas y requiere mucha mayor atención del programador hacia el espacio físico de la ventana en la pantalla.

Como puede verse los canvas apilados resultan muy útiles para mostrar información que deba “viajar” entre diversas ventanas sin la necesidad de que tengamos que utilizar código para moverla, otra ventaja es que esa información siempre estará presente para el programador ahorrando así campos, memoria y controles.

6.4-BARRAS DE HERRAMIENTAS:

Debido a su complejidad y a que las barras de herramientas precisan un conocimiento mas avanzado de Forms estudiaremos este tipo de canvas en capítulos posteriores junto a elementos como los menús de la aplicación.

-Built_In:

Existen una serie de funciones interesantes para acceder y modificar propiedades de canvas, estos Built in son:

-GET_CANVAS_PROPERTY(‘Nombre_del_canvas’,PROPIEDAD);
Obtiene el estado de una propiedad.

-SET_CANVAS_PROPERTY(‘Nombre_del_canvas’,PROPIEDAD);
Modifica el estado de una propiedad.

-ID_NULL(‘Nombre del canvas’); Me indica si existe o no ese canvas.

-SHOW_VIEW(‘nombre_del_canvas’) Muestra el canvas indicado

Del mismo modo existen una serie de Built in para las ventanas.

-GET_WINDOW_PROPERTY(‘nombre_ventana’,PROPIEDAD);

-SET_WINDOW_PROPERTY(‘nombre_ventana’,PROPIEDAD);

-HIDE_WINDOW(‘NOMBRE_VENTANA’); oculta la ventana, pero no la descarga de memoria con lo que sus datos siguen disponibles

-RESIZE_WINDOW(‘nombre_ventana’,Tamaño X, Tamaño Y) modifica las dimensiones de la ventana.

-SHOW_WINDOW(‘Nombre de ventana’) muestra la ventana.

7- BUILT-IN PARA CONEXIÓN A LA BBDD

Antes de continuar profundizando en Forms se hace necesario explicar estos Built In. En ocasiones nuestras aplicaciones llegaran a un punto en el que serán varios usuarios los que deban acceder al mismo programa y a la misma o a distintas bases de datos y como es lógico suponer no todos los usuarios deben acceder a los mismos datos, existen privilegios y cuestiones de seguridad que nos obligan a que esto sea así, encontramos por tanto en Forms unas funciones que permiten asegurar la conexión basadas en el reconocimiento del nombre de usuario y una contraseña, las podremos utilizar al arrancar la aplicación o incluso dentro de un bloque tabla.

En líneas generales el código PL/SQL para la conexión a la base de datos seria así:

DECLARE

vUSER	VARCHAR2(80);
vPASSWORD	VARCHAR2(80);
vCONNECT	VARCHAR2(80);

BEGIN

--Nos desconectamos y presentamos la pantalla de conexión

LOGOUT;
LOGON_SCREEN;

--Obtenemos los datos tecleados en la ventana de conexión

vUSER := GET_APPLICATION_PROPERTY(USERNAME);
vPASSWORD := GET_APPLICATION_PROPERTY(PASSWORD);
vCONNECT := GET_APPLICATION_PROPERTY(CONNECT_STRING);

IF vCONNECT IS NOT NULL THEN

LOGON (vUSER,vPASSWORD||'@'||vCONNECT);

ELSE

LOGON(vUSER,vPASSWORD);

END IF;

--Si la conexión fuese rechazada nos devolvería la ventana de conexión en forma modal,

--de tal manera que deberíamos pulsar cancelar para poder volver a la aplicación

--si sucediese esto ultimo podríamos cancelar la aplicación con EXIT FORM;

END;

La ventana de conexión mostrara el aspecto de la Imagen 30.

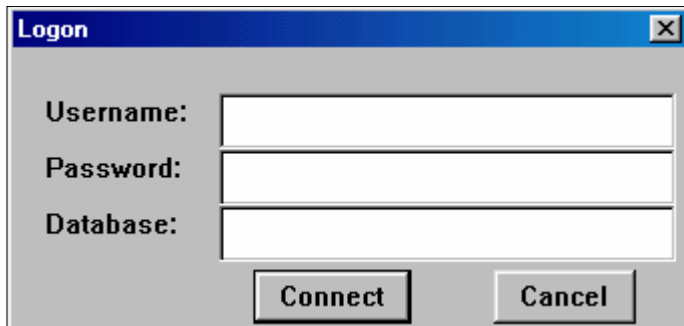


Imagen 30.

Lo ideal seria incluir esta porción de código como una función a la que podamos llamar en cualquier momento, para ello acceda a Unidades de Programa dentro del navegador de objetos y añada la función con el retorno que mas le pueda interesar para averiguar si se puedo establecer la conexión o no.

A continuación puede usted leer otra serie de Built In para gestionar la conexión de diversas acciones contra la BBDD

CONSULTAS

ABORT_QUERY	Cancela la Query actual.
COUNT_QUERY	Nos devuelve el número de registros que habría al ejecutar la Query.
ENTER_QUERY	Entra en modo consulta.
EXECUTE_QUERY:	Ejecuta la Query.

REGISTROS

CLEAR_RECORD: Limpia el registro actual.
CREATE_RECORD: Crea un nuevo registro
DELETE_RECORD : Borra el registro actual.
DOW: Se sitúa en el siguiente registro en el mismo ítem.
UP: Se sitúa en el anterior registro en el mismo ítem.
GO_RECORD: Se posiciona en el registro elegido.
FIRST_RECORD: Se posiciona en el primer registro.
INSERT_RECORD: Inserta un nuevo registren
LAST_RECORD: Se posiciona en el último registro
LOCK_RECORD: Bloquea el registro actual.
NEXT_RECORD: Se sitúa en el siguiente registro, primer ítem.
PREVIOUS_RECORD: Se sitúa en el anterior registro, primer ítem.
UPDATE_RECORD: Actualiza el registro actual.

TRANSACCIONES

LOGON	Conexión con la Base de Datos.
LOGON_SCREEN	Presentación de la pantalla de conexión.
LOGOUT	Desconexión de la Base de Datos.

8-VARIABLES DEL SISTEMA:

Existen una serie de variables en Forms a las que usted puede acceder en todo momento para saber el estado de múltiples elementos de la aplicación, dichas variables solo pueden ser leídas y no modificadas en la mayoría de los casos, a continuación se detallan las mas significativas.

SYSTEM.BLOCK_STATUS

SYSTEM.COORDINATION_OPERATION

SYSTEM.CURRENT_BLOCK Bloque de navegación actual.

SYSTEM.CURRENT_DATETIME Fecha y hora actual.

SYSTEM.CURRENT_FORM Form actual.

SYSTEM.CURRENT_ITEM Item actual

SYSTEM.CURRENT_VALUE Valor del ítem actual

SYSTEM.CURSOR_BLOCK

SYSTEM.CURSOR_ITEM

SYSTEM.CURSOR_RECORD

SYSTEM.CURSOR_VALUE

SYSTEM.EFFECTIVE_DATE*

Cambia la fecha y la hora, su uso esta restringido

SYSTEMEFFECTIVE_DATE := 'DD-MON-YYYY HH:MI:SS'

SYSTEM.EVENT_WINDOW

SYSTEM.FORM_STATUS

SYSTEM.LAST_QUERY

SYSTEM.LAST_RECORD

SYSTEM.MASTER_BLOCK

SYSTEM.MODE NORMAL, QUERY, ENTRE_QUERY

SYSTEM.MOUSE_BUTTON_PRESSED

SYSTEM.MOUSE_BUTTON_SHIFT_STATE

SYSTEM.MOUSE_ITEM

SYSTEM.MOUSE_CANVAS

SYSTEM.MOUSE_X_POS

SYSTEM.MOUSE_Y_POS

SYSTEM.MOUSE_RECORD

SYSTEM.MOUSE_RECORD_OFFSET

SYSTEM.RECORD_STATUS CHANGED, INSERT, NEW, QUERY

SYSTEM.TRIGGER_BLOCK

SYSTEM.TRIGGER_ITEM

SYSTEM.TRIGGER_RECORD

9- FUNCIONALIDAD:

Analicemos ahora de forma mas detallada la capacidad funcional de los ítems mas utilizados para los que hasta el momento solo hemos visto una introducción muy general.

Podemos distinguir en Forms dos tipos de ítems, los generales y los específicos, los generales podremos encontrarlos en casi cualquier entorno de desarrollo visual, los específicos son propios de Forms. No obstante otras entornos de desarrollo como Visual Basic de Microsoft disponen de controles similares a los controles especificos de Forms.

a) Items generales:

- Check Box
- Radio Button
- Radio Group
- List Item
- Image Item

b) Items especificos:

- Alertas
- Record Group
- Lov
- Timers
- Editores
- Parámetros
- Grupo de objetos

9.1-CHECK BOX:



Admite dos estados; chequeado o no chequeado, o bien true o false, por tanto solo nos va a servir para optar entre dos opciones. Siempre que se instancie un check box debemos definir tres propiedades:

- Valor por defecto
- Valor cuando esta chequeado
- Valor cuando no esta chequeado

Para asignar estas propiedades acudiremos a las propiedades del checkbox, observe como ha sido definido un checkbox en las imágenes 30 y 31.

Imagen 30.

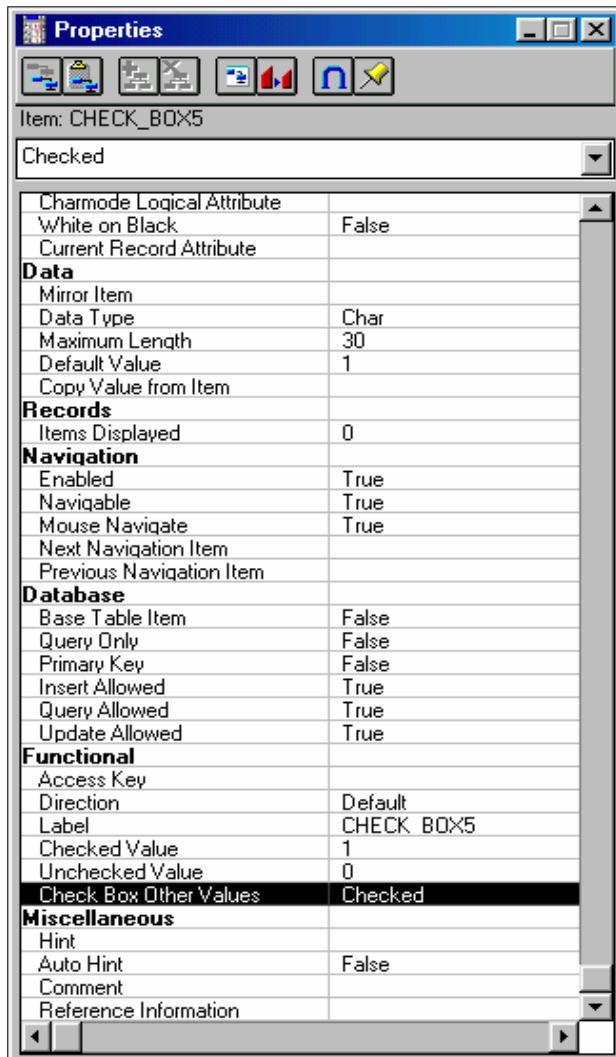
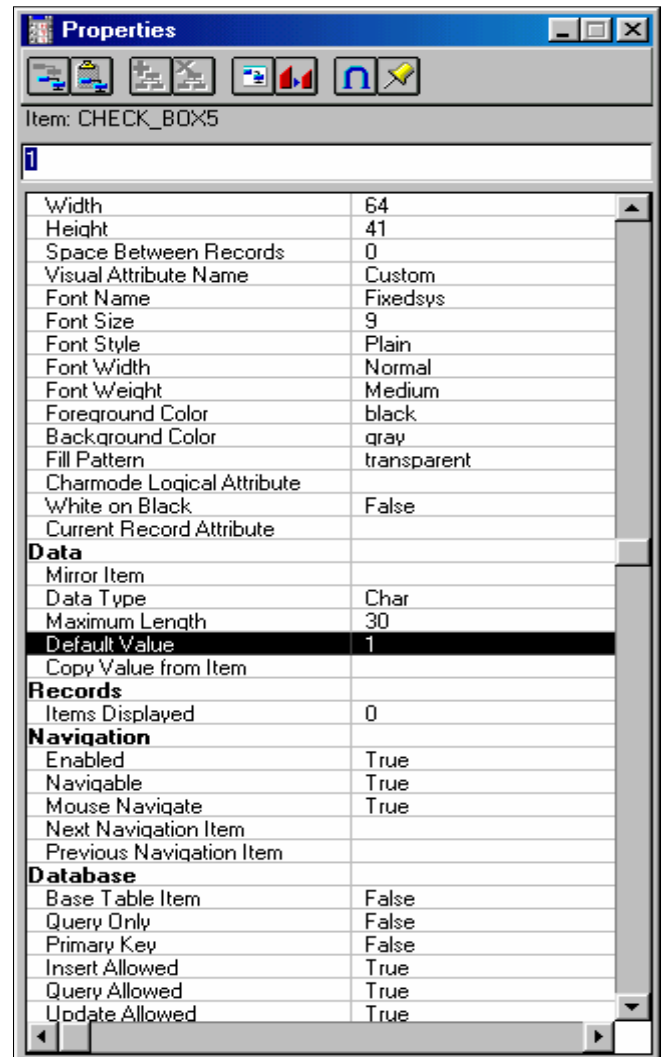


Imagen 31.



Como puede ver en la Imagen 30 en el apartado Funcional el check box toma el valor 1 cuando este esta chequeado y el valor 0 cuando no lo esta, además se ha establecido la propiedad Check Box Other Values como true de tal modo que todo valor que no sea ni 0 ni 1 se entenderá como chequeado.

Si observa ahora la imagen 31 vera como en el apartado Datos el valor por defecto del checkbox se ha inicializado a 1 por lo que este checkbox aparecerá chequeado en la aplicación, esto es muy útil ante opciones en un programa que varíen muy poco, imagine por ejemplo una compañía aérea que tiene una casilla de verificación en su aplicación de venta de billetes en la que se especifica si la persona es

cliente o empleado, lógicamente se inicializaria a cliente siempre ya que serán clientes el noventa y nueve por ciento de los casos.

Otra propiedad importante es la que aparece en la Imagen 30 llamada Tipo de dato, dentro del apartado Datos, asignada a tipo Char, esto es así porque esta propiedad se usa normalmente cuando se asocia el check box a una columna de la BBDD (mediante un bloque tabla) del tipo: Español S/N (Español Si o No) donde el valor chequeado seria S y el valor no chequeado seria N, aunque no tiene porque ser un solo carácter, podría ser directamente SI y NO, esto solo depende del valor establecido en chequeado y no chequeado.

-Propiedades mas importantes:

- Etiqueta
- Tipo de dato
- Valor por defecto
- Valor comprobado
- Valor no comprobado
- Otros valores

-Built in mas importantes:

```
GET_ITEM_PROPERTY('NOMBRE_CHECKBOX');  
SET_ITEM_PROPERTY('NOMBRE_CHECKBOX');  
CHECKBOX_CHECKED('NOMBRE_CHECKBOX'); me devolverá  
"true" si esta chequeado y "false" si es el caso contrario, solo admite retornos en  
variables boolean
```

Triggers mas importantes:

WHEN_CHECK_BOX_CHANGED

Se activa cuando el checkbox cambia de estado

*Si asociamos un checkbox al crear un bloque tabla mediante la opción Casilla de control y en sus propiedades pongo en valor comprobado el valor "X" la base de datos lo activara siempre que este valor coincida con el contenido en la fila correspondiente como si fuese el usuario

9.2-RADIO BUTTON:



Un radiobutton es un ítem que solo tiene sentido junto a otro radiobutton o dentro de un conjunto de varios radiobutton pues lo utilizaremos para escoger entre

opciones excluyentes entre si, y lo que es mas importante siempre habrá uno de ellos seleccionado, por tanto son ítems que especifican valores dentro de un conjunto existiendo para ello un radiobutton por cada uno de los valores entre los que podamos escoger.

Un radiobutton estará dentro de un radio group y este ultimo será el que contenga el valor del radiobutton seleccionado.

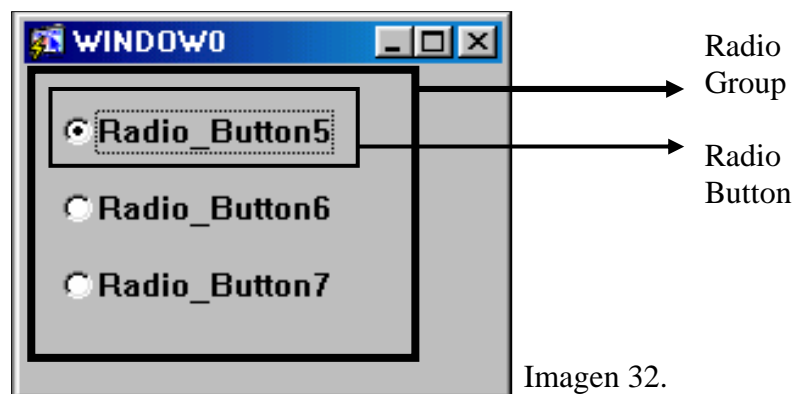


Imagen 32.

De este modo si quisiésemos saber que radiobutton esta seleccionado solamente deberíamos extraer el contenido de la variable Radio Group, veamos un ejemplo basado en la Imagen 32.

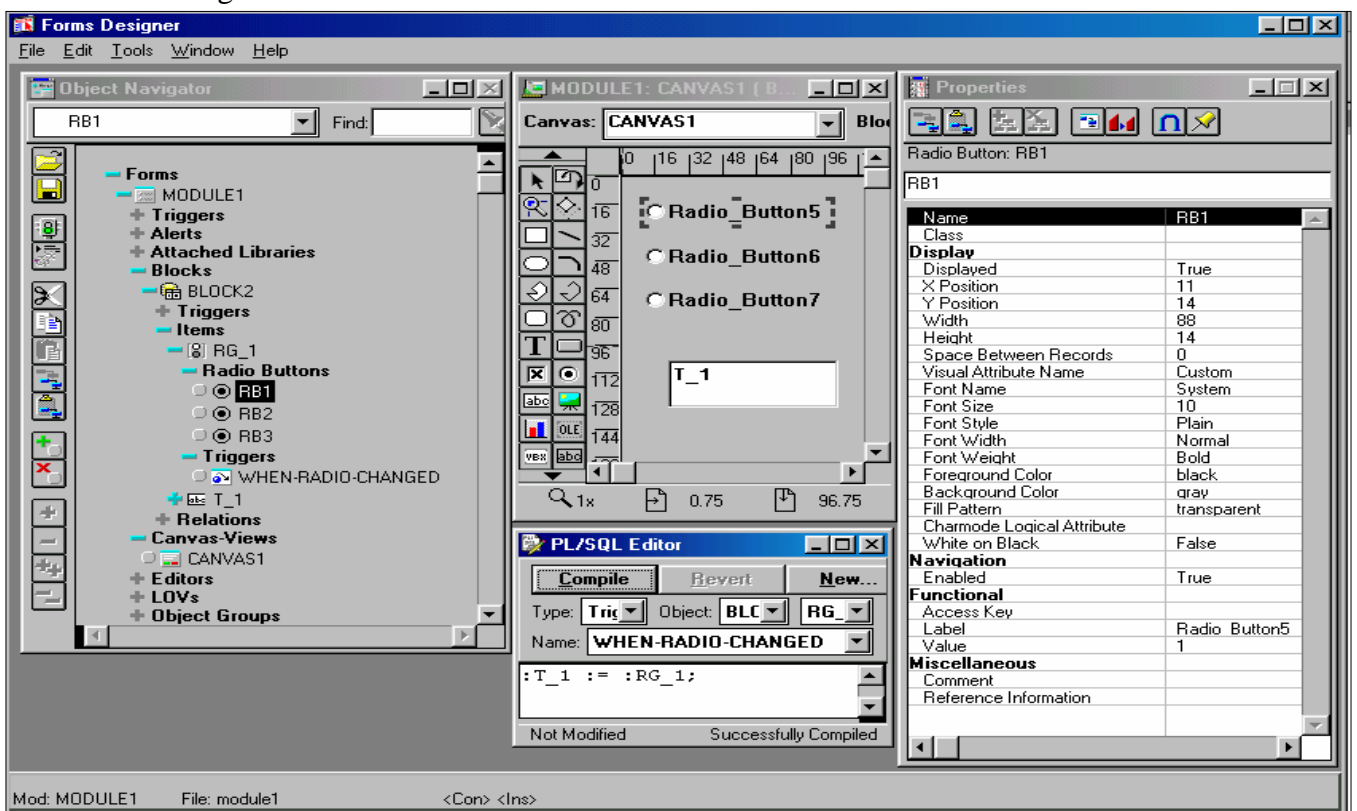


Imagen 33.

Observe ahora la Imagen 33, en ella puede ver como se han definido los radiobutton y como los hemos incluido dentro del radiogroup generado por Forms, después hemos dado valores al radiobutton en el apartado funcional de las propiedades y vea como el radiobutton RB_1 toma el valor 1 cuando este es seleccionado, el radiobutton RB_2 tomara el valor 2 y el radiobutton RB_3 tomara el valor 3 uno de este valor será el que tenga en un momento determinado el radiogroup RG_1.

Hemos añadido un trigger al radiogroup que se disparara cada vez que cambie el valor de selección de algún radiobutton, este trigger es WHEN_RADIO_CHANGED, asociado a este trigger hemos añadido el código :T_1 := :RG_1 , lo que hará que cada vez que cambie la selección en un radio button aparezca en el text ítem el numero del radiobutton seleccionado (que como acabamos de explicar será el que tenga en cada momento el radiogroup) el resultado de la compilación es el que vemos en la Imagen 34.



Imagen 34.

Hemos seleccionado todas las posibilidades para que vea como actúan los radiogroup.

-Built In mas importantes:

```
GET_RADIO_BUTTON_PROPERTY(radio_group,radio_button,propiedad);  
SET_RADIO_BUTTON_PROPERTY(radio_group,radio_button,propiedad);
```

-Trigger mas importantes:

WHEN_RADIO_CHANGED

*se dispara cuando el usuario selecciona un radiobutton del radiogroup.

9.3-LIST ITEM (listas):



El tratamiento de datos se hace abrumador en muchas ocasiones, imagine que precisa mostrar a un usuario un Text Item que dinamicamente fuese leyendo registros de una columna de una tabla para ir mostrando al usuario datos de esa columna en concreto, para ayudarnos en esta tarea Forms pone a nuestra disposición las listas, que en definitiva serán conjuntos de datos (normalmente de una misma tabla o columna) mostrados en un ítem similar al Text Item.

Las listas son un conjunto de valores almacenados en una variable común. Por norma general estos valores son cadenas de unos treinta caracteres de tipo carácter.

¡Es muy importante saber que todo elemento de lista se compone de un nombre o etiqueta y un valor que denominamos Elemento de Lista y Valor del Elemento de Lista respectivamente!

Existen dos tipos de listas:

a) List Item:

Se dividen a su vez en tres tipos:

- Pop List: es de solo lectura para el usuario y de lectura y escritura para el programador.
- Tlist: idéntica a la Pop List en su funcionalidad pero no en su apariencia final
- Combo Box: igual que las anteriores salvo en que el usuario también podrá insertar datos en ellas, no obstante esa escritura deberá ser controlada a través de código PL/SQL.

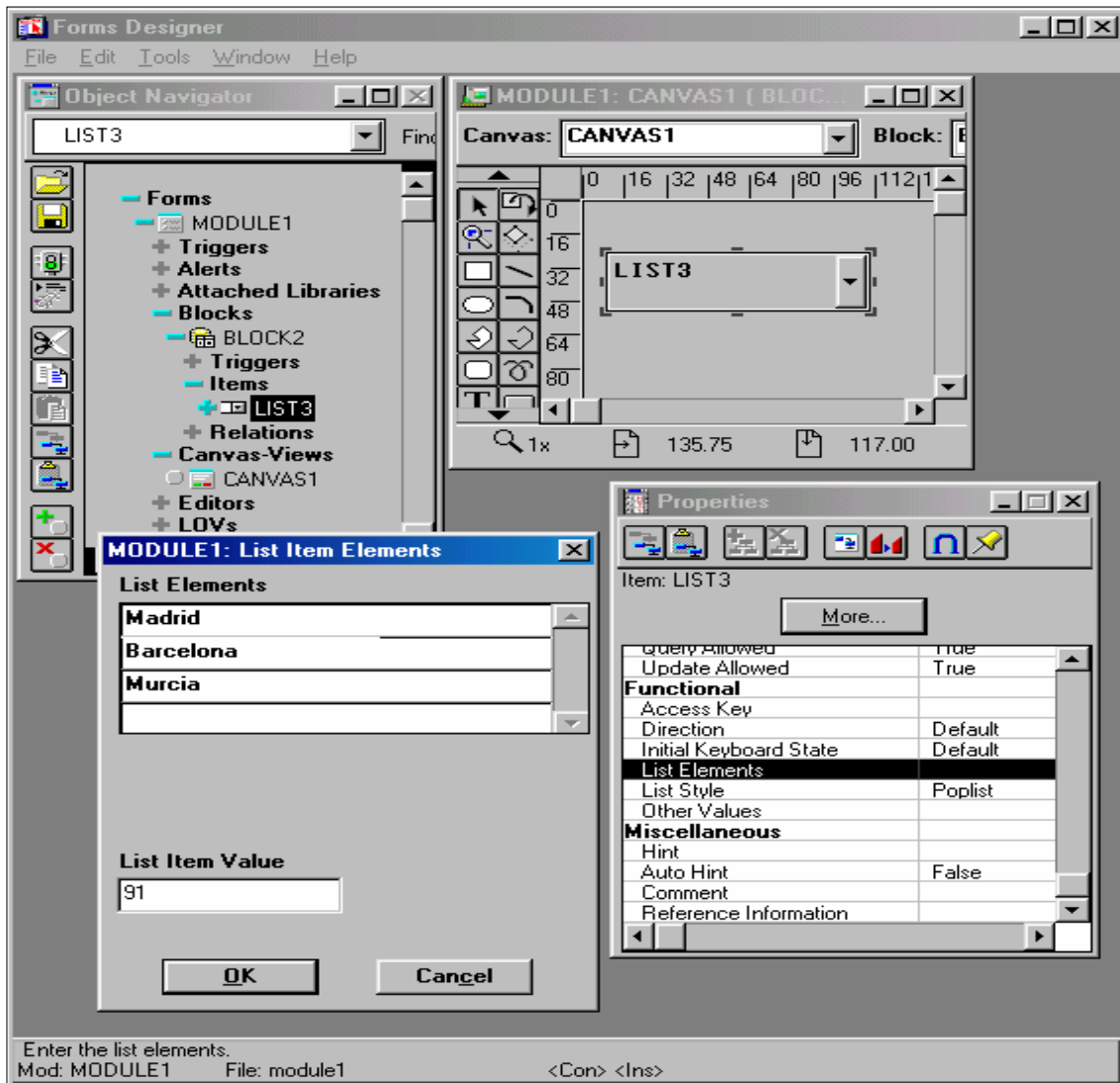
Las listas pueden ser “rellenadas” de dos formas:

-En tiempo de diseño: insertaremos un List Item en el editor de diseño como en la Imagen 35. En dicha imagen puede ver como se ha añadido un List Item, Forms crea estos ítems por defecto del tipo Pop List. Accediendo al apartado Funcional en las propiedades de la lista encontramos Elementos de Lista, al pulsar esta propiedad en la parte superior de la ventana de propiedades encontramos un botón con la etiqueta Mas, si pulsamos sobre el se nos presenta un cuadro en el que iremos insertando los datos de la lista, en este caso son ciudades, inserte una ciudad en Elementos de Lista y verá como en la parte inferior de dicho cuadro tiene un cuadro de texto titulado Valor del List Item, este valor será el asociado al elemento de lista mostrado y será este el valor a buscar por programa para saber que elemento de lista ha escogido el usuario, este último punto es muy importante ya que al referirnos a la etiqueta que identifica a la lista en el código esta siempre nos devolverá el valor del List Item, nunca el nombre asignado al elemento.

Accediendo al apartado Datos de las propiedades podemos establecer el valor por defecto de la lista que aparecerá al arrancar la aplicación.

Cree esta aplicación usted mismo y compílela para ver el resultado.

Imagen 35.



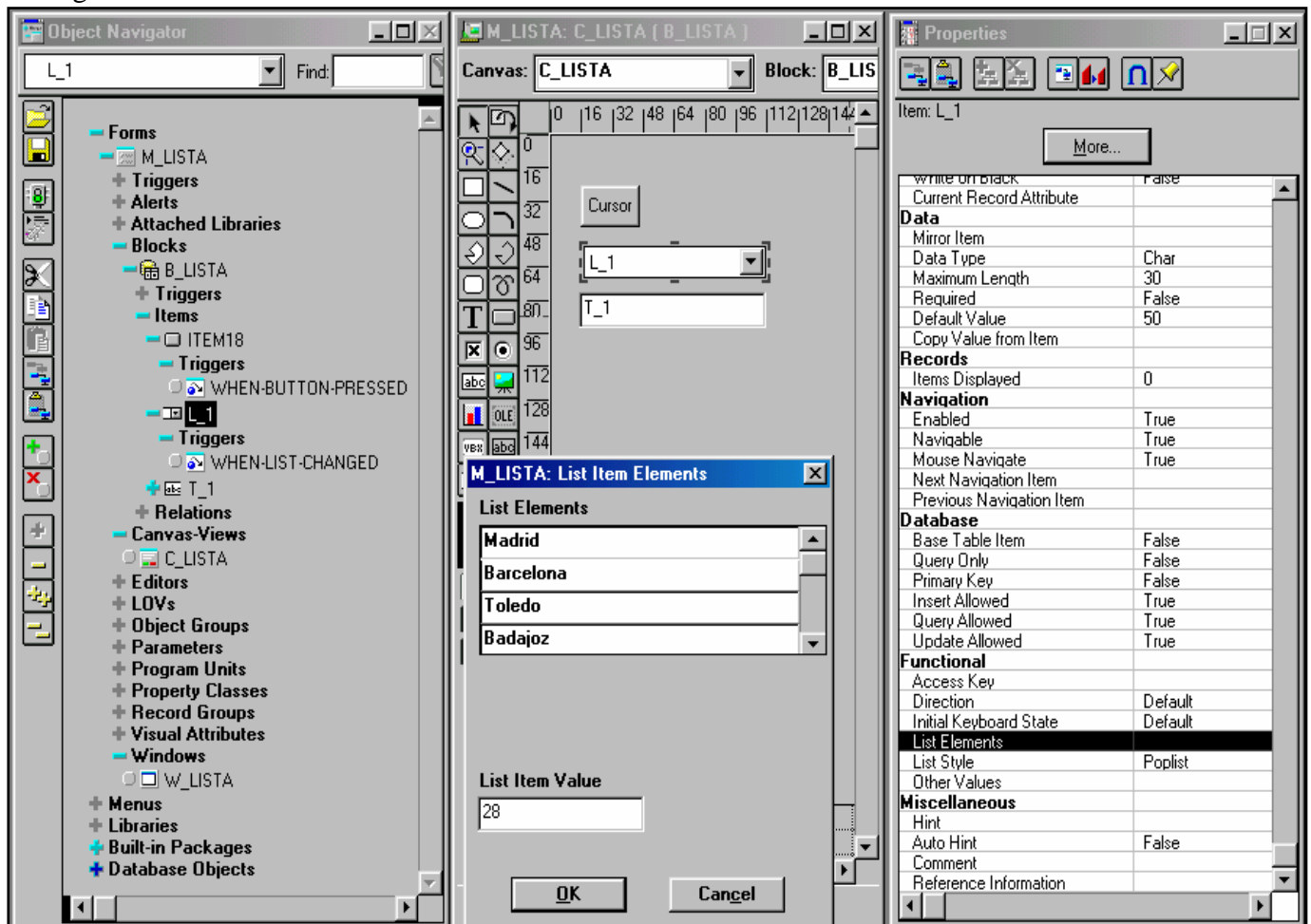
-En tiempo de ejecución:

Como habrá podido apreciar rápidamente el método explicado para cargar elementos en tiempo de diseño no es muy adecuado para crear listas que contengan decenas o cientos de posibles elementos, es más, imagine que esos valores cambian a menudo lo que nos llevaría a tener que modificar el programa frecuentemente, para esas listas mas grandes (que generalmente extraerán los datos de tablas de la BBDD) Forms dispone de mejores herramientas que comentamos a continuación.

Si queremos mostrar una Pop List de forma que cargue los datos desde una tabla la mejor opción será agregar algo de código. Para nuestro siguiente ejemplo crearemos un List Item de tipo Pop List (recuerde que por defecto este es el tipo de lista que crea Forms) y un Button.

La intención en este ejemplo es crear un cursor y mediante una serie de funciones o Built in ya existentes en Forms llenar el List Item creado previamente mediante las instrucciones que hayamos asociado al botón, observe los elementos que intervienen en la Imagen 36.

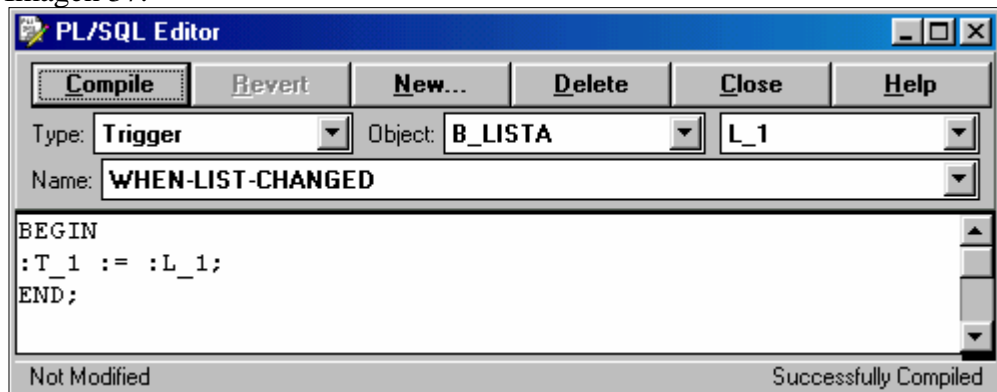
Imagen 36.



Analicemos lo que vemos con detenimiento, en primer lugar observe que la lista es de tipo Pop List, que ya existen algunos elementos de lista creados de forma manual y no en tiempo de ejecución, por otra parte aparece en el navegador de objetos un disparador que no hemos visto hasta ahora; **WHEN_LIST_CHANGED**.

Primer objetivo del ejemplo: conseguir que cada vez que cambie el elemento de la lista por la acción del usuario aparezca en el Text Item T_1 el valor de dicho elemento, para conseguir esto utilizamos el disparador WHEN_LIST_CHANGED que es lógicamente un trigger específico de los List Item. Instancie dicho disparador y añada la línea de código mostrada en la Imagen 37.

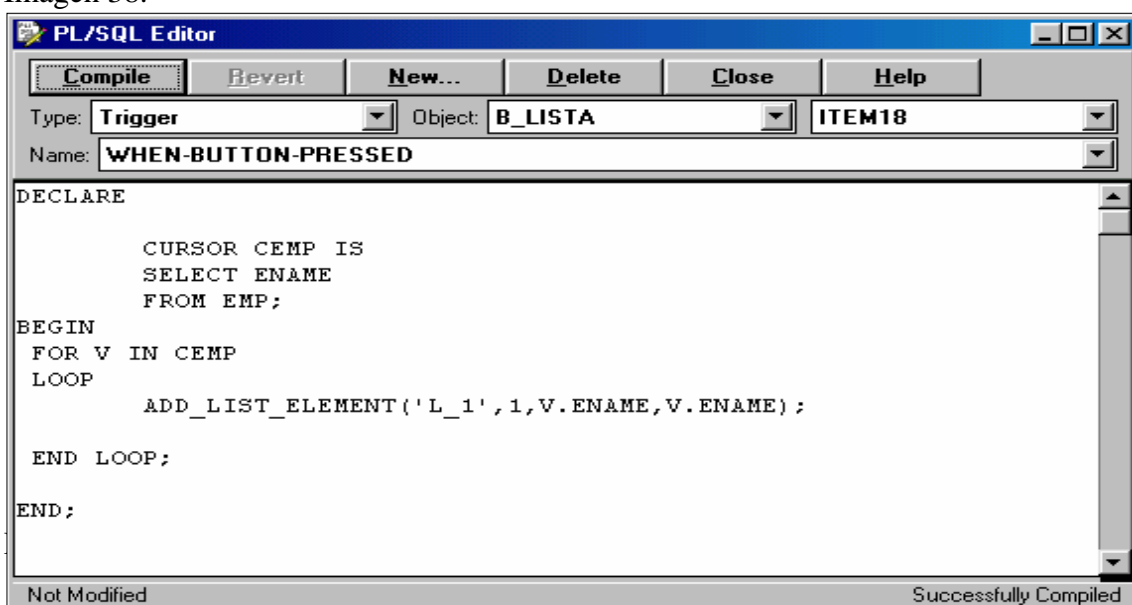
Imagen 37.



Con este código comprobaremos al ejecutar el programa como solo podemos extraer del List Item L_1 el valor del elemento no el nombre o etiqueta asociada a ese valor que es el denominado Elemento de Lista.

Segundo objetivo del ejemplo: cargar desde un cursor la lista L_1. En el botón que hemos creado asociaremos al evento WHEN_BUTTON_PRESSED el siguiente código:

Imagen 38.



El código mostrado en la Imagen 38 requiere que nos detengamos y analicemos lo que allí esta escrito.

DECLARE

--Declaración del cursor sobre la tabla EMP de la que extraemos todos las filas de la
--columna ENAME

```
CURSOR CEMP IS  
SELECT ENAME  
FROM EMP;
```

BEGIN

--Bucle For con apertura del cursor previamente declarado
--El bucle recorrerá todo el cursor y lo ira cargando en la lista L_1 mediante el Built In
--ADD_LIST_ELEMENT, este tipo de For también cerrara el cursor cuando termine de
--leerlo

```
FOR V IN CEMP  
LOOP  
    ADD_LIST_ELEMENT('L_1',1,V.ENAME,V.ENAME);  
  
END LOOP;
```

END;

Como puede ver este código declara un cursor sobre la tabla EMP, hasta aquí todo es “normal”, pero habrá podido observar que en la sección BEGIN hay “cosas raras”. El bucle For con cursor de PL/SQL es una herramienta potentísima que consta de dos elementos:

- Una variable puntero. ('V')
- Un cursor previamente declarado. ('CEMP')

La variable puntero en este caso llamada 'V' funciona de la siguiente manera; al abrir el cursor en la propia instrucción for el cursor es generado y la variable V apunta a la primera fila recuperada, en este caso el primer campo ENAME de la tabla EMP, a continuación entramos en el bucle esa fila recuperada es procesada.

No debemos confundir el tratamiento de la variable 'V' con los punteros del lenguaje C pues no es exactamente lo mismo.

El proceso que realizamos de la fila recuperada es el siguiente

```
ADD_LIST_ELEMENT('L_1',1,V.ENAME,V.ENAME);
```

Este es un Built In que carga elementos en un List Item, en este caso L_1, analicemos los parámetros reales pasados a esta función.

```
('L_1',  
 1,  
 V.ENAME,  
 V.ENAME)
```

```
('Nombre_del_List_Item,  
 Posición del registro en la lista,  
 Elemento de Lista,  
 Valor del Elemento de lista)
```

Nombre del List Item: List Item sobre el que queremos cargar datos

Posición del registro en la lista: Posición en la lista donde queremos cargar el registro en este caso al ser 1 ira desplazando todos los que hubiese hacia abajo ya que inserta siempre en la posición 1, el 1 pasaría a ser el 2, el 2 el 3 etc., este proceso se repetiría en cada iteración del bucle.

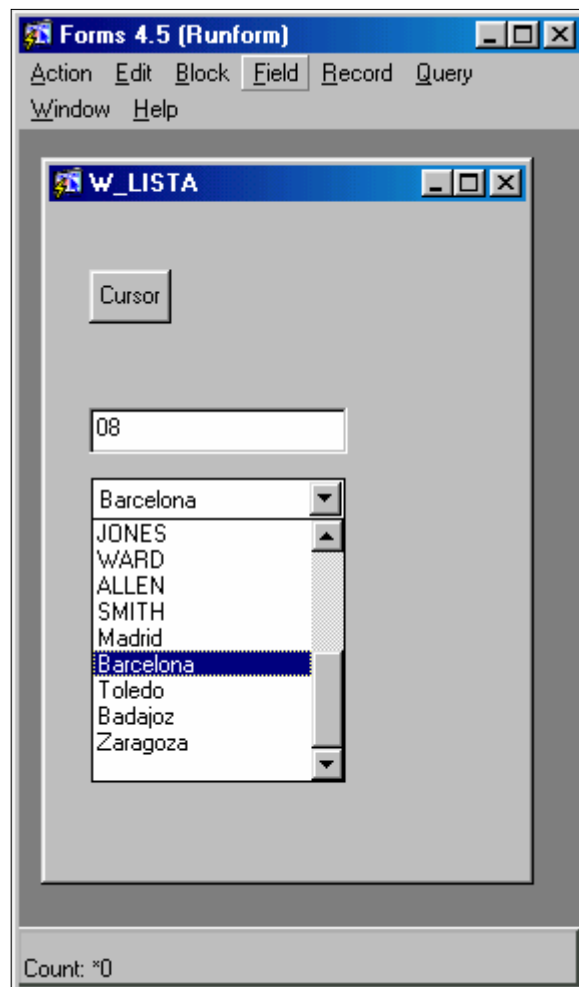
Elemento de Lista: Etiqueta o nombre del elemento de lista incluido (serian el equivalente a las ciudades que insertamos antes manualmente)

Valor del elemento de Lista: es el valor que nos devuelve la lista de cada elemento, son los prefijos telefónicos que insertamos en un ejemplo anterior manualmente.

Observe como utilizamos la variable puntero 'V' para recuperar en cada momento el registro recuperado e introducirlo en la lista L_1.

Compile la aplicación observe el resultado que deberá ser parecido al de la Imagen 38.

Imagen 38.



Podrá ver como se han cargado todos los nombres de la tabla EMP tras pulsar el botón y como al cambiar de registro el trigger WHEN_LIST_CHANGED y su código asociado insertan el valor del Elemento de Lista en el Text Item T_1, si selecciona un nombre de empleado vera como el valor del elemento es el mismo que el nombre pues así se lo indicamos en el bucle for con V.ENAME en el ultimo parámetro de ADD_LIST_ELEMENT.

-Propiedades mas importantes:

- Elementos de Lista
- Valor por defecto

-Built In mas importantes:

ADD_LIST_ELEMENT('Lista',Indice,Etiqueta,Valor)

CLEAR_LIST('Lista'); Elimina todos los elementos de la lista

DELETE_LIST_ELEMENT('Lista',Indice); Elimina un elemento concreto

GET_LIST_ELEMENT_COUNT('Lista'); Nos da el numero total de elementos.

GET_LIST_ELEMENT_LABEL('Lista',Indice); Nos devuelve la etiqueta de un elemento

GET_LIST_ELEMENT_VALUE;Nos devuelve el valor de un elemento de lista.

-Triggers mas importantes:

WHEN_LIST_CHANGED Se ejecuta cuando cambiamos el elemento de lista seleccionado.

WHEN_LIST_ACTIVATED Se ejecuta cuando el foco se sitúa en la lista.

9.4-LOVS Y RECORD GROUPS.



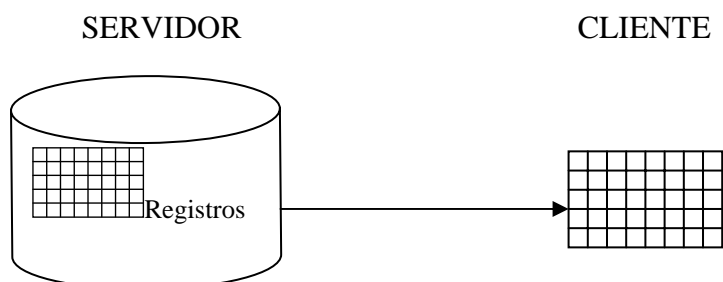
Lovs y Record Groups son probablemente las dos herramientas mas potentes de las que dispone Forms para el tratamiento de grupos de registros.

Comencemos definiendo cada una de ellas.

-Record Groups: es una minitabla que se aloja en la memoria del equipo cliente para de esta manera descargar trafico de la red y procesos en el gestor. encontramos dos tipos.

Estáticos: no provienen de la BBDD y se gestionan como una matriz

Dinámicos: se rellenan desde tablas remotas.



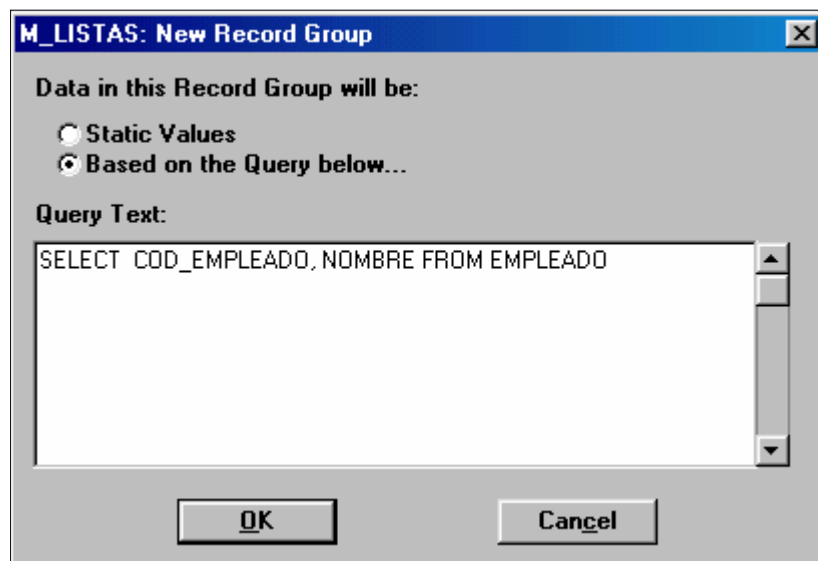
-Listas de valores Lovs: una lista de valores es la interfaz gráfica de un grupo de registros o Record Group.

Entre otras utilidades una Lov sirve para evitar que un usuario escoja o introduzca valores incorrectos entre un rango de valores posibles..

Veamos en el siguiente ejemplo como crear un Record Group para acontinuacion cargarlo en una Lov.

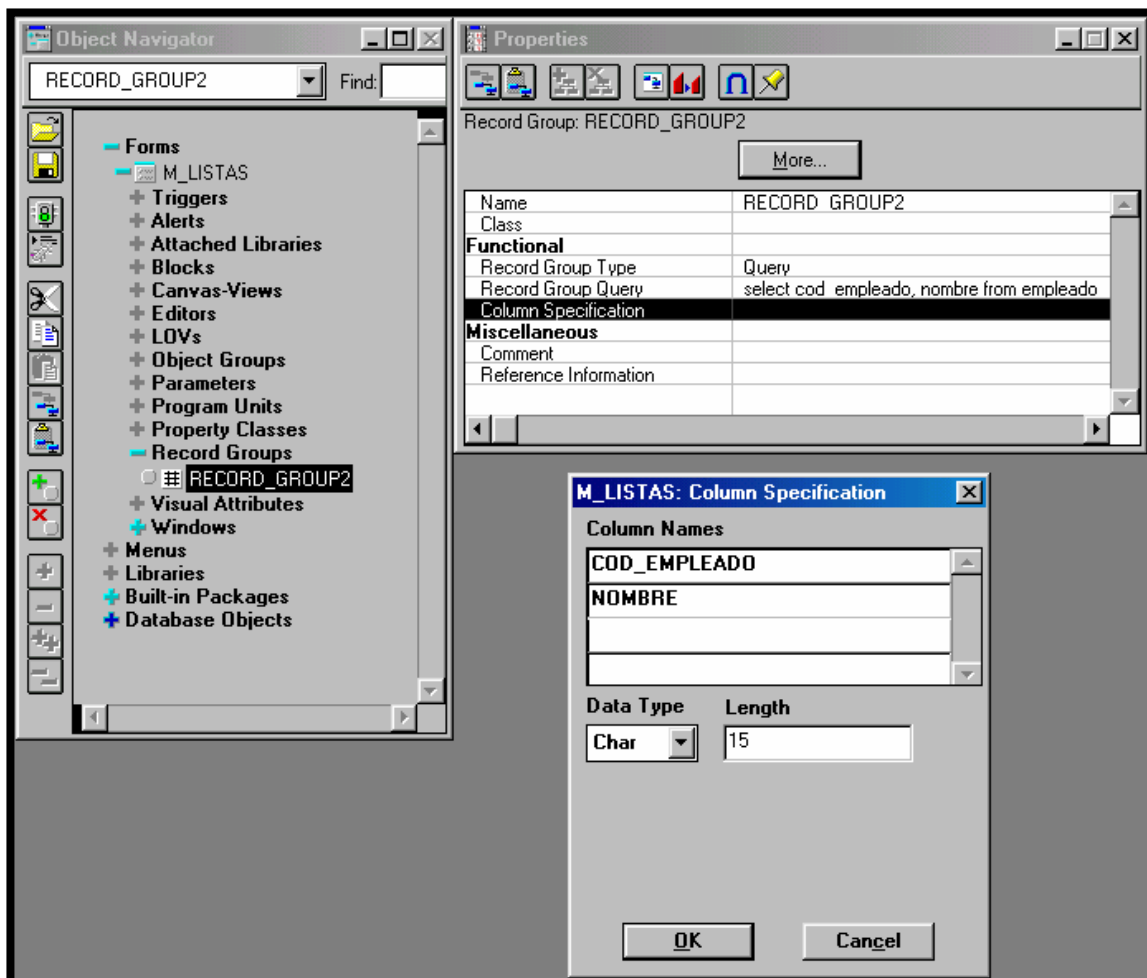
En primer lugar accedemos al navegador de objetos y creamos un grupo de registros, al crearlo nos aparecerá una pantalla para que especifiquemos si el Record Group será estático (en cuyo caso la creación será muy parecida a la de un List Item con valores introducidos manualmente) o de consulta, escogeremos dinámico e introduciremos una sentencia SQL como la que aparece en la Imagen 39.

Imagen 39.



Puede ver que lo que extraemos de la tabla Empleado para crear el Record Group son las columnas COD_EMPLEADO y NOMBRE, pulsaremos aceptar y se habrá creado el grupo de registros en el navegador de objetos, acceda a las propiedades de este y vea como aparecen dentro de Forms en la Imagen 40.

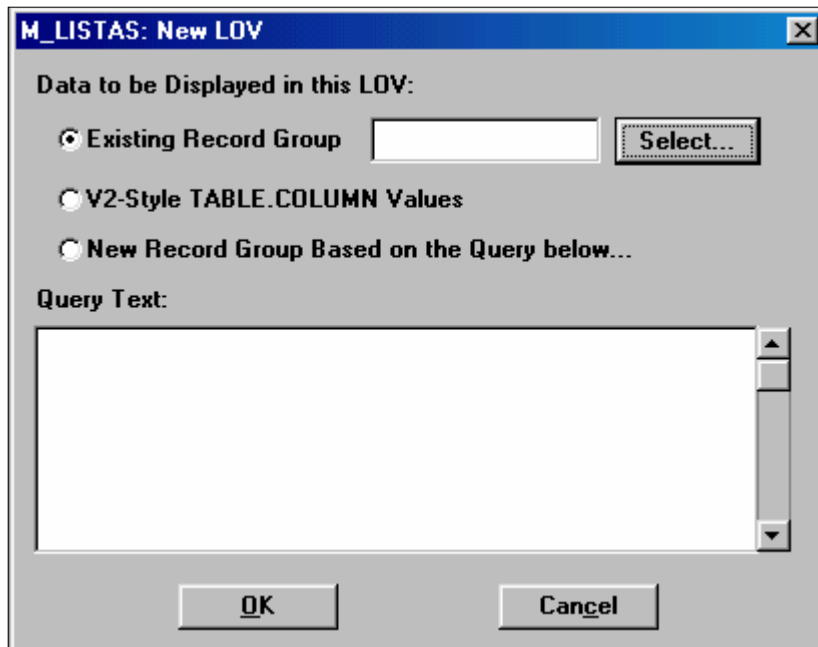
Imagen 40.



Como ve Forms ha creado campos idénticos a los de la BBDD y en la columna de propiedades puede ver incluso la sentencia select antes escrita y que el grupo de registros esta basado en una consulta y no es estático, ahora podría usted modificar los valores de estas propiedades y cambiar a estático, esto puede ser muy útil cuando se quiera crear un grupo de campos iguales a los de una BBDD ya que no tendríamos que introducir estos manualmente, bastaría con hacer la consulta y después pasar a estático donde podría modificar longitudes, tipos de los datos, etc.

A continuación vamos a crear la lista de valores, para ello vuelva a acceder al navegador de objetos, deberá aparecerle una ventana como la de la Imagen 41.

Imagen 41.



Como puede ver se le da escoger entre tres opciones de las que nos interesa verdaderamente la primera que es la que creara la lista Lov basándola en un Record Group ya existente, pero observe que podríamos crear en esta misma ventana un Record Group activando la tercera opción y tecleando allí una consulta SQL, obteniendo exactamente el mismo resultado que creando primero el Record Group. Pulse la primera opción y a continuación el botón Select..., se le mostrara una ventana en donde le da a escoger entre los Records Groups existentes, en nuestro caso y como ve en la Imagen 42 escogemos el RECORD_GROUP2.



Imagen 42.

Pulsaremos aceptar y la lista Lov estará creada. Ahora llega el momento de ver para que sirve todo esto realmente. Tenemos un grupo de registros que extraen datos de la tabla Empleado de nuestra base de datos y recalamos que no hace nada mas, el grupo de registros es una especie de puente que trae los datos de las columnas seleccionadas hasta nuestra maquina local y crea lo que podríamos llamar una tabla temporal con esos datos de tal manera que nuestra lista Lov que es la interfaz gráfica entre esa tabla temporal y la aplicación solo mostrara datos contenidos en esa tabla al usuario.

Usted se preguntara ¿y como le muestra esos datos al usuario?, la respuesta es utilizando tantos Text Item como columnas del grupo de registros queramos mostrar en el lienzo de la aplicación.

Cree a continuación un canvas e inserte en el dos Text Item, a continuación acceda a las propiedades de ambos ítem y en el apartado Varios o Misceláneo podrá ver una propiedad que es LOV, que lógicamente sirve para asociar ese ítem a una Lov previamente creada, asíciela a la lista que ya existe como ve en la Imagen 43

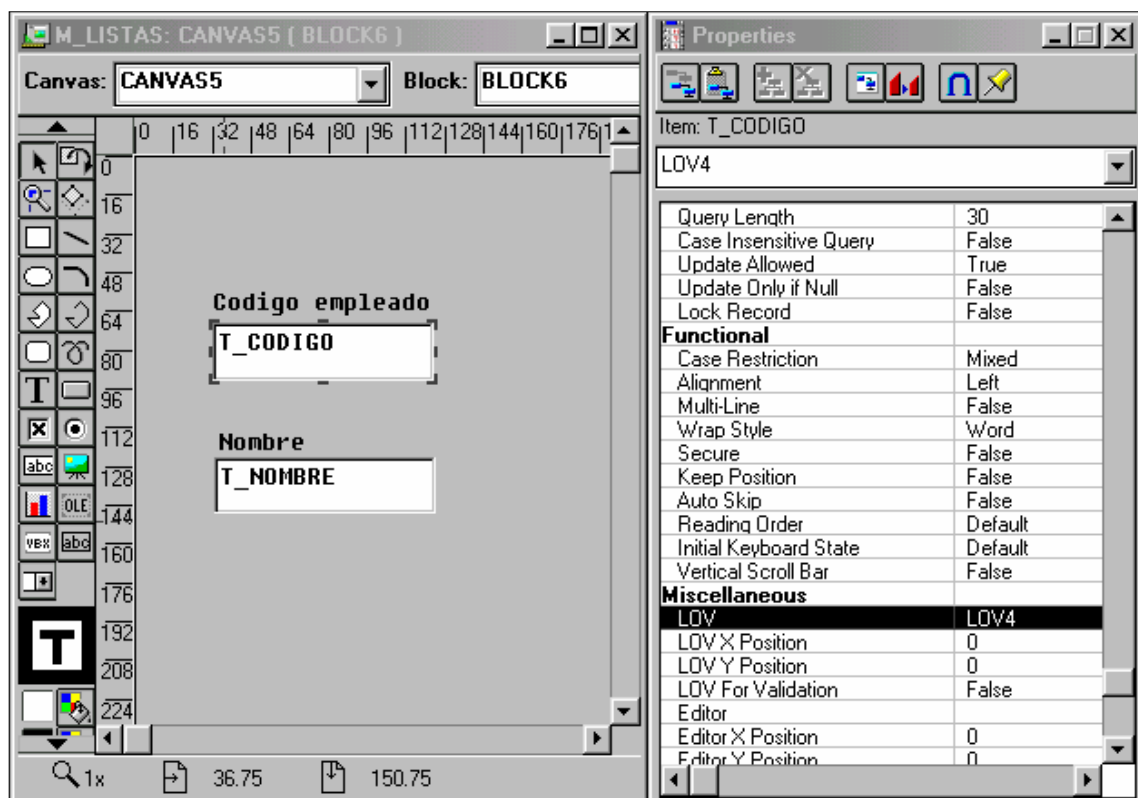
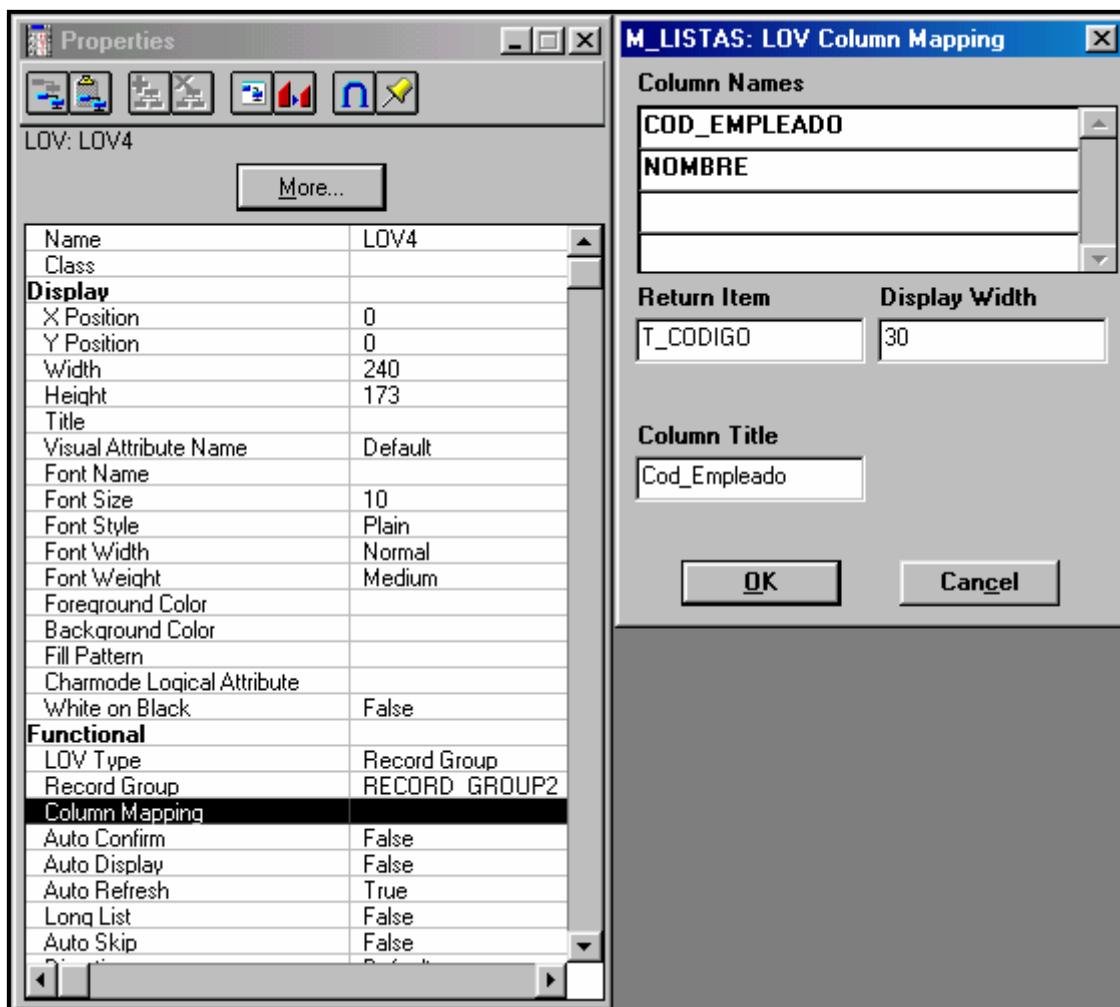


Imagen 43.

Repita el proceso con el otro Text Item.

Ya tenemos un Record Group, una lista Lov y dos Text Item preparados para recibir los datos de la lista Lov, debemos ahora decirle a la lista Lov donde debe insertar el contenido que ha recibido del Record Group, para ello acceda a las propiedades de la lista Lov en el apartado Funcional, observe esto mismo en la Imagen 44.

Imagen 44.

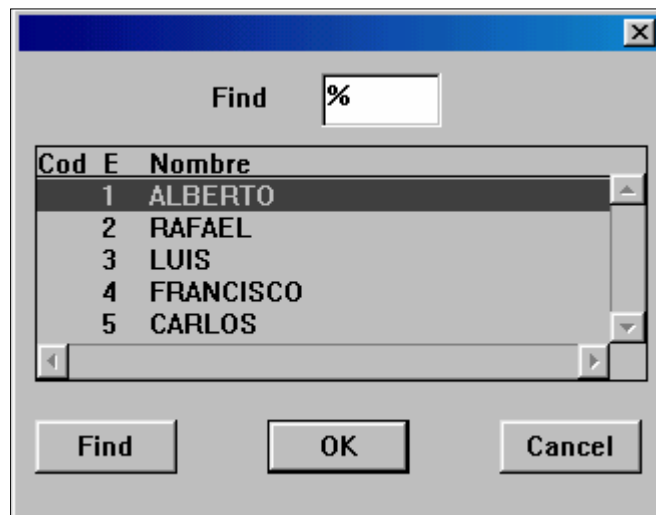


Como ve en ella propiedad mapeado de columnas y tras pulsar el botón Mas..., nos disponemos a indicarle a la lista Lov en que items debe ir incluyendo los valores, en este caso vea como se ha asociado la columna COD_EMPLEADO con el Text Item T_CODIGO. No es necesario que aparezcan todas las columnas si no nos interesa, para ello bastara con indicar en el cuadro de texto Ancho de visualización que la longitud es 0 con lo que el valor no aparecerá en el ítem, o también podemos simplemente no asociarlo a ningún Item.

Ya casi hemos terminado pero falta un detalle importante, hasta ahora solo hemos creado las especificaciones del grupo de registros, la lista Lov y unos ítems, además hemos indicado a la lista que ítems son los que debe utilizar, pero si compila la aplicación vera que no sucede nada. Es importante saber que existen dos métodos para que una lista Lov “entre en acción”:

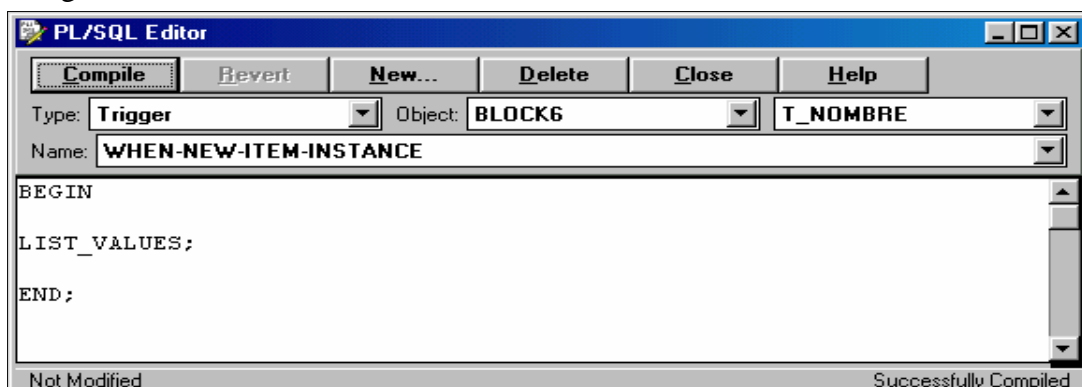
- a) Acuda a las propiedades de la lista de nuevo al apartado Funcional y establezca a true la propiedad Mostrar Automáticamente o Auto Display y compile la aplicación, vera como al situar el cursor sobre el ítem asociado al primer campo establecido en la propiedad Mapa de columnas de la lista Lov aparecerá una ventana modal que le obligara a escoger un registro o cancelar, Debe aparecer una ventana similar a la de la Imagen 45.

Imagen 45.



- b) El segundo método es utilizar el trigger WHEN_NEW_ITEM_INSTANCE, este disparador se ejecuta siempre que un usuario navegue dentro de un ítem y es aplicable a todos los ítems de Forms. Asociaremos este trigger por ejemplo al ítem T_NOMBRE incluyendo la línea de código de la Imagen 46.

Imagen 46.



El Built in LIST_VALUES lo que hace es arrancar la lista de valores asociada a ese ítem.

Ambos métodos producen idéntico resultado pero existe una diferencia sutil entre ellos, utilizando la propiedad Mostrar Automáticamente la lista de valores solo se activara al navegar sobre el ítem que esta asociado a la primera columna del grupo de registros mientras que usando el Built In LIST_VALUES usted puede extender esa posibilidad a todos los ítems que precise.

Para finalizar con los Record Groups y listas Lows diremos que la tabla temporal que se genera en la maquina cliente es actualizada cada vez que el usuario navegue dentro del ítem que esta asociado a la lista que extrae datos de un Record Group de consulta. Por otra parte también debemos saber que el limite de espacio de esa tabla temporal viene marcado por el limite de la memoria en el equipo cliente, es conveniente tener esto en cuenta pues un Record Group de una tabla excesivamente grande podría llegar a ralentizar en exceso el ritmo de una aplicación si esta debe copiar una tabla de varios miles de registros si las comunicaciones no fuesen muy buenas y la maquina local no fuese muy potente, en estos casos deberíamos buscar soluciones alternativas.

*Aunque tratamos con anterioridad los List Item explicamos a continuación una forma de cargar un List Item con los datos obtenidos en un record group, para ello inserte un List Item y cree un Record Group, genere el trigger WHEN_NEW_FORM_INSTANCE a nivel de modulo y asocie a este el siguiente código PL/SQL:

```
DECLARE
```

```
    NUM NUMBER;  
BEGIN
```

```
--RG_1 seria el grupo de registros y L_1 el list item que los va a recibir
```

```
    NUM := POPULATE_GROUP('RG_1');  
    POPULATE_LIST('L_1','RG_1');  
END;
```

La función POPULATE_GROUP prepara el registro para ser insertado y devuelve un NUMBER, cogerá el registro recibido y calculara si este puede ser insertado en el List Item

El procedimiento POPULATE_LIST inserta los registros recuperados del Record Group en el List Item.

-Propiedades mas importantes:

Tipo de Lov: estático o dinámico.

Record Group: origen de los datos

Columna: ancho, alto, etc.

Autodisplay: visualización automática de la lista

Autorefresh: cuando se modifique el record group se actualizan sus datos

Autoskip: al cerrar la lista guarda el elemento y salta al siguiente.

-Built In mas importantes:

FIND_LOV('lista'); devuelve el identificador interno en formato number

SHOW_LOV('Lista',X,Y); posiciona la lista

GET_LOV_PROPERTY

SET_LOV_PROPERTY

9.5-ALERTAS:

En muchas ocasiones se habrá encontrado con una pequeña ventana que le informa sobre algún error que se ha producido o le advierte de las consecuencias de llevar a cabo determinada acción, en Forms el tratamiento de estas situaciones recibe el nombre de Alertas.

Las alertas toman el aspecto de una pequeña ventana modal de la que solo se puede salir pulsando alguno de los botones que en ella aparecen, los botones que aparezcan y las consecuencias de pulsarlos dependen del programador de Forms.

Sin duda alguna el epígrafe “clave” de este ítem es el apartado Funcional. Dentro de el veremos que existen tres tipos de alertas:

-Parar (stop)

-Cuidado (warning)

-Notas

Una alerta podrá tener hasta tres botones visualizándose solo aquellos que hayan sido etiquetados, cada botón tendrá un valor que será el que devuelva después de ser pulsado. Tendremos que definir también el mensaje o texto que aparecerá en la alerta, dicho mensaje procuraremos que sea discreto y no culpabilizador con respecto al usuario, cuestión esta muy descuidada por los programadores y que ha

producido mas de una discusión entre cliente y desarrolladores, debe evitar mensajes como: “La aplicación se ha cerrado porque usted pulsó el botón que no debía”.

Veamos paso a paso como crear una alerta:

En primer lugar diríjase al navegador e instancie una nueva alerta, después cree un canvas con un botón y un display ítem.

Imagen 47.

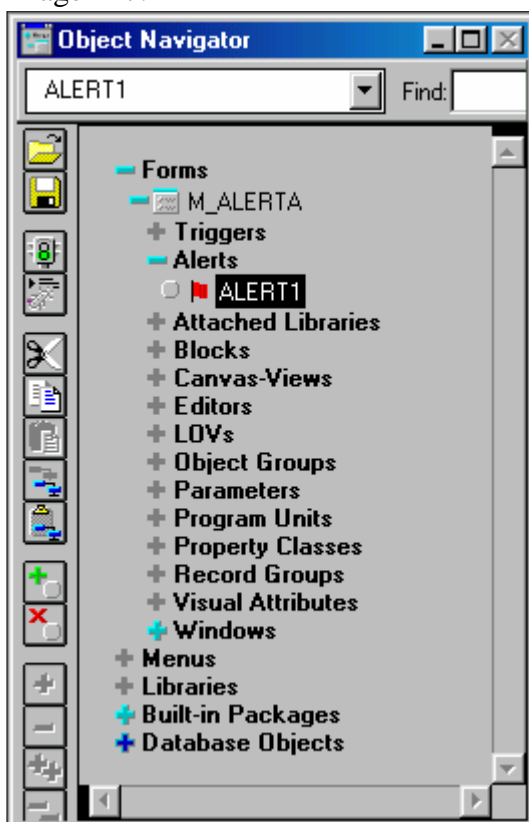
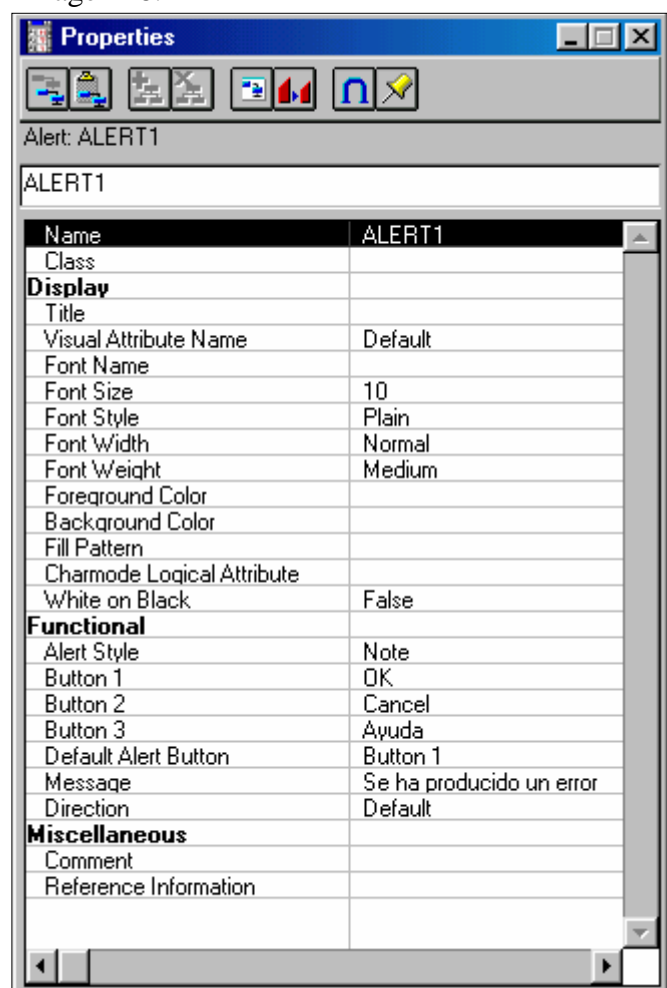


Imagen 48.



Acceda ahora a las propiedades de la alerta creada tal y como se muestra en la Imagen 48 y fíjese bien en el apartado funcional pues en el aparece toda la información que podrá mostrar la ventana de alerta, analicemos en detalle cada una.

-Alert style: mostrara una alerta informativa o de nota, el usuario podrá distinguir de que tipo de alerta se trata por dos motivos, en primer lugar por el icono que aparezca en dicha alerta y en segundo lugar porque se le habrá indicado que ese icono pertenece a una alerta informativa, el icono en cuestión es el que ve en la Imagen 49.



Imagen 49.

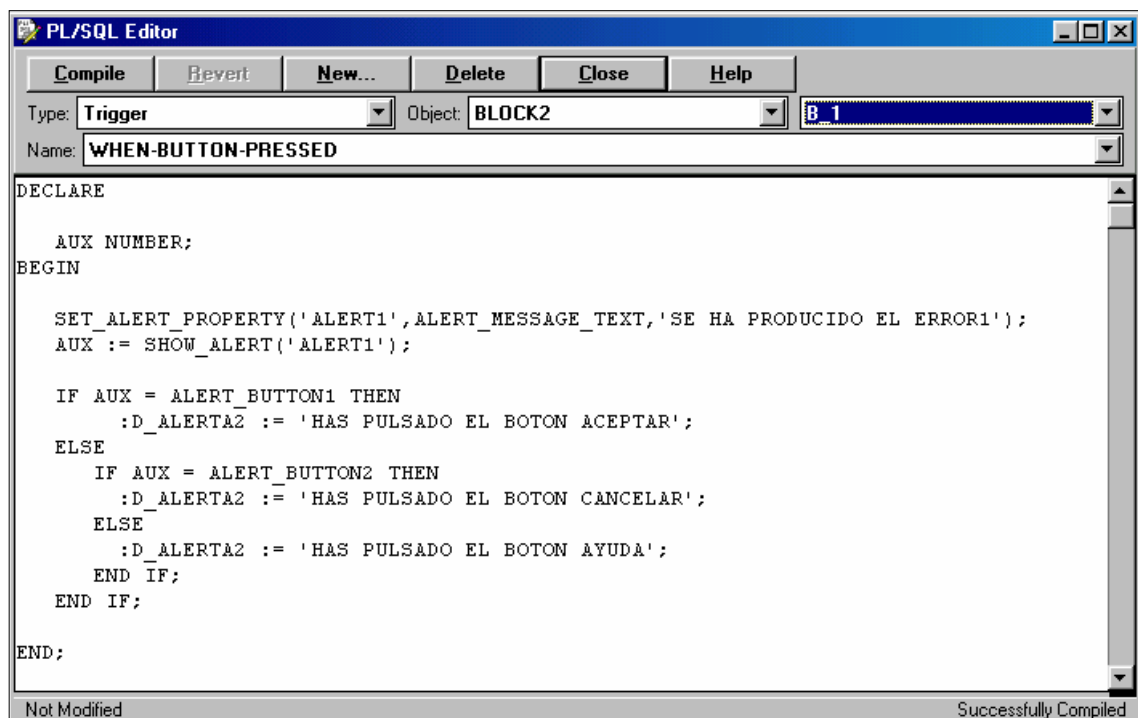
-Button1, Button2, Button3: en estas propiedades escribirá usted la etiqueta que aparecerá en cada uno de los botones de la alerta, si no escribiese usted etiqueta alguna dicho botón no aparecería.

-Default Alert Button: en esta propiedad se establece el botón que recibirá el foco por defecto al aparecer la alerta.

-Message: será el texto que se podrá leer en la ventana de la alerta (recuerde; discreto y no culpabilizador).

Una vez que hemos definido el aspecto de la alerta debemos utilizar código PL/SQL para hacer que esta se muestre cuando a nosotros nos interese, para ello asocie el siguiente código al evento WHEN_BUTTON_PRESSED del botón que inserto en el canvas.

Imagen 50.

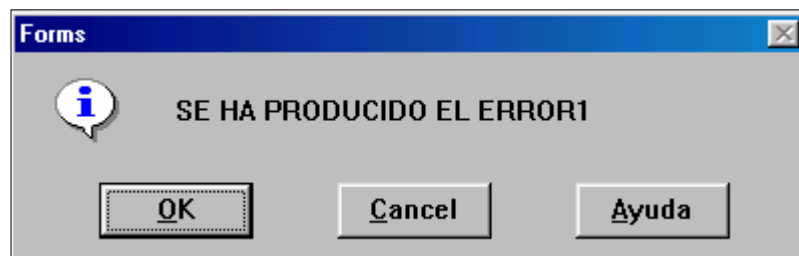


Estudiemos este bloque PL/SQL. Como podrá ver utilizamos un procedimiento y una función, el procedimiento es SET_ALERT_PROPERTY en el que establecemos un nuevo mensaje de error para la alerta 'SE HA PRODUCIDO EL ERROR1', con lo que usted puede ver que ese mensaje por defecto de las propiedades se puede modificar dinámicamente.

La siguiente función es la que verdaderamente muestra la alerta y es SHOW_ALERT, esta función devuelve un valor de tipo Number por lo que en la sección declarativa hemos creado la variable AUX de este tipo, en primer lugar por el orden de operadores se ejecuta lo que se encuentre al lado derecho del signo igual para después realizar la asignación, es por esto que en primer lugar se muestra la alerta, en segundo lugar y al tratarse de una ventana modal hasta que el usuario no pulse uno de los botones no podrá quitar el foco de la ventana, inmediatamente después de esto la variable AUX recibe en forma de constante simbólica el valor del botón pulsado que puede ser ALERT_BUTTON1, ALERT_BUTTON2 o ALERT_BUTTON3, de esta manera podemos saber que botón se ha pulsado y actuar en consecuencia, en el programa puede ver que simplemente se muestra un mensaje de texto en el Display Item indicando que botón se pulso pero usted podría cerrar la aplicación drásticamente, abrir otro modulo o formulario, etc.

El resultado de la compilación del programa anterior debe mostrar un aspecto similar a la Imagen 51.

Imagen 51.



9.6-TIMER:

Un timer es un objeto no gráfico lo que quiere decir que no podremos interactuar con ningún control para crearlo, su creación y ejecución se realizan totalmente a través de código PL/SQL.

Podríamos definir un timer como un reloj o cronometro que al cabo de un tiempo por nosotros indicado ejecutara las instrucciones que se le asocien.

Los timers se declaran a nivel de Form y funcionan de la siguiente manera:

CREACION → DESCUENTO DE TIEMPO → TRIGGER
(WHEN_TIMER_EXPIRED)

Para crear un timer debemos especificar como mínimo tres cosas:

- Nombre del Timer.
- Duración en milisegundos (1 segundo = 1000 milisegundos).
- Si es repetitivo o no.

El Built In para la creación del Timer es:

CREATE_TIMER('Nombre_del_timer', Tiempo, Repeat/No_Repeat);

Cuando el tiempo de vida del timer llegue a 0 se ejecutara el Trigger WHEN_TIMER_EXPIRED, pero ¡ojo! solo existe un WHEN_TIMER_EXPIRED en todo el Form de tal modo que si existiesen varios Timers debo saber cual ha disparado el Trigger para actuar en consecuencia.

Veamos un ejemplo:

En primer lugar y dado que los Timer se crean a nivel de Modulo haremos que se creen dos Timer nada mas arrancar la aplicación, para ello utilizaremos el disparador WHEN_NEW_FORM_INSTANCE y asociaremos a este el código que aparece en la Imagen 53 para crear los timer, observe en la Imagen 52 como se han dispuesto los elementos de la aplicación en el navegador de objetos.

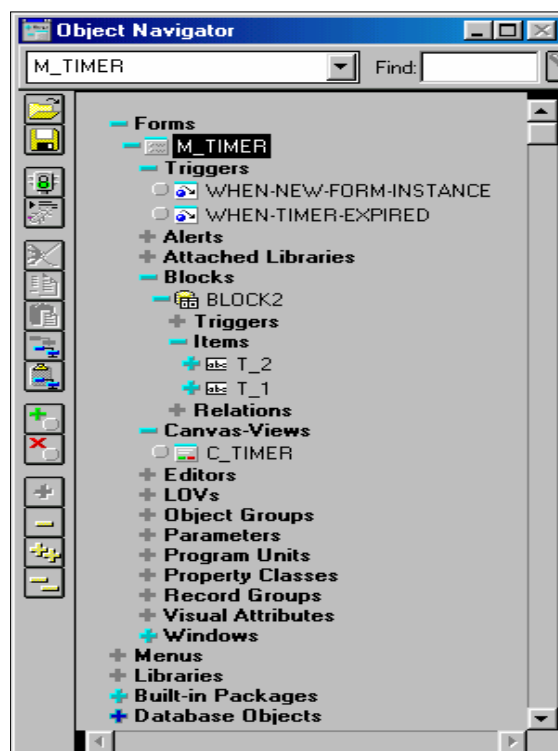
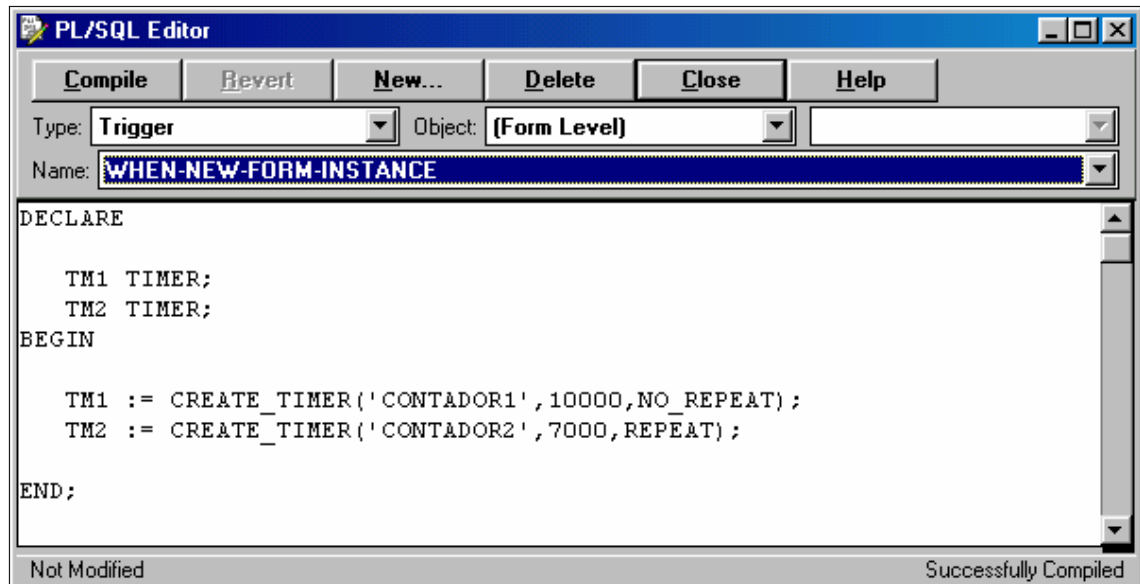


Imagen52.

Imagen 53.



En este código puede apreciar como creamos dos Timer, para ello y debido a que `CREATE_TIMER` es una función hemos creado dos variables de tipo Timer que son tipos de datos existentes en Forms que nos facilitan el uso de este objeto.

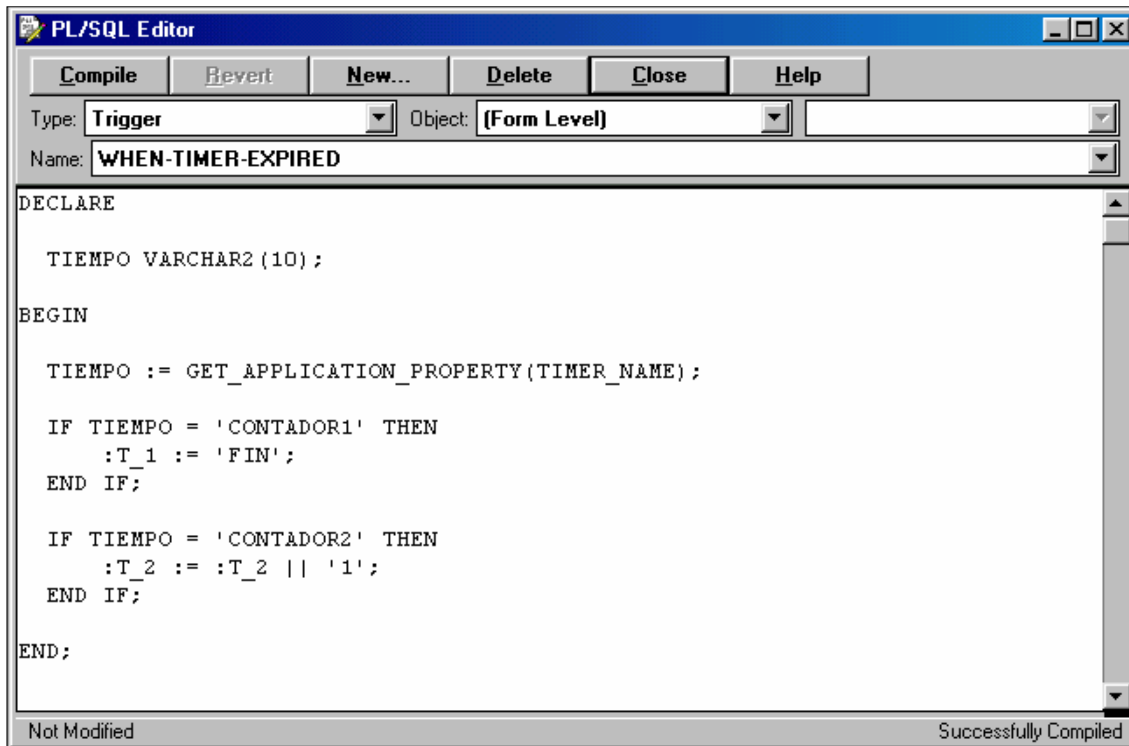
El primer contador llamado `CONTADOR1` tardara en expirar desde el mismo momento en que es creado 10000 milisegundos o lo que es lo mismo 10 segundos y no comenzara de nuevo pues se ha establecido así con `NO_REPEAT`.

El segundo contador se llama `CONTADOR2` y tardara en expirar 7 segundos, tras los cuales comenzara de nuevo debido a la opción `REPEAT`.

Una vez creados los timer podemos o no asociar código a los mismos para que se ejecute algo una vez que su tiempo haya expirado, observe la Imagen 52 y vea como debajo del Trigger `WHEN_NEW_FORM_INSTANCE` hemos creado otro; `WHEN_TIMER_EXPIRED`, cree usted también este trigger e inserte en el canvas dos Text Item llamados por ejemplo `T_1` y `T_2`.

Vayamos ahora a ver el código asociado al disparador `WHEN_TIMER_EXPIRED` tal y como se ve en la Imagen 54.

Imagen 54.



En primer lugar declaramos una variable llamada TIEMPO del tipo VARCHAR2 y con una longitud 10, mas que suficiente para albergar el nombre del Timer que acaba de expirar.

En la sección de instrucciones lo primero que hacemos es utilizar la función GET_APLICACION_PROPERTY para averiguar el nombre del Timer que ha expirado y a continuación con una instrucción condicional realizamos una acción u otra en función del timer, en el caso del Timer CONTADOR1 simplemente mostramos la cadena FIN en el Text Item T_1, en el caso de CONTADOR2 concatenamos el numero 1 cada vez que expire el Timer ya que este si es repetitivo. Compile la aplicación y observe el resultado.

-Built In mas importantes:

```
TIMER := CREATE_TIMER('NOMBRE',DURACION,REPEAT/NO_..);
TIMER := FIND_TIMER('NOMBRE');
Si existe el timer devuelve el nombre sino devolverá NULL
SET_TIMER('NOMBRE',NUEVA_DURACION,NUEVO_ESTADO,
          REPEAT/NO_REPEAT);
```


9.7-EDITORES:

Un editor tal y como lo entiende Forms en el “interior” de una aplicación es un pequeño cuadro de texto que sirve para poder escribir de forma mas cómoda que el Text Item en el que queremos escribir esa información.

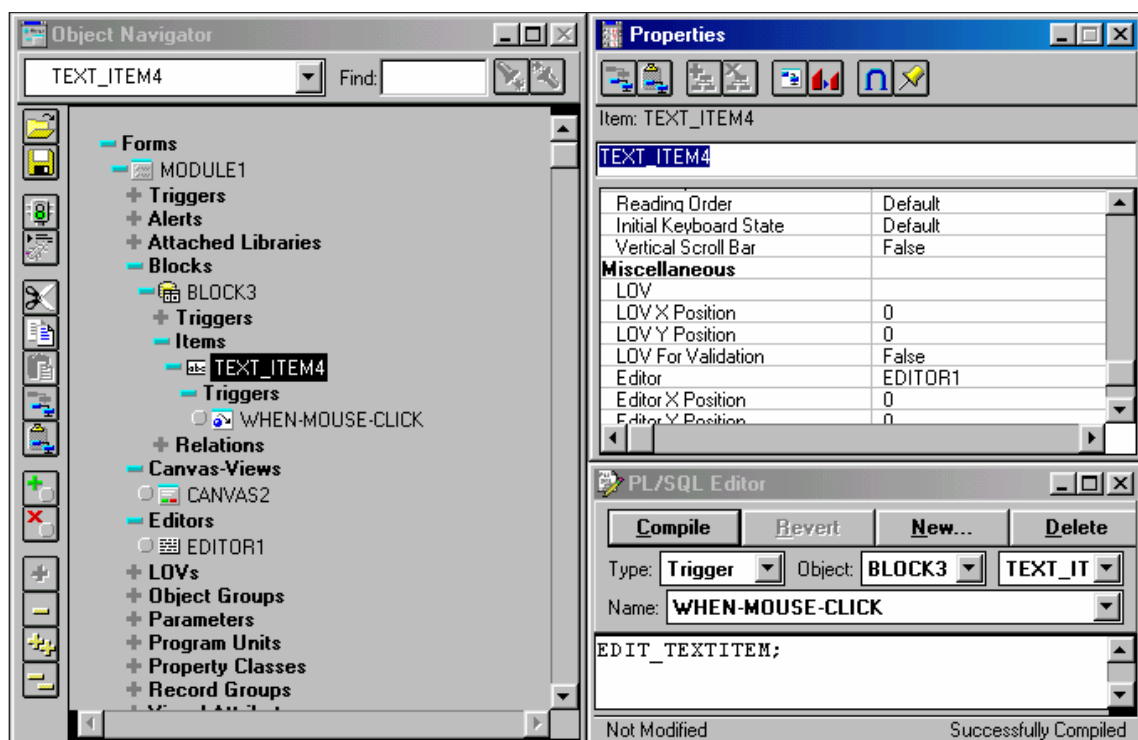
Normalmente lo utilizaremos cuando haya muchos controles en el canvas y agrandar un Text Item sea un lujo que no podamos permitirnos.

Existen tres tipos de editores:

- a) Formato normal y simple.
- c) Del sistema similar al Notepad de Microsoft Windows.
- d) Personalizado o creado por el usuario.

Para crear un editor de este tipo en primer lugar se instancia el mismo desde el navegador de objetos y en segundo lugar se asocia al Text Item correspondiente, puede ver el proceso completo en la Imagen 55 y observar que es muy simple, solo debemos añadir una pequeña línea de código al evento WHEN_MOUSE_CLICK del Text Item que nos interese para activar el editor, comprobara que en las propiedades del Item es donde indicamos que editor se va a asociar al Text Item.

Imagen 54.



Es importante tener en cuenta que el numero máximo de caracteres que podamos escribir será el mismo que hayamos indicado para el ítem, además en el apartado funcional del ítem debemos especificar que la propiedad Multilinea y Scrollbar sean True para permitir una visualización mas cómoda. Ajuste las propiedades como hemos indicado y compile la aplicación para ver el resultado.

10-COMUNICACIÓN ENTRE DIVERSOS ELEMENTOS:

Hasta ahora todo lo que hemos visto han sido elementos de Forms de manera independiente; como un canvas, un check box, un radio button, un modulo, etc., pero como ya habrá podido imaginar las aplicaciones de Forms no generan elementos para ser tratados de forma aislada unos de otros, si fuese así Forms seria una herramienta muy limitada, por supuesto Oracle ha creado sistemas para poder comunicar las herramientas no solo de Forms si no incluso entre aplicaciones distintas dentro del propio Developer 2000. Un elemento fundamental en esa comunicación son los parámetros que permitirán que pasemos información de un modulo a otro o desde un bloque a otro para poder averiguar por ejemplo los asientos disponibles en una base de datos de aviones con un modelo de avión pasado como parámetro desde un modulo por poner un ejemplo.

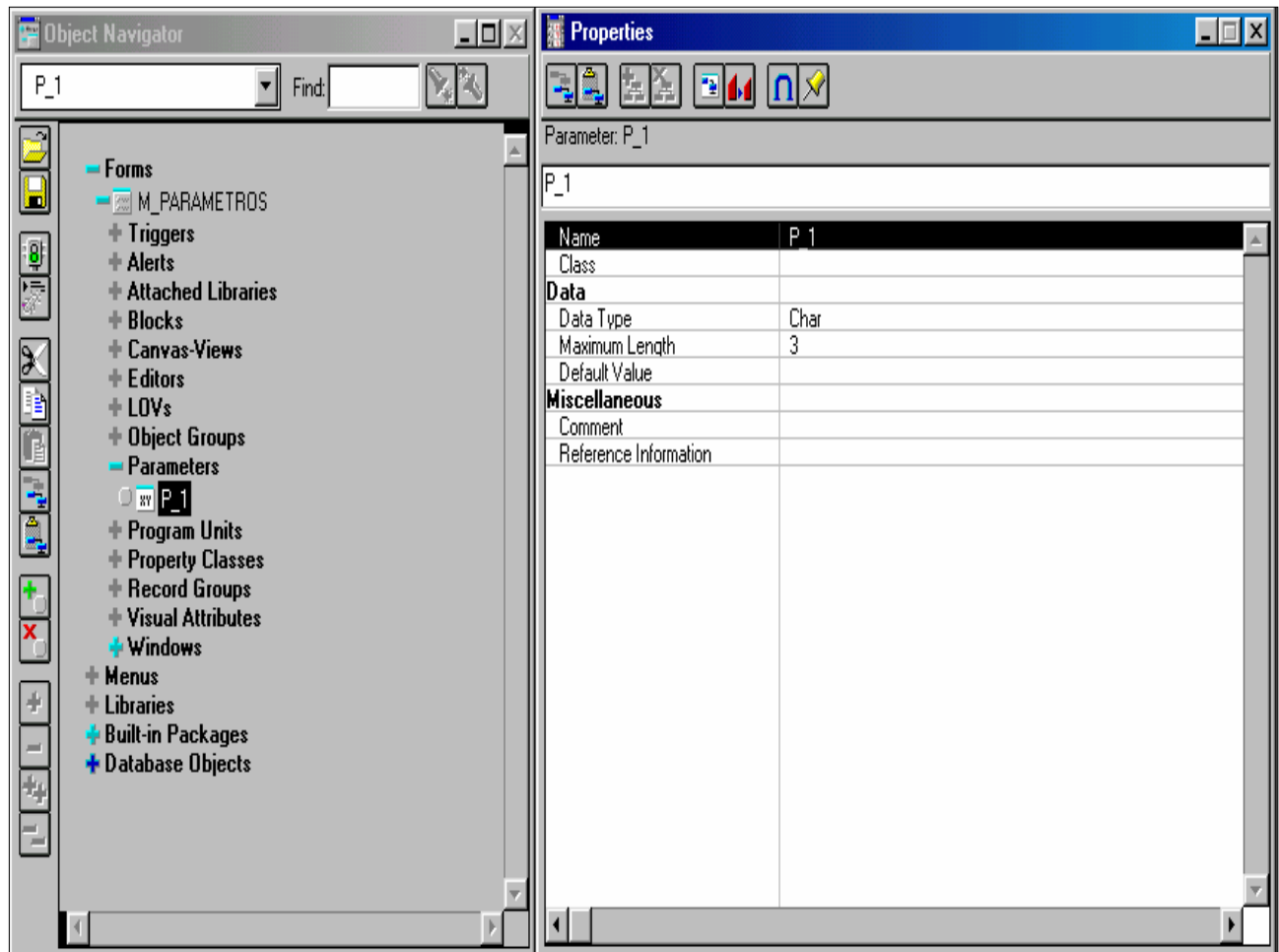
Veamos a continuación como maneja Forms esa comunicación entre elementos dentro de una misma aplicación.

10.1-PARAMETROS 1:

Debemos ver un parámetro como un testigo que contiene una información determinada, información de la que los elementos de Forms van a hacer uso si nosotros así lo especificamos, en resumen un parámetro no es mas que una variable global.

Acceda al navegador de objetos y vea como encuentra un elemento llamado Parámetros, es aquí donde se instancia un parámetro, vea como lo hemos hecho nosotros en la Imagen 55, hemos creado el parámetro P_1 de tipo char y con una longitud de 3 caracteres que utilizaremos para acceder a un departamento o a otro según el contenido del parámetro.

Imagen 55.



Observe también que podríamos haber definido un valor inicial para P_1.

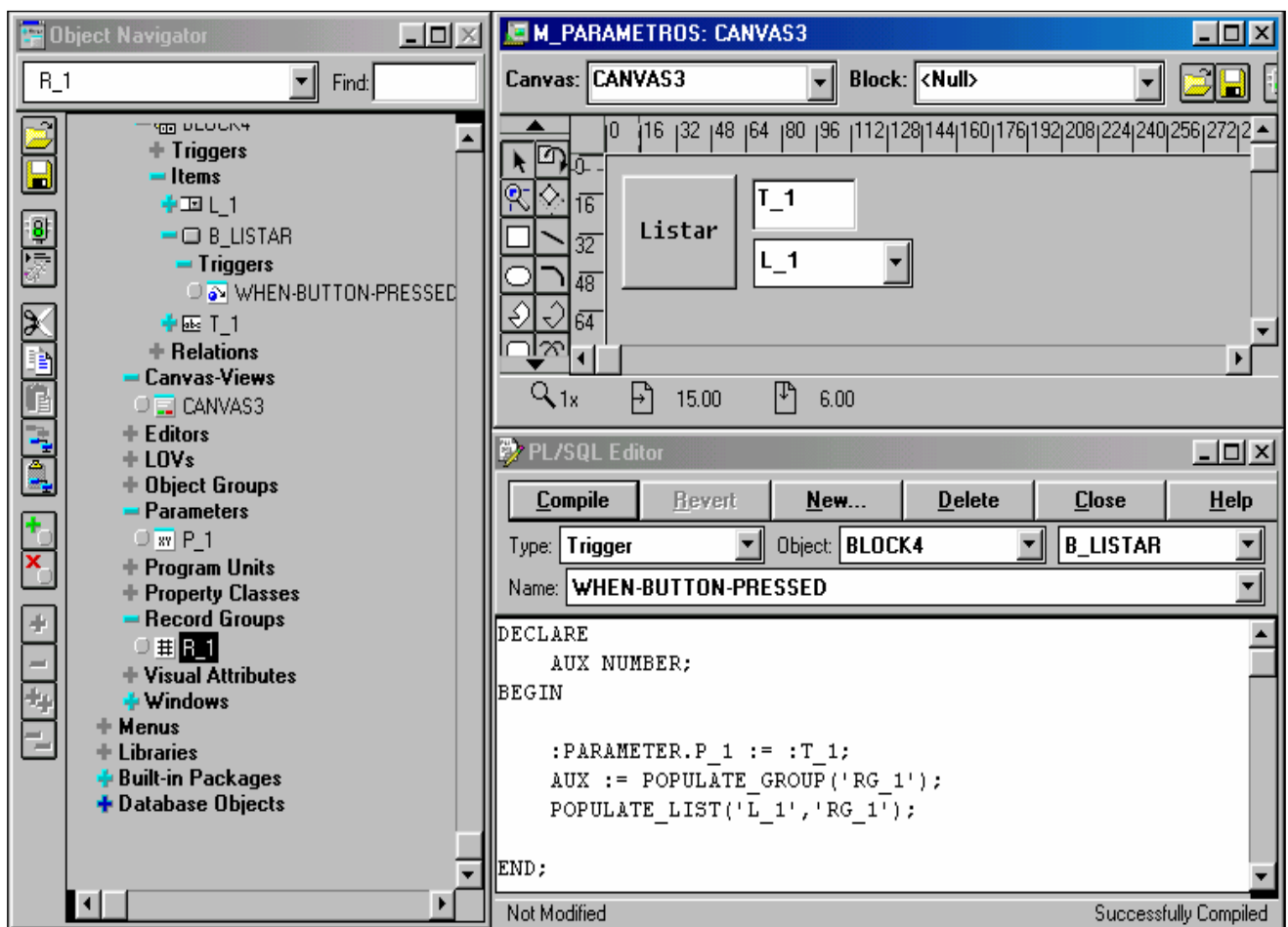
Ahora para ver la utilidad del parámetro utilizaremos un List Item y un Record Group, Record Group que será cargado en función del valor contenido en el parámetro. Para realizar esto cree un grupo de registros basado en una consulta con la siguiente instrucción sql:

```
SELECT NOMBRE, APELLIDO  
FROM TEMPLA  
WHERE DEPT = :PARAMETER.P_1
```

A continuación cree un canvas con un botón, un Text Item y un List Item de tipo Pop List. La intención de la aplicación es que cuando al usuario le aparezca la pantalla de la aplicación escriba un código de departamento en el Text Item, pulse el botón y que en el List Item aparezcan los nombres de los empleados del departamento que pulsó.

Para que la aplicación funcione es necesario introducir el siguiente código PL/SQL en el disparador WHEN_BUTTON_PRESSED del botón, puede ver la disposición de los elementos y del código en la Imagen 56.

Imagen 56.



Como puede ver nos referimos al parámetro anteponiendo el prefijo :PARAMETER. y lo que hacemos con el es asociarlo a lo que el usuario escriba en el cuadro de texto T_1, después extraemos la longitud del registro con POPULATE_GROUP y a continuación llenamos el List Item L_1.

Podría usted pensar que esto también se podría haber hecho extrayendo del Text Item T_1 lo tecleado por el usuario pero piense que eso supondría una labor de concatenación y de control sobre lo introducido en el ítem excesiva si tenemos en cuenta que podemos utilizar parámetros. Compile la aplicación y observe el resultado, vera que en seguida le vienen a la cabeza ideas para futuras aplicaciones vista la facilidad de carga de registros en un ítem.

10.2-RELACION ENTRE BLOQUES:

Forms dispone de herramientas para evitarnos escribir código PL/SQL, la relación entre bloques es una de ellas, controlar la salida por pantalla de datos surgidos de un join entre dos tablas en un lenguaje de tercera generación siempre ha sido algo complicado, esto es precisamente lo que viene a hacer Forms por nosotros en la llamada relación entre bloques, para comprobar su funcionamiento utilizaremos las tablas EMP y DEPT presentes como tablas de ejemplo en los gestores Oracle.

En primer lugar crearemos dos bloques tabla, uno basado en EMP y el otro en DEPT, a continuación acceda al elemento del navegador de objetos Relaciones en cualquiera de los dos bloques y genere una nueva relación, una vez haga esto deténgase ante la pantalla que le aparece que será como la de la Imagen 58.

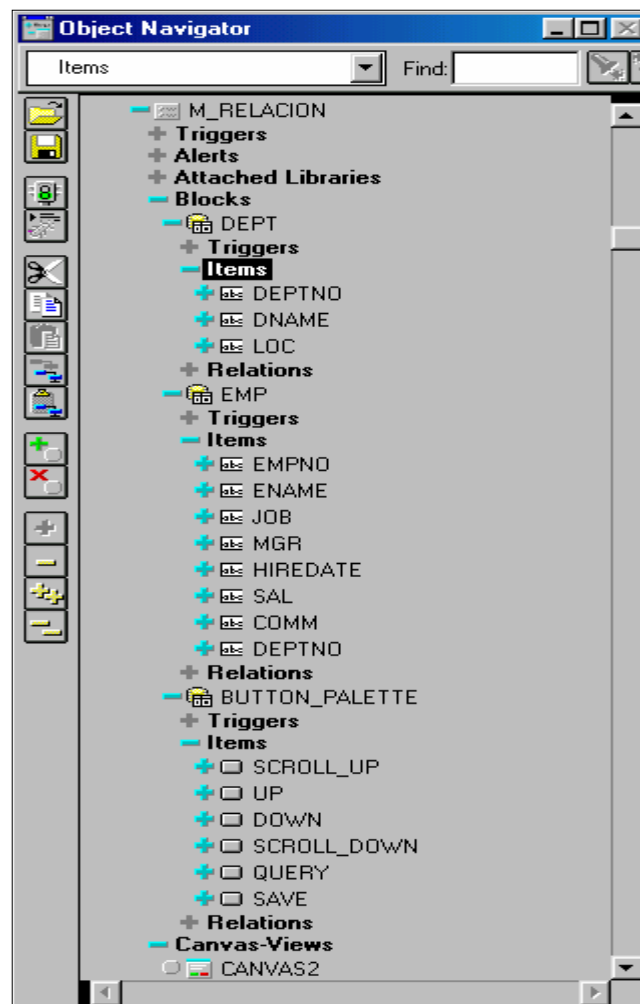


Imagen 57.

Imagen 58.

M_RELACION: New Relation

Relation Name:

Master Block:

Detail Block:

Master Deletes: Coordination:

☐ Cascading ☐ Deferred

☐ Isolated ☒ Auto-Query

☒ Non-Isolated ☐ Prevent Masterless Operation

Join Condition:

Estudiemus lo que nos aparece en esta ventana pues son datos importantes, comencemos por la parte inferior donde hemos escrito la condición de unión.

En el apartado Join Condition hemos establecido la relación que queremos entre los dos bloques; el bloque maestro DEPT y el bloque detalle EMP, al escribir DEPT.DEPTNO = EMP.DEPTNO lo que estamos haciendo es informar a Forms de que vamos a detallar la información de cada departamento (DEPT.DEPTNO) en el bloque detalle (EMP.DEPTNO) de tal modo que por cada departamento queremos que nos aparezcan los campos relacionados a través de esta instrucción en el bloque detalle, la sintaxis siempre será igual:

BLOQUE_MAESTRO.COLUMNNA = BLOQUE_DETALLE.COLUMNNA

En este caso da la casualidad de que el maestro contiene la clave principal y el detalle la clave ajena pero esto no tiene porque ser así siempre, crearemos bloques maestro y detalle basándolos en las claves que nos interesen de tal modo que el maestro podría no ser el que tuviese la clave principal sino una clave ajena y el detalle la clave principal, ocurre sin embargo que en la mayoría de las ocasiones encontraremos casos

como el del ejemplo. Es conveniente que estudie la estructura de ambas tablas para comprender mejor la relación entre bloques.

Continuemos con las opciones que nos muestra la ventana de relación, en la parte izquierda puede ver las opciones de borrado, detallémoslas.

- Non Isolate: no se borrara ningún registro maestro mientras existan relacionados en el detalle
- Isolate: permite borrar registros maestros aunque existan relaciones en el detalle.
- Cascading: borra primero los relacionados en el detalle y después el maestro.

El siguiente apartado es coordinación (visual) o como van a aparecer los registros en el detalle.

- Auto Query: actualiza los bloques inmediatamente
- Si Deferred y Auto Query están desactivados la coordinación es automática, en caso contrario hasta que no haya bloque no aparecen los datos.
- Si Deferred está activado y Auto Query no lo está debo saltar al bloque y ejecutar la query para volver a consultar

Establezca las opciones que hemos especificado en la imagen o cámbielas a tenor de lo visto y pulse aceptar, ahora vuelva al navegador de objetos y observe la gran cantidad de Trigger que Forms ha incluido, vamos a ver que hacen algunos de esos disparadores.

Muchos de los trigger generados ejecutan acciones de borrado y están directamente relacionados con las opciones que hayamos definido a la hora de crear la relación. Dependiendo de lo que hayamos especificado existirán algunos y otros no, puede ver los creados con las opciones de la Imagen 58 en el navegador de objetos que aparece en la Imagen 59.

ON_CLEAR_DETAILS

Aparece en toda relación y se activa al intentar borrar los registros detalle.

ON_POPULATE_DETAILS

Aparece en toda relación, rellena los registros detalle realizando la query
Oportuna

ON_CHECK_DELETE_MASTER

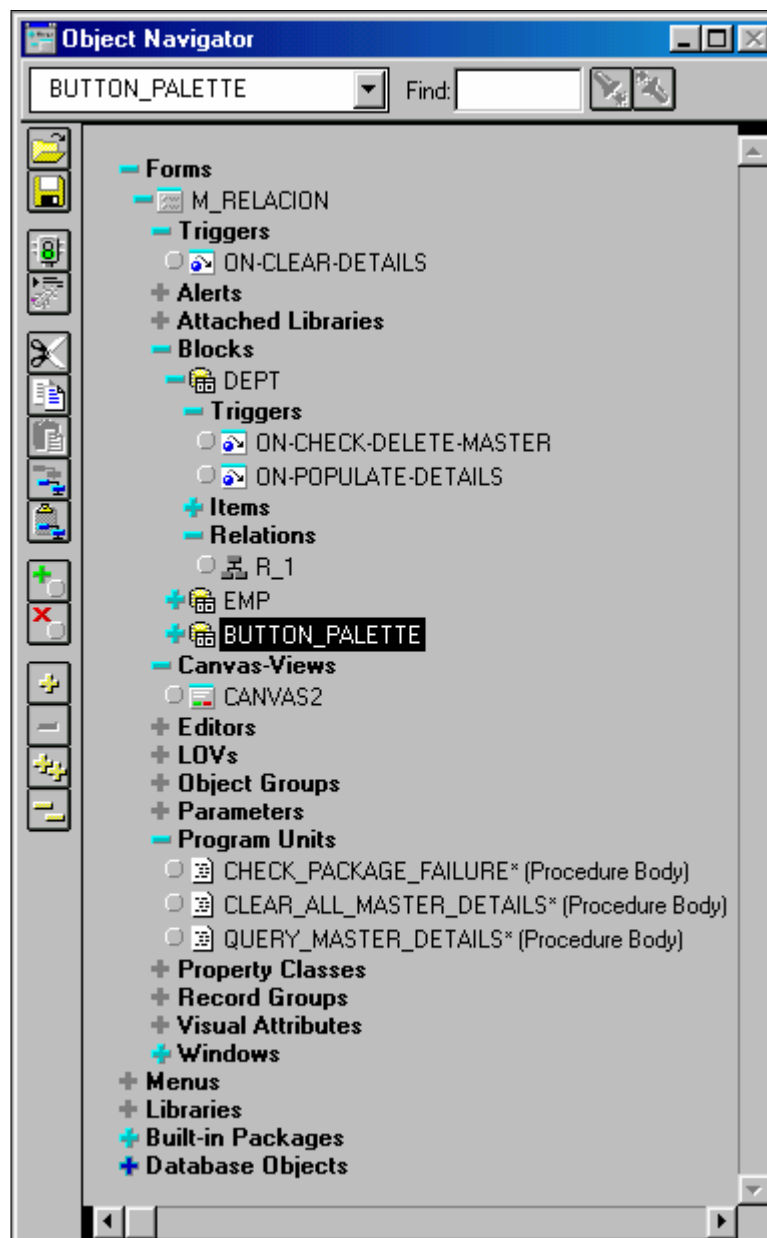
Aparece en una relación No Isolated si existen registros detalle. La existencia de este trigger implica verdaderamente que la opción sea Isolated o no, independientemente de lo especificado en la casilla de verificación.

PRE_DELETE

Solo existe en los tipo cascada, comprueba la relación y si existen registros detalle relacionados elimina estos primero.

Es importante saber que si copiamos un bloque tabla con relaciones se copiaran los triggers pero no las relaciones que hubiésemos establecido, con los consecuentes errores que ello conllevaría.

Imagen 59.



10.3-COMUNICACIÓN ENTRE FORMS:

10.3.1-OPEN FORM:

Existen dos maneras de comunicar módulos o de llamar a módulos concretos.

a) Básica:

```
OPEN_FORM('RUTA_DEL_MODULO');
```

b) Avanzada:

```
OPEN_FORM('RUTA',  
          ACTIVACION,  
          MODOSesion,  
          DATOS,  
          LISTA_DE_PARAMETROS);
```

Activación :

Activate: lleva el foco al modulo (por defecto)

No_activate: no lleva el foco al nuevo modulo

Modosesion:

Sesión: crea una conexión con el gestor para realizar las validaciones del modulo abierto de forma independiente al resto de módulos que hubiese en memoria en ese momento.

No_sesion: el modulo abierto se “une” a la sesión que ya este abierta en la base de datos.(por defecto)

Datos:

NO_SHARED_LIBRARY_DATA: no comparte sus datos con otros módulos.(por defecto)

SHARED_LIBRARY_DATA: comparte sus datos con otros módulos. Este modo crea un área común de datos en memoria para los módulos lo que obliga a la aplicación a gestionar a mas bajo nivel de lo normal, esto implica una carga extra de trabajo para la aplicación.

Lista de parámetros:

Posibles valores que el modulo llamado va a recibir.

10.3.2-NEW FORM:

A diferencia de OPEN_FORM que permite abrir módulos y que estos convivan en pantalla y en memoria NEW_FORM cerrara los módulos existentes para arrancar el modulo llamado con este procedimiento. No obstante NEW_FORM implica que se pregunte al usuario si quiere validar el contenido de los módulos anteriores por si quiere guardar las modificaciones o no, además es conveniente que en el modulo destino incluyamos un EXIT_FORM en un botón por si acaso.

Podemos ejecutar de dos maneras este Built In:

a) Básica:

```
NEW_FORM('RUTA_DEL_MODULO');
```

b) Avanzada:

```
NEW_FORM('RUTA',  
          MODO_ROLLBACK,  
          MODO_QUERY,  
          DATOS,  
          LISTA_PARAMETROS);
```

Modo_rollback:

Full_rollback: ejecuta un rollback completo de la transacción(por defecto)

No_rollback:no restaura y por tanto es como el commit

To_savepoint ejecuta un rollback hasta el ultimo punto de ruptura

Modo_query (se sitúa en el nuevo modulo solo en modo de consulta o no)

No_query_only (por defecto)

Query_only

10.3.3 CALL FORM:

Abre un formulario en formato modal y no permite navegar a otros módulos hasta que se cierre, existen dos formas de manejarlo:

a) Basica:

```
CALL_FORM('RUTA_DEL_MODULO');
```

b) Avanzada

```
CAL_FORM('RUTA',  
          DISPLAY,  
          MENU,  
          MODO_QUERY,  
          LISTA_PARAMETROS);
```

Display:

HIDE : oculta el form llamante

NO_HIDE: no oculta el form llamante

Menú:

NO_REPLACE(por defecto)

DO_REPLACE

Cambia el menú superior del forms que llama o lo mantiene.

10.3.4-RUN PRODUCT:

Run_Product difiere en su funcionalidad con respecto a los built in anteriores en que no solo podrá abrir otro modulo sino cualquier producto incluido en Developer 2000, la diferencia fundamental es la lentitud, no tiene sentido llamar desde un modulo Forms a otro modulo Forms con este built in, pero si quisiésemos llamar a un informe de Reports no existe otra forma.

Su sintaxis es:

```
RUN_PRODUCT('PRODUCTO',  
            'RUTA',  
            COMUNICACIÓN,  
            TIPO_EJECUCION,  
            LOCALIZACION,  
            ,LISTA_PARAMETROS);
```

-Producto:

Reports
Graphics
Books

-Comunicación

SYNCDHRONOUS (foreground o primer plano)
ASYNCHORNOUS(secundo plano o batchgrounnd)

-Tipo de ejecucuion

BATCH: solo se usa en Graphics
RUNTIME: mas rápido, se usa para el resto.

-Localización

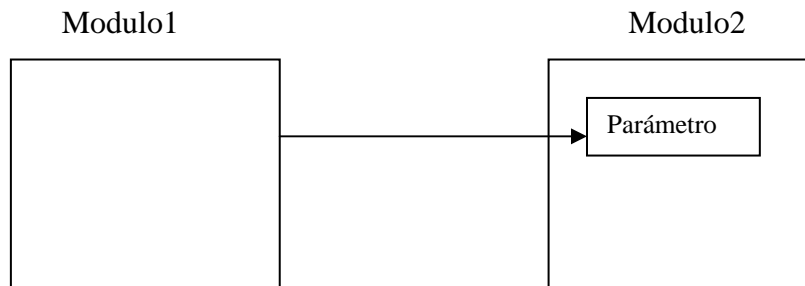
DB: en la base de datos
FILESYSTEM: en soporte físico como disco o cinta

-Lista de parámetros

NULL o NULL, NULL puede variar en función de la versión del producto.

10.4-PARAMETROS 2:

Vamos a estudiar como podemos pasar parámetros entre dos o mas módulos de Forms.



Solo es necesario declarar el parámetro en el modulo que lo va a recibir, lo haremos de la siguiente manera:

- 1º - Creamos una lista de parámetros.
- 2º - Rellenamos dicha lista.
- 3º - Llamamos al modulo.

Este tipo de listas se crean programaticamente no existen objetos visuales que podamos utilizar. Como ejemplo crearemos un canvas e insertaremos en el un botón con la etiqueta Parámetros, asociado al trigger WHEN_BUTTON_PRESSED escribiremos el siguiente código PL/SQL.

```

DECLARE
    LIST_ID PARAMLIST;

BEGIN
    --CREO LA LISTA
    LIST_ID := CREATE_PARAMETER_LIST('LISTAP');

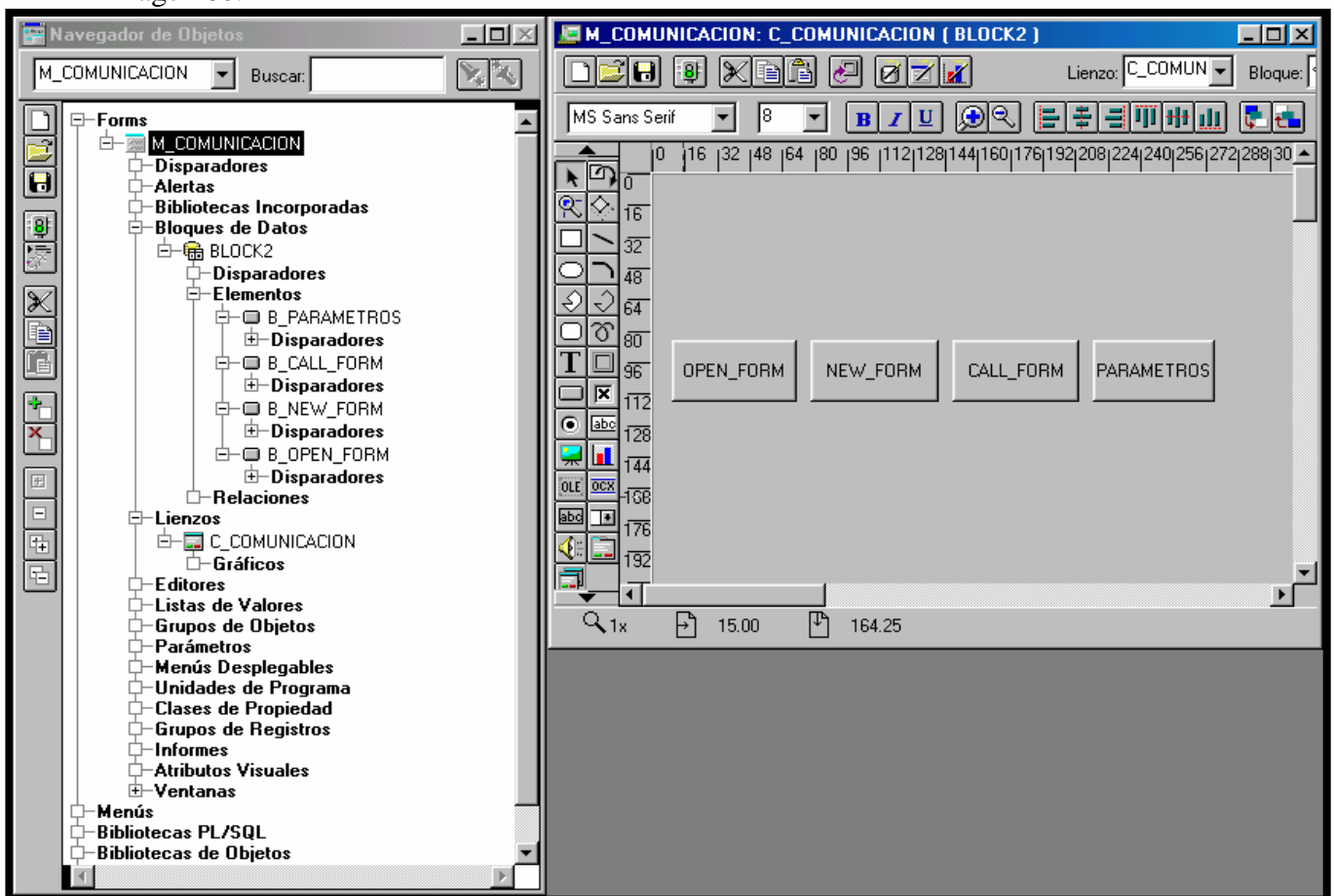
    --AÑADO EL PARAMETRO/OS A LA LISTA, PRIMERO INDICAMOS LA LISTA,
    --DESPUES EL PARAMETRO QUE DEBE ESTAR PREVIAMENTE DECLARADO EN EL
    --MODULO A LLAMAR, EL TIPO DE PARAMETRO UTILIZANDO LA CONSTANTE -
    --SIMBOLICA TEXT_PARAMETER Y EL VALOR EN SI DEL PARAMETRO

    ADD_PARAMETER(LIST_ID, 'P_1', TEXT_PARAMETER, 'e21');

    --LLAMO AL FORM QUE TIENE EL PARAMETRO AÑADIDO
    CALL_FORM( 'D:\Ucon12\Fuentes\M_PARAMETROS',
              HIDE,
              NO_REPLACE,
              NO_QUERY_ONLY,
              LIST_ID
            );
END;
  
```

Le habrá parecido extraño comparando con el resto del tutorial que no hayamos utilizado ni en un ejemplo en este capítulo dedicado a las comunicaciones entre módulos, no se preocupe, no lo hemos hecho porque hemos preferido explicar primero los tipos de comunicación para ahora exponerle un ejercicio con todos ellos unidos de una forma muy sencilla, vea en las siguientes imágenes como desde un mismo modulo llamamos a otros muchos.

Imagen 60.



Observe que cada botón en la Imagen 60 va a realizar una de las tareas de comunicación que hemos estudiado, a continuación le mostramos el código asociado a cada botón y vea lo fácil que hace Forms la comunicación y llamadas entre módulos.

Imagen 61.

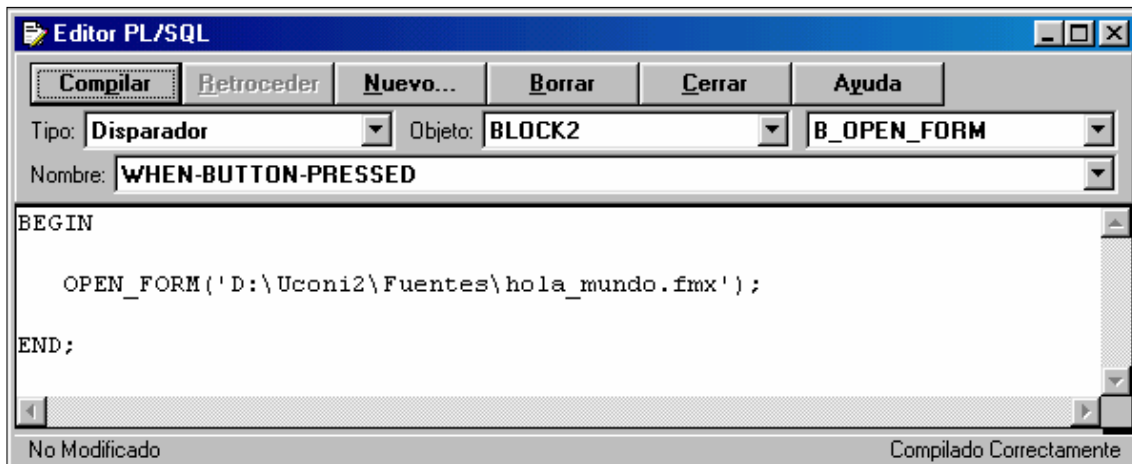


Imagen 62.

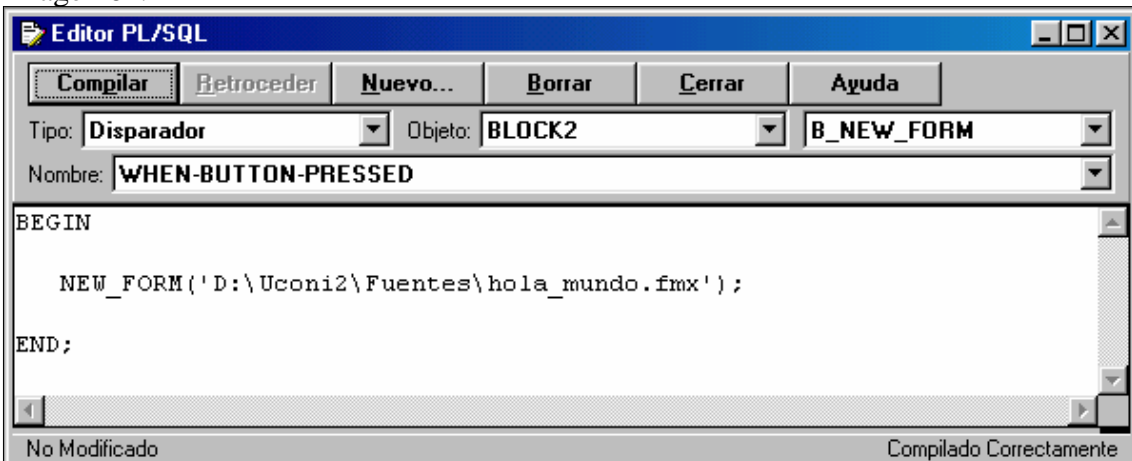


Imagen 63.

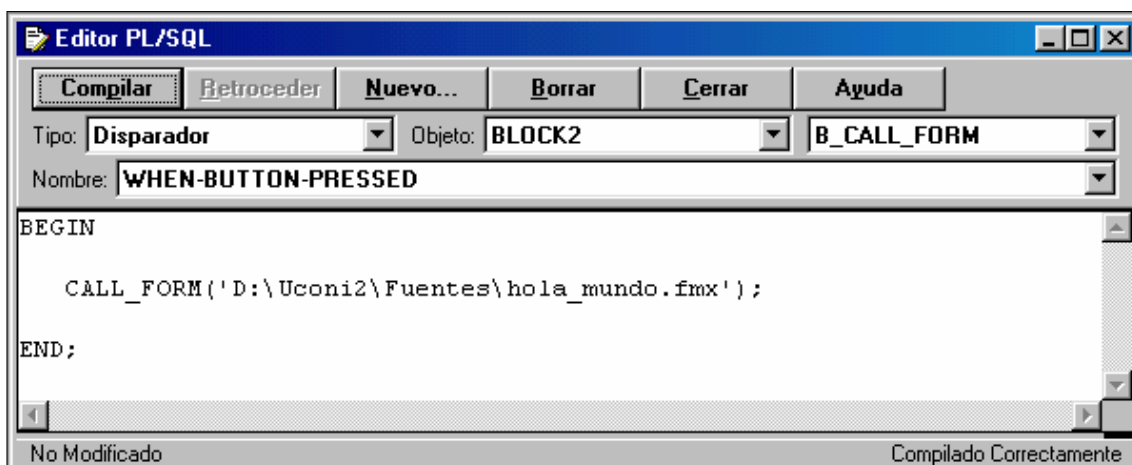
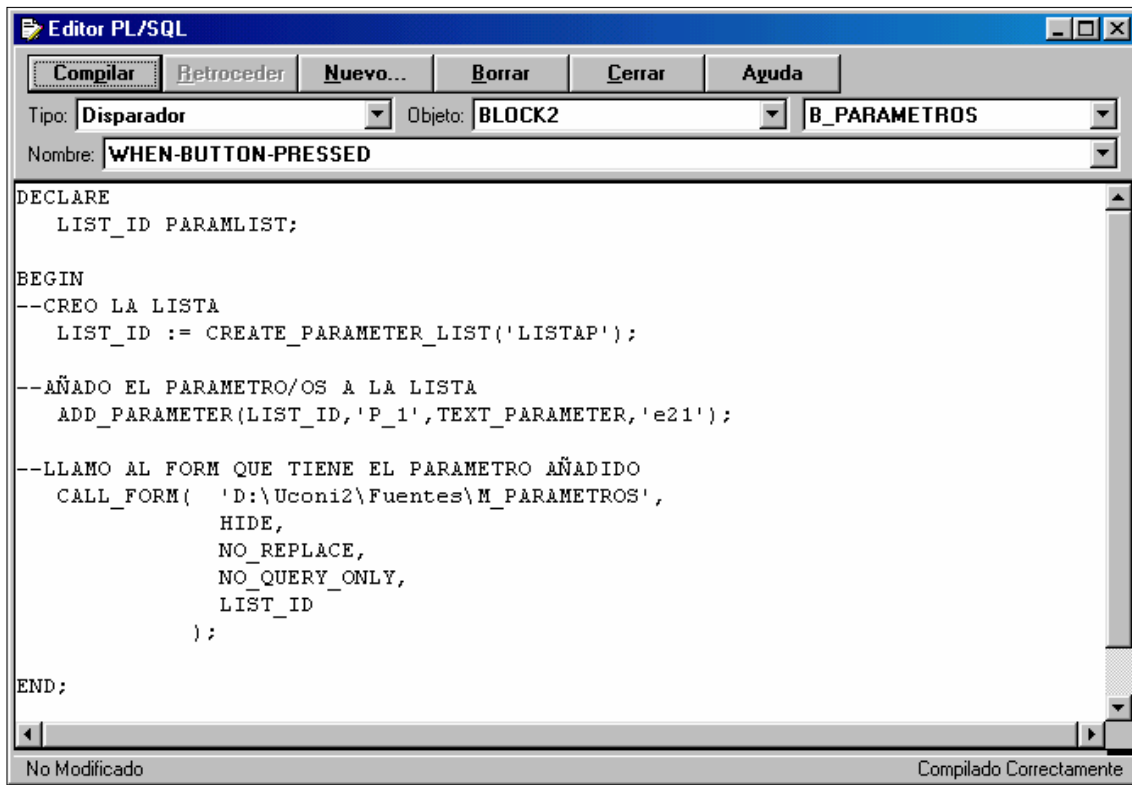
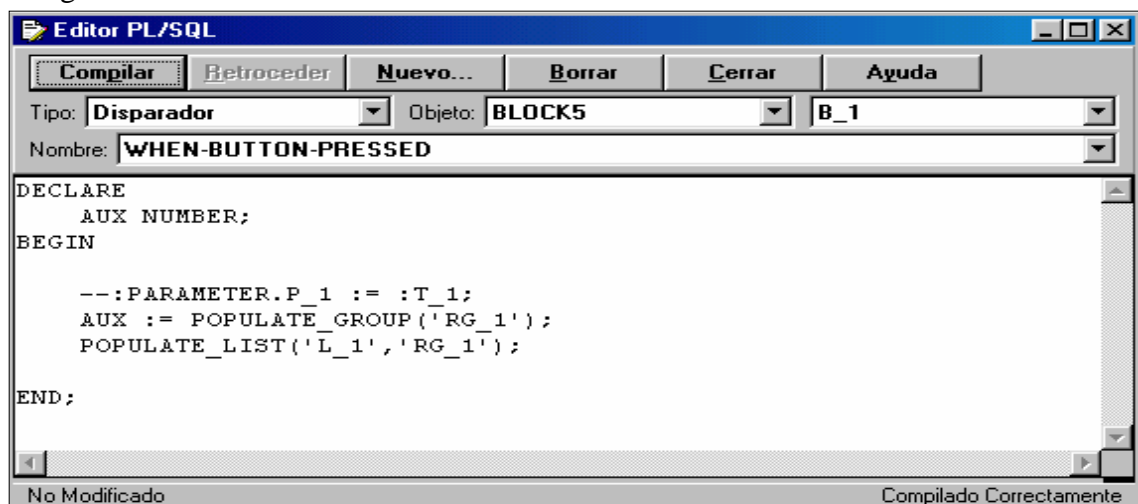


Imagen 64.



Como vimos anteriormente en el código de la Imagen 64 creamos una lista con un parámetro que después será pasado a otro modulo al ser llamado con CALL_FORM, este modulo llamado es el que debe tener el parámetro definido para poder recogerlo, hemos utilizado como modulo llamado el del ejemplo de carga de un List Item con un parámetro, repase la sección PARAMETROS_1 y vea como era el ejemplo, nosotros le mostramos las líneas de código que incluía a continuación para que vea que hemos modificado en ese programa la entrada de datos al parámetro que se hacia desde un Text Item para que ahora el parámetro se coja desde el modulo llamante, en concreto lo único que se ha de hacer es comentar una línea como ve en la Imagen 65.

Imagen 65.



Debe usted también como es lógico escribir las rutas correctas donde se encuentran sus ficheros .fmx que son como ya vimos al principio del tutorial los ejecutables de Forms. Compile después la aplicación y observe como se produce todo, pruebe a cambiar los parámetros en las llamadas para comprobar los efectos que se producen.

11-CREACION DE MENUS:

La creación de menús en Forms es tratada de forma totalmente independiente a los canvas, se generan dentro de un modulo existente pero son ficheros totalmente independientes de los FMB, un menú en forms consta de tres ficheros:

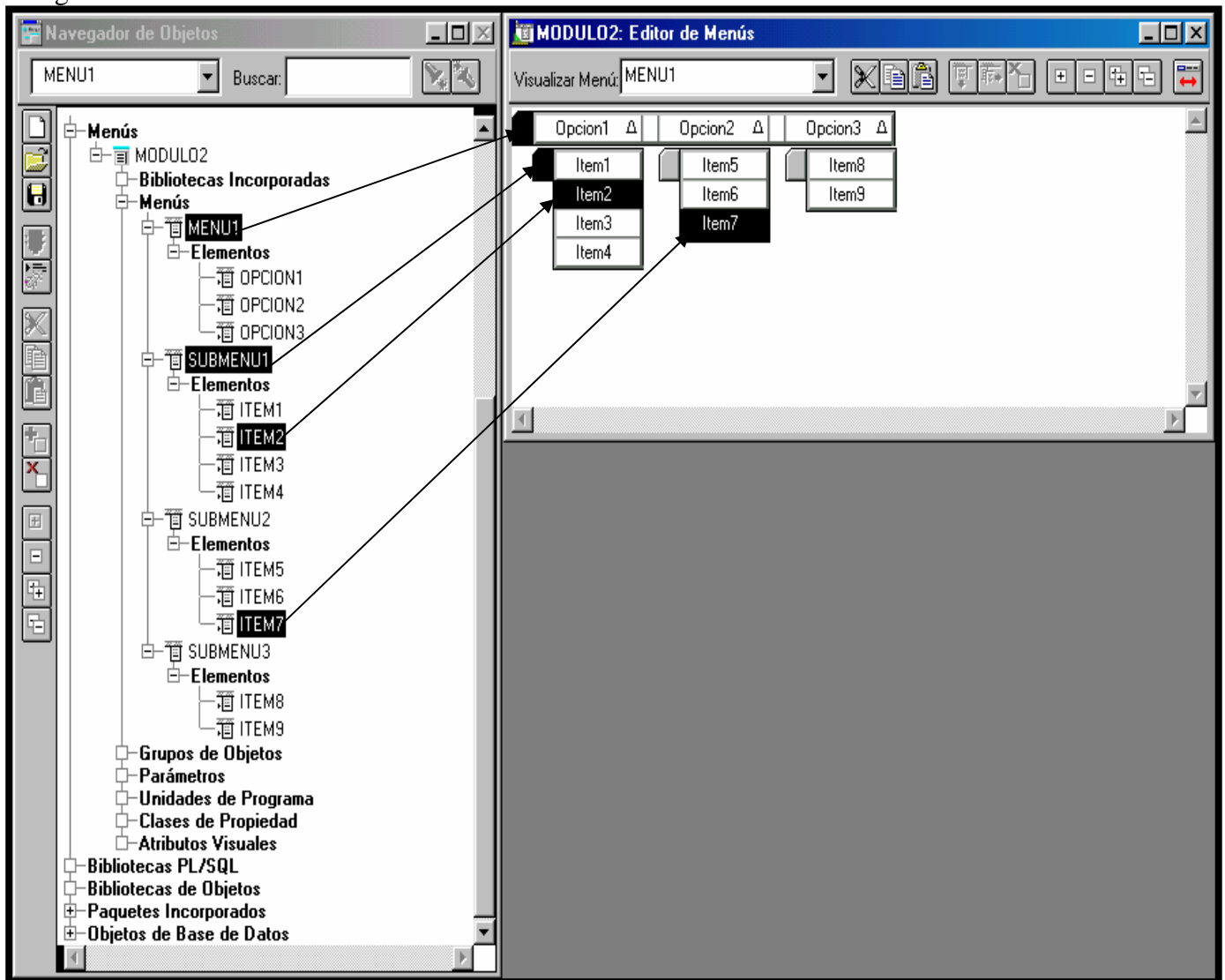
- *.mmb: fichero fuente visual de un menú
- *.mmt: fichero fuente en modo texto del menú
- *.mmx: fichero pseudoejecutable del menú

Debemos saber que solo puede haber un menú por aplicación en un momento determinado, si usted genera un menú propio vera como desaparece el menú que nos sitúa Forms por defecto, este menú no debe confundirnos pues aparece debido a que esta puesto ahí a propósito por los desarrolladores de Forms con la idea de que contiene las opciones mas habituales en una aplicación de bases de datos Oracle, por tanto en cuanto creamos nuestro menú el menú por defecto desaparecerá.

Acuda ahora al navegador y busque el elemento menú, genere un nuevo menú y no se confunda con Menú Desplegable si esta usando usted la versión 5.0 de Forms pues esos son otro tipo de menús.

Una vez creado un menú vera como se ha creado un modulo nuevo, pero esta vez no a nivel de Forms sino a nivel de Menú, haga doble click ahora sobre el nombre del modulo a nivel de Menú y vea como aparece la ventana de diseño del menú, puede ver el ejemplo creado en la Imagen 67.

Imagen 67.



Como puede ver Forms establece varios niveles dentro del Menú; en primer lugar encontramos el menú propiamente dicho, en este caso llamado MENU1, a partir de aquí se empiezan a establecer los niveles en el navegador que están en relación con la ubicación en el diseño no con la funcionalidad, en primer lugar encontramos MENU1 que englobara las opciones que siempre podrán ser vistas por los usuarios sin necesidad de desplegar nada (Archivo Edición Ver etc..) las etiquetas de estas opciones serian Opcion1, Opcion2 y Opcion3, dentro de cada una de estas opciones habrá submenús a desplegar, estos son SUBMENU1, SUBMENU2 y SUBMENU3, y dentro de esos submenús habrá a su vez nuevas opciones, o nuevos submenús o simplemente acciones a realizar, estas ultimas serian Item1, Item2, etc.

Para ir añadiendo o eliminando elementos del menú utilice los botones:



Como puede ver el sistema de creación de menús es muy sencillo basta con añadir o quitar, para introducir código en un menú basta con situarse sobre el elemento del menú que nos interesa y con el botón derecho invocar al editor PL/SQL, es muy importante saber que un elemento de menú siempre debe llevar código por lo que para hacer sus pruebas añada si no tiene una funcionalidad pensada la instrucción NULL a aquellos elementos del menú que lo necesiten.

Una vez creado el menú haga un click en algún elemento del mismo y diríjase a la barra de menú de Forms, abra Fichero/Administración/Compilar fichero, una vez hecho esto vaya al directorio de trabajo y compruebe que aparece en la parte inferior de Forms un mensaje indicando que el modulo se ha generado correctamente, si es así vaya al directorio de trabajo de Oracle o donde este trabajando actualmente y compruebe que se encuentra allí el fichero del menú ejecutable, si le ha llamado MI_MENU por ejemplo será MI_MENU.mmx.

Ahora nos falta saber como se asocia un modulo normal con un menú, es muy sencillo, escoja cualquier ejemplo que haya realizado anteriormente y acceda a las propiedades del modulo, basta con que escriba la ruta completa donde se encuentra el menú ejecutable dentro de la propiedad Modulo de Menús en el apartado Funcional y este aparecerá en la aplicación sustituyendo al menú que hasta ahora le aparecía siempre por defecto en sus aplicaciones.

Existe un convenio tácito entre los desarrolladores de Forms acerca de los menús que puede ser utilizado, en principio no se suelen crear menús y submenús con mas de nueve o diez opciones existiendo un máximo de dos submenús por cada menú, tampoco se suelen crear menús dinámicos pues esto suele confundir bastante al usuario final.

-Built In mas importantes:

DISABLE_ITEM('opción_principal', 'opción') deshabilitaría una opción
Concreta
('main_menu', 'opción') deshabilita una opción
completa.

ENABLE_ITEM: hace lo contrario al built in anterior

ITEM_ENABLED('nombre_menu', 'nombre_del_item') devuelve true o false si la opción indicada esta activada o desactivada respectivamente.

REPLACE_MENU('nombre del nuevo menú');

Una propiedad importante dentro de un Elemento del Menú es Tipo de elemento de menú, se encuentra dentro del apartado funcional, en ella podremos especificar si queremos que el elemento se comporte como un Check Box o un Radio Button, así mediante el siguiente trigger podremos modificar esta propiedad.

SET y GET_MENU_ITEM_PROPERTY('menu','opción','propiedad')

Dentro de Elemento del Menú encontrara el valor Magic, este no es mas que las opciones estándar establecidas en los entornos Windows como son Copiar, Pegar, Cortar, etc.

12-ANEXOS:

12.1-IDENTIFICADOR INTERNO:

Hasta ahora siempre nos hemos referido a los elementos creados por la etiqueta que les hemos asignado, así a los botones solemos denominarlos B_SALIR, B_ACTUALIZAR, etc. Este nombre sin embargo no es por el que Forms conoce verdaderamente a los Items creados, Forms crea en memoria una tabla de ítems con la siguiente estructura:

Identificador	Etiqueta	Propiedad1	Propiedad2
1	B_ACEPTAR	100	False
2	C_MENU	299	100
3	C_PRIMERO	300	True

Como puede ver Forms asigna a cada control un identificador interno que es transparente al programador pero que este puede utilizar si lo desea. La diferencia entre un método y otro estriba en la rapidez, cada vez que por ejemplo modificamos una propiedad de un ítem Forms se ve obligado a buscarlo en esta tabla por el nombre, lo que implica recorrer la tabla hasta que este sea encontrado. Sin embargo podemos averiguar el identificador que ha asignado Forms a un ítem concreto con lo que la búsqueda será mucho mas rápida pues se realizara de forma dicotómica, para hacer esto debemos averiguar como hemos dicho el identificador asignado por Forms, realizaremos dicha operación con un Built In especifico que existe para cada Item, como ejemplo utilizaremos una alerta.

Imagina la alerta ALERT_1, con la función FIND_ALERT('Nombre Alerta') Forms nos devolverá un valor del tipo ALERT, que es un subtipo de NUMBER creado previamente por Forms, de este modo tendríamos:

```
DECLARE

    ALERT_AUX      ALERT;

BEGIN

    ALERT_AUX := FIND_ALERT('ALERT_1');

END;
```

A partir de este momento podríamos utilizar ALERT_AUX para nuestras operaciones con la alerta.

El inconveniente de este método es que el código pierde claridad ya que no utilizaremos la etiqueta de la alerta por lo que este tipo de acciones es conveniente que estén bien documentadas en el código. Consulte en la documentación de Forms para saber en que ítems puede utilizar estos Built In.

12.2-VALIDACIONES:

En muchas ocasiones se plantea la necesidad de comprobar que los datos escritos por el usuario son correctos, en Forms podremos realizar validaciones de forma clásica es decir mediante código PL/SQL o a través de los propios instrumentos que tiene Forms.

La validación por código se reduce a controlar los eventos y lo que queremos que se controle mediante código PL/SQL, así si un ítem no tiene restricciones de entrada y es un char de cuarenta caracteres y no queremos que Forms lo controle por hacerlo nosotros por código bastara con preguntar si el ítem es NULL o NOT NULL para saber si tiene contenido, a continuación analizaremos el contenido, etc. lógicamente cada caso revestirá unas características particulares.

Si queremos aprovechar las herramientas de Forms debemos prestar atención a lo definido en las propiedades de cada ítem y a la secuencia de validación del propio Forms que mostramos a continuación.

Forms validara a cuatro niveles de una forma semiautomático:

- Ítem
- Registro
- Bloque
- Formulario

Casi siempre validaremos a nivel de ítem por lo que vamos a detenernos mas en ello.

- 1º- Comprobará mascarar de formato establecidas en las propiedades.
- 2º- Comprobará si el campo es requerido u obligatorio.
- 3º- Comprobará si la longitud es correcta
- 4º- Comprobará el tipo de dato
- 5º -Comprobará que este dentro de un rango

Después de estos cinco controles se lanzaran dos trigger que ya podemos controlar nosotros:

```
POST_CHANGE  
WHEN_VALIDATE_ITEM
```

-A nivel de registro validara registro a registro utilizando los cinco pasos anteriores para a continuación ejecutar dos triggers que podemos controlar

```
WHEN_RECORD_CHANGE  
WHEN_VALIDATE_RECORD
```

-A nivel de bloque primero se valida el bloque y después se repite el proceso anterior

-A nivel de Formulario se valida el formulario y después se repite el proceso anterior.

Como puede verse al final siempre se acaba comprobando el ítem.

Un ultimo detalle es que no debemos confundir el termino validación con la acepción aquí utilizada con la validación asociada al COMMIT, esta se produce cada vez que se inserta, modifica o suprime algo en la base de datos y esta mas asociada con este hecho que con una validación en el sentido anterior, aunque se produzca también una validación de datos a un nivel mucho mas interno en la base de datos.