FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACION Y DOCUMENTACION "ING. BRUNO MASCANZONI"

El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.

Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

Los materiales a disposición son:

- **Libros.**
- **Tesis de posgrado.**
- **Noticias técnicas.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1980 a la fecha.**

En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.

El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.

El horario de servicio es de 10:00 a 19:30 horas de lunes a viernes.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

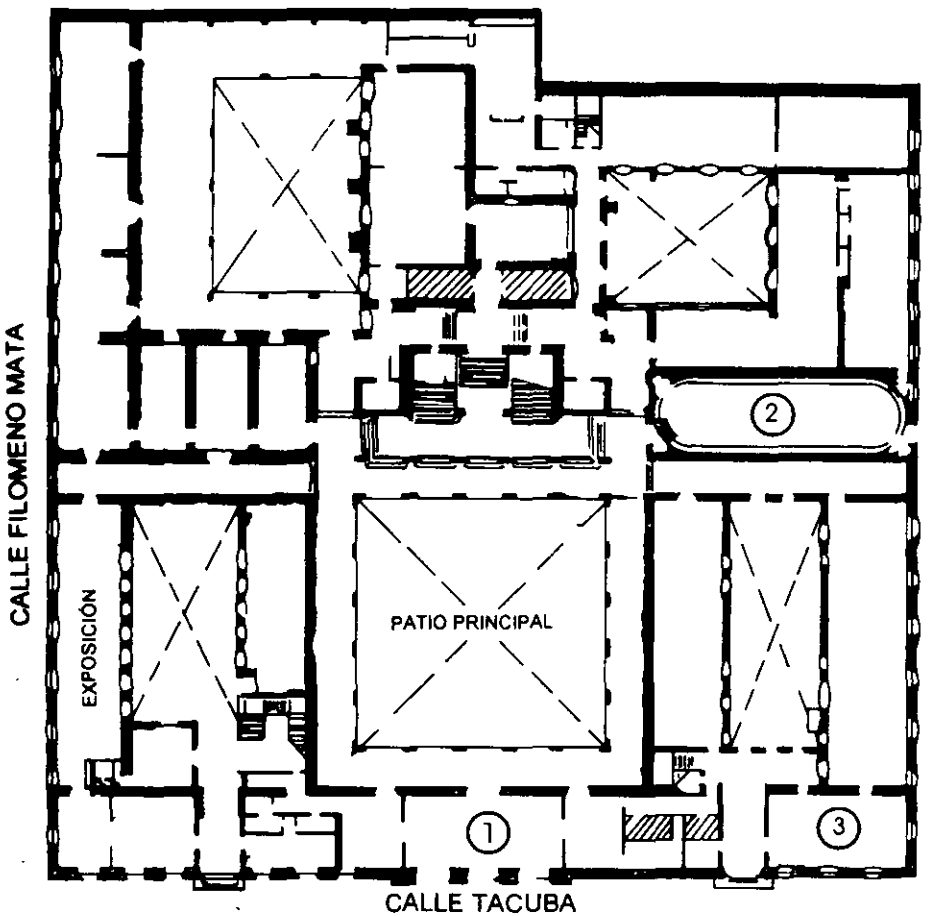
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

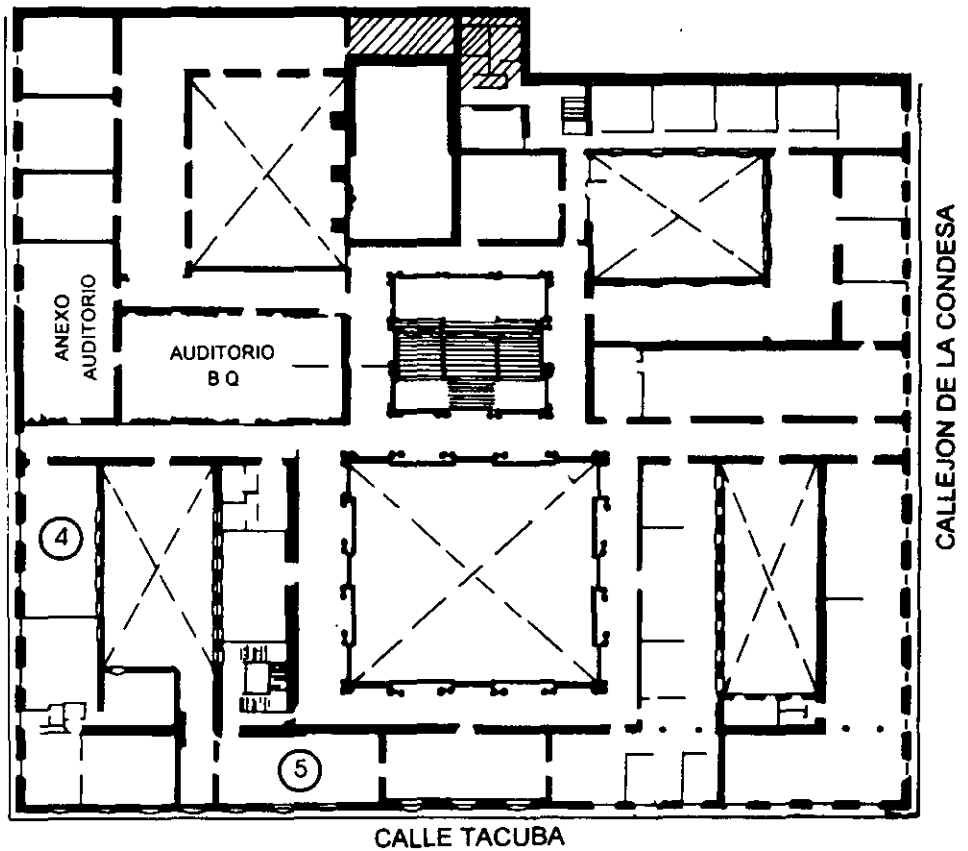
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO Postal M-2285
Teléfonos: 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

PALACIO DE MINERIA

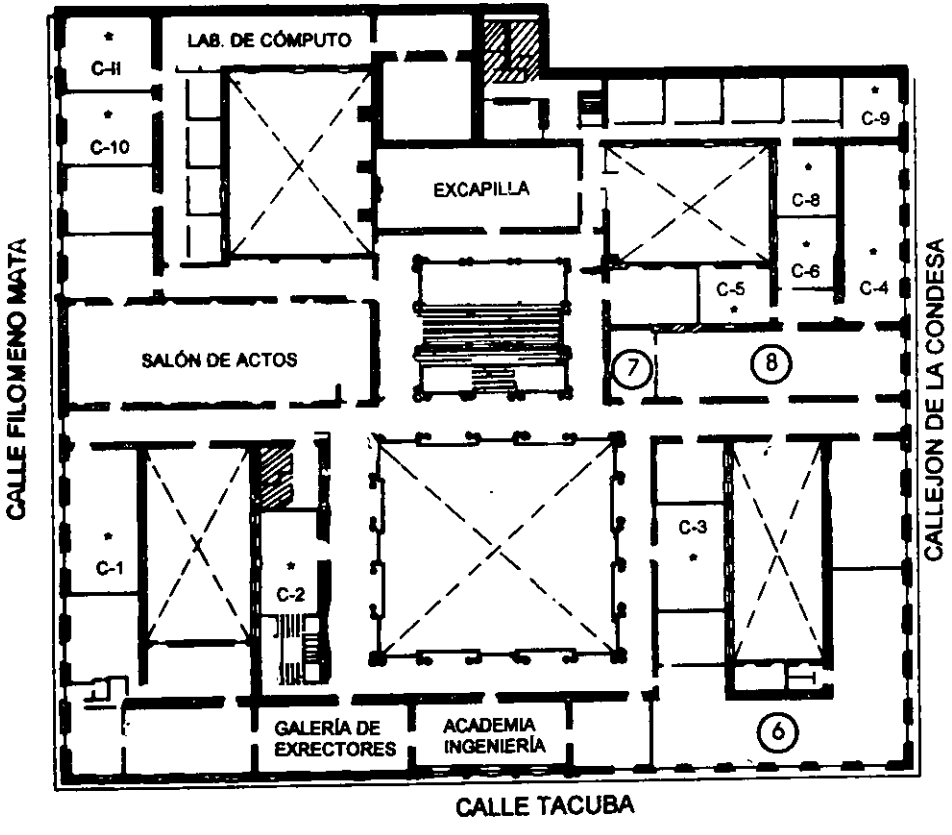


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

DIVISIÓN DE EDUCACIÓN CONTINUA DE LA FACULTAD DE INGENIERÍA, UNAM
EVALUACIÓN DEL PERSONAL DOCENTE

Curso: CC18 VISUAL BASIC BASICO

Instructor: ING. OSCAR MARTÍN DEL CAMPO CARDENAS

Duración: 32 Hrs.

El propósito de este cuestionario es conocer su opinión acerca del desarrollo. Marque con una "X" considerando el siguiente puntaje. Es muy importante contar con su objetividad.

10=Excelente 9=Muy bueno 8=Bueno 7=Suficiente 6=Malo 5=Deficiente

EVALUACIÓN DEL INSTRUCTOR

Factores a evaluar	10	9	8	7	6	5
1. Mostró dominio del tema						
2. Utilizo un lenguaje claro y sencillo						
3. Propicio la integración del grupo con el propósito de alcanzar el objetivo del curso						
4. Despertó y mantuvo el interés de los participantes						
5. El instructor supervisó adecuadamente los trabajos						
6. Resolvió oportunamente las dudas y los problemas de los participantes						
7. Manejo correctamente los apoyos y recursos didácticos durante su intervención						
8. Ante situaciones conflictivas presentadas por el grupo el instructor fue profesional en su actuación						
9. Ilustro los temas con casos prácticos						
10. Inició y concluyó puntualmente y empleó adecuadamente el tiempo destinado para su exposición						

Comentarios y sugerencias:

Factores a evaluar	10	9	8	7	6	5
1. El temario del curso cumplió sus expectativas						
2. El conocimiento adquirido es aplicable en las funciones que desempeña						
3. Los temas tuvieron una secuencia lógica						
4. Las instalaciones fueron adecuadas y cómodas						
5. La coordinación del curso fue adecuada						
6. La duración del curso fue suficiente						
7. Los ejercicios y la dinámica fueron acordes con el contenido del curso						
8. Los temas acordes tuvieron un equilibrio teórico práctico						
9. Los materiales de apoyo y manuales empleados fueron suficientes y de calidad						

Comentarios y sugerencias:

FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

Material didáctico del curso

Visual Basic Avanzado

ING. OSCAR MARTÍN DEL CAMPO
MAYO, 2004

Palacio de Minería, Calle de Tacuba # 5-Entrepiso, Centro Histórico, Delegación Cuauhtémoc, Código Postal 06000,
México D.F. Tel/FAX 5510-4673, Teléfonos 5623-2947 y 5623-2960

ÍNDICE

Prólogo

Este documento sirve como material de referencia para el Curso de *Visual Basic* Avanzado. El propósito de este manual no es abarcar, explorar y/o profundizar en su totalidad cada uno de los temas presentados.

El documento está acompañado por ejercicios útiles para ejemplificar muchos de los conceptos que se exponen en el curso. Para seguir los ejemplos es necesario poner atención a los íconos que los acompañan:

 Descripción del ejercicio e instrucciones a seguir para el desarrollo del ejercicio.

 Operaciones que se deben realizar en tiempo de diseño; tales como: incluir controles, ajustar propiedades a controles, presionar controles de la barra de herramientas, etc. Para los controles que insertemos se incluye entre paréntesis el nombre que el ejemplo contempla para cada control.

 Programación que debemos incluir en nuestro proyecto. Se mostrará el nombre de la rutina en la que debemos incluir o modificar el código, pero sólo se mostrarán los parámetros de las funciones o subrutinas en casos particulares.

 Para reforzar el aprendizaje cada capítulo contempla un laboratorio para que el alumno realice fuera de clase.

Muchas veces es más fácil explicar un concepto con el uso de un ejemplo, por lo que varios ejemplos están planteados como ejercicios ha ir desarrollando en los que se van ampliando paso a paso los conceptos.

Durante las explicaciones, el tipo de letra más pequeño se utiliza para mostrar como sería el código que podríamos ver directamente en nuestra aplicación en *Visual Basic*. Por ejemplo:

```
Sub Form_Load ( )

End Sub
```

Pero en los ejemplos, por facilitar la lectura se sigue utilizando el tipo de letra normal.

El formato de *itálica* se usa para conceptos que no tengan una buena traducción al español, términos que se entiendan mejor en inglés o tengamos que trabajar en inglés, nombres de archivos o por ser términos especiales.

En muchos ejemplos donde se trabaja con archivos encontrará que la ruta de estos archivos empieza con **VB** ; esto se debe remplazar con la ruta completa en donde tenga instalado *Visual Basic*.

Capítulo 1.

Desarrollo de una interfaz

Es muy importante aparte de crear una aplicación robusta el facilitar al usuario el uso de la misma. Mucho del buen uso o frustración por parte del usuario dependerá de que el sistema funcione correctamente y que el usuario entienda y pueda utilizar fácilmente la aplicación que desarrollemos.

Una aplicación dentro de Windows también debe ser fácilmente controlada por medio del teclado, no debemos hacer que una aplicación dependa del ratón para poder trabajar (claro, en una aplicación de dibujo sería difícil implementar todo sin ratón). Muchas aplicaciones de captura de datos son más eficientes de trabajar exclusivamente con el teclado. Si le evitamos al usuario el siguiente proceso - tener que separar las manos del teclado, tomar el ratón, moverlo, seleccionar alguna función y volver al teclado- y en su lugar nuestro programa le permite controlar las funciones desde teclado; el usuario quedará muy complacido con el programa.

El desarrollo de una interfaz gráfica se basa en cuatro puntos muy importantes. Estos cuatro puntos no se deben considerar por separado, de hecho, deben estar muy bien relacionados entre sí. Por lo que el desarrollo de una interfaz debe considerar los siguientes puntos:

1.1 Intuitiva

El usuario al ver nuestra interfaz, debe de poder imaginarse como trabaja la aplicación. Pudiera ser que el usuario haya leído nuestro "Manual de Usuario", pero seguramente no va a recordar todo, cuando necesite realizar una función que no recuerde, lo más seguro es que intentará encontrar la función por su cuenta. Para encontrar lo que busca, el usuario va a utilizar su intuición y en mucho dependerá la interfaz que tenga enfrente para que tome los pasos adecuados. Si el usuario después de 3 intentos (¡o menos!) no logra hacer "funcionar" la aplicación, lo único que tendremos será un usuario frustrado que deberá interrumpir su trabajo para buscar ayuda de cómo realizar lo que requiere.

Es lógico que no podemos simplificar muchas funciones complicadas al grado de hacerlas intuitivas, pero una buena interfaz (tomando en cuenta todos los puntos para el desarrollo de una interfaz) seguro le proporcionará al usuario los medios para realizar satisfactoriamente todas las tareas que necesite.

Es importante que se entienda el alcance que tiene el poder de las comunicaciones visuales y que mucho de lo que haga el usuario o intente hacer, dependerá de cómo perciba él la aplicación.

1.2 Eficiente

La interfaz debe estar diseñada para que el usuario pueda realizar sus tareas de la manera más fácil y práctica. La interfaz le debe permitir al usuario realizar las tareas más comunes directamente, es decir, sin necesidad de pasar por varios menús o submenús, o tener que elegir entre muchas opciones si no es necesario, o incluir datos por default a las pantallas de captura (donde sea conveniente), etc.

1.3 Retroalimentación

Este es un punto de vital importancia, si el usuario presiona un botón de comandos, ¡esperará que algo pase! Por ejemplo, si el usuario presiona el botón de "Agregar" en una aplicación de base de datos, esperará que la interfaz le presente una forma de captura, si por el contrario al presionar el botón de "Agregar" el usuario no ve nada – aunque internamente el sistema esté listo para recibir un nuevo registro, pero eso él no lo sabe- volverá a presionar el botón de Agregar esperando que algo suceda, y como no pasa nada, ¡lo volverá a presionar!

Se disponen de varios métodos para darle información acerca del estado del sistema al usuario, esto es, retroalimentar al usuario. Si el usuario encuentra un menú en gris, con esto el sistema le indica que esa función existe, pero que no se han dado las condiciones necesarias para ejecutarla. Otro ejemplo sería que con un buen uso de colores le podemos resaltar al usuario las partes más importantes de una interfaz (no es recomendable utilizar demasiados colores para una, interfaz gráfica) o en las cuales queremos que centre su atención.

1.4 Consistente

Esto es que si en una parte de nuestra aplicación el usuario debe seguir ciertos pasos para llevar a cabo una función, debemos hacer que toda nuestra aplicación siga los mismo pasos para llevar a cabo la misma función. Poniéndolo de otro modo, si el usuario aprendió a manejar una función de nuestra aplicación, el usuario esperará que donde deba realizar la misma función, se hará de la manera que el ya aprendió.

Hay ciertas normas al programar en el ambiente de Windows, es prudente que nuestra aplicación sea consistente con ellas para que el usuario que está acostumbrado a manejar Windows no deba aprender dos cosas contradictorias para manejar nuestro sistema. Por ejemplo, si queremos brindarle a nuestro usuario atajos para copiar y pegar texto, no sería nada conveniente hacer que el atajo para copiar fuera CTRL+V y para pegar CTRL+C.

1.5 Amigable

El ser amigable es ser considerados con nuestro usuario. Si se presenta un error, debemos evitar desplegar inmediatamente cajas de mensaje con datos técnicos sobre el error (esto sólo confunde al usuario), debemos por lo general informar al usuario que se presentó un error y procurar darle indicaciones sobre que pasos seguir para corregirlo o volver a evitar el error en un futuro. Otra característica importante es darle al usuario la oportunidad de cancelar acciones. Esto, con sus límites, en una interfaz de captura no sería conveniente confirmar cada acción de "Grabar" por parte del usuario, en este caso considero sería hacer trabajar de más al usuario si después de presionar "Grabar" tuviera que responder "si" a la acción que acaba de solicitar al sistema.

Otra manera de hacer amigable al sistema es proporcionar al usuario los datos que necesita para trabajar. Por ejemplo, si en la interfaz de captura en un campo en particular el usuario sólo puede insertar ciertos valores, podríamos poner esos valores en un control Combo y así el usuario podría capturar registros más fácilmente y con menos posibilidades de error.

1.6 Laboratorio

- Desarrollaremos una interfaz que le permita al usuario controlar la aplicación utilizando el teclado. Se tomarán en consideración los puntos mencionados para el desarrollo de una interfaz gráfica.

Una de las funciones que el usuario da al ratón es el utilizarlo para dar el foco a un cierto control. Por ejemplo, en una forma de captura de datos podríamos tener muchas cajas de texto en las cuáles el usuario debe capturar su información. Desarrollaremos un ejemplo utilizando la base de datos *Biblio.Mdb*.

- Abra un nuevo proyecto y cree las cajas de texto y etiquetas necesarias para la captura de información de la tabla "*Titles*". Los campos de esta tabla son los siguientes:

Nombre	Tipo	Tamaño (bytes)	Llave	Descripción
Title	Texto	255		Título del libro.
Year Published	Número (entero)	2		Año de publicación
ISBN	Texto	20	Primaria	Estándar en la clasificación de libros.
PubID	Número (largo)	4	Foránea	Identificador de la editorial
Description	Texto	50		Descripción del libro.
Notes	Texto	50		Notas.
Subject	Texto	50		Temas.
Comments	Memo	-		Comentarios.

Biblio.Mdb* en *Visual Basic 6.0

Nombre	Tipo	Tamaño (bytes)	Llave	Descripción
Title	Texto	255		Título del libro.
Year Published	Número (entero)	2		Año de publicación
AU_ID	Número (largo)	4	Foránea	Identificador de autor
ISBN	Texto	20	Primaria	Estándar en la clasificación de libros.
PubID	Número (largo)	4	Foránea	Identificador de la editorial

***Biblio.Mdb* en *Visual Basic* 3.0**

- Agregue los botones de comandos para las operaciones de agregar, borrar, grabar, búsquedas y navegación (registro anterior, siguiente, último, primero).

Recuerde que la palabra principal de un sistema gráfico debe ser "retroalimentación". El orden en que mostremos los campos le puede indicar al usuario la importancia de los mismos, por ejemplo no sería lo mismo si presentáramos el campo de Comentarios al principio o final de nuestra interfaz de captura. Con la interfaz le podemos indicar al usuario que campos son los más importantes, cuáles son indispensables, cuáles son opcionales o cuáles sólo aceptan valores específicos.

Basándonos en que el orden de presentación en la interfaz indica la importancia del campo, un orden adecuado para la tabla de "*Titles*" sería el siguiente: ISBN, Title, PubID, Year Published, Subject, Description, Notes y Comments. En *Visual Basic* 3.0 podrían ser ISBN, Title, Au_ID, PubID y Year Published.

- Modifique su interfaz para que refleje este orden.

Para facilitar el proceso de captura, lo primero será proveer una manera de accesar los controles dentro de la interfaz utilizando el teclado. Con la tecla [TAB] el usuario de Windows puede ir moviéndose a través de los controles en una pantalla, las aplicaciones de *Visual Basic* funcionan también de esta manera, pero para que sea útil, es importante especificar correctamente el orden de recorrido de los controles.

- Ejecute su aplicación y pruebe el orden de recorrido.

Como apreció el orden hasta el momento no es útil para el usuario. La propiedad TabIndex le indica a *Visual Basic* cual es el orden que debe de seguir para recorrer controles. Al insertar objetos dentro de una forma, *Visual Basic* asigna el TabIndex progresivamente.

- Modifique la propiedad TabIndex para que se recorran las cajas de texto y los botones en el orden correcto.

Con este pequeño cambio hacemos un poco más fácil la utilización de nuestra aplicación. Además, con la combinación de [SHIFT] + [TAB] los controles se recorren en orden inverso. Pero, ¿qué hacemos si el usuario quiere ir directamente a un campo sin tener que recorrer todos los controles? En nuestra aplicación son hasta el momento pocos controles, pero en alguna aplicación más complicada los controles a recorrer pueden ser muchos hasta el punto que se vuelva poco práctico utilizar este método de recorrer controles para llegar a uno en especial.

Habrá notado que las etiquetas no reciben el foco. Cuando *Visual Basic* encuentra un objeto que no puede recibir el foco, automáticamente le asigna el foco al control con el TabIndex siguiente. Si ningún control puede recibir el foco, el foco queda sobre la forma. Siempre que diseñemos una interfaz, debemos ponernos en el lugar del usuario para imaginarnos que podría necesitar el usuario para trabajar más fácilmente. Muchas veces un usuario no nos exigirá una característica en particular, pero si un día llega a necesitarla y la aplicación la provee, esto seguro hablará bien de la aplicación.

Por ejemplo, en nuestra interfaz le proveemos al usuario un medio de navegación, pero ¿qué pasa si el usuario quiere modificar sólo un campo que se encuentra justo en el medio de la forma? Lo que puede hacer hasta el momento sería navegar por todos los controles hasta alcanzar el que requiero o bien, recurrir al ratón para seleccionar el campo. Entonces lo que nos corresponde hacer es darle a nuestro usuario una manera para accesar de manera directa el campo que necesite. Haremos esto utilizando atajos de teclado. Un atajo de teclado es por lo general una combinación de teclas presionadas al mismo tiempo o teclas especiales (como F1 – F10), que disparan un acción. Un lugar muy común para poner atajos de teclado es en los menús, seguro ya ha trabajado con aplicaciones que en su menú contengan el menú Archivo, la interfaz gráfica nos indica al subrayar la letra 'A' que con la combinación de teclas ALT + A podrá desplegar ese menú.

Aprovecharemos este tipo de atajos para que el usuario pueda seleccionar directamente cualquier campo de su elección utilizando las etiquetas. En nuestra interfaz será la etiqueta la que tenga definida un atajo de teclado, per al no poder recibir el foco la etiqueta, éste pasará automáticamente al control con el *TabIndex* siguiente, que deberá ser la caja de texto que el usuario busca seleccionar.

- Defina a las etiquetas una letra que funcione como atajo de teclado (anteponga el carácter '&' a la letra que quiera se utilice en la propiedad *Caption*). Y vuelva a ordenar la propiedad *TabIndex*. Pruebe su interfaz.

NOTA: Tenga cuidado en no repetir atajos de teclado en su interfaz.

Pero una de las maneras más comunes de cambiar de campo es presionando la tecla de [RETORNO] o [ENTER]. Esto funcionaría para todas las cajas de texto excepto por las cajas multilínea. Para agregar esto, podríamos detectar cuando el usuario presione [ENTER] en el evento *keypress* de las cajas de texto. En este evento podríamos asignar el foco a la siguiente caja de texto en nuestra interfaz.

```
Sub txtISBN_Keypress ( keyascii as integer)
      If keyascii = 13 then ' 13 es el código ascii del [Enter]
            txtTitulo.SetFocus
      End if
End Sub
```

Todos los campos texto de una base de datos tienen una longitud definida (los campos Memo son diferentes). Será muy importante que el tamaño de mi caja de texto le dé una indicación al usuario de cuanta información puede capturar. Por ejemplo, los campos Descripción, Temas y Notas miden 50 caracteres, en la interfaz estas cajas de texto deberán medir lo mismo (ser consistentes) y el campo ISBN, que mide 20 caracteres, deberá de ser de una longitud menor.

También podemos definir la propiedad *MaxLength* en nuestras cajas de texto para limitar el número de caracteres que cada caja de texto puede recibir y esto nos podría evitar errores al momento de grabar a la base de datos.

- Defina los tamaños de campos (algunos campos pueden ser texto multilínea) y defina su propiedad *MaxLength*.

Aún más, deberíamos agregar reglas de validación en nuestra interfaz. Por ejemplo, el campo Año sólo debe aceptar 4 dígitos. Podríamos desplegar un mensaje cada vez que el usuario tecleará un carácter indicándole que ese campo sólo acepta dígitos. Se tienen controles especializados para definir máscaras de entrada de datos, con los cuáles podríamos definir máscaras de captura para el RFC, fechas, teléfonos; o bien, si no contamos con estos controles realizar la verificación con código.

Para completar:

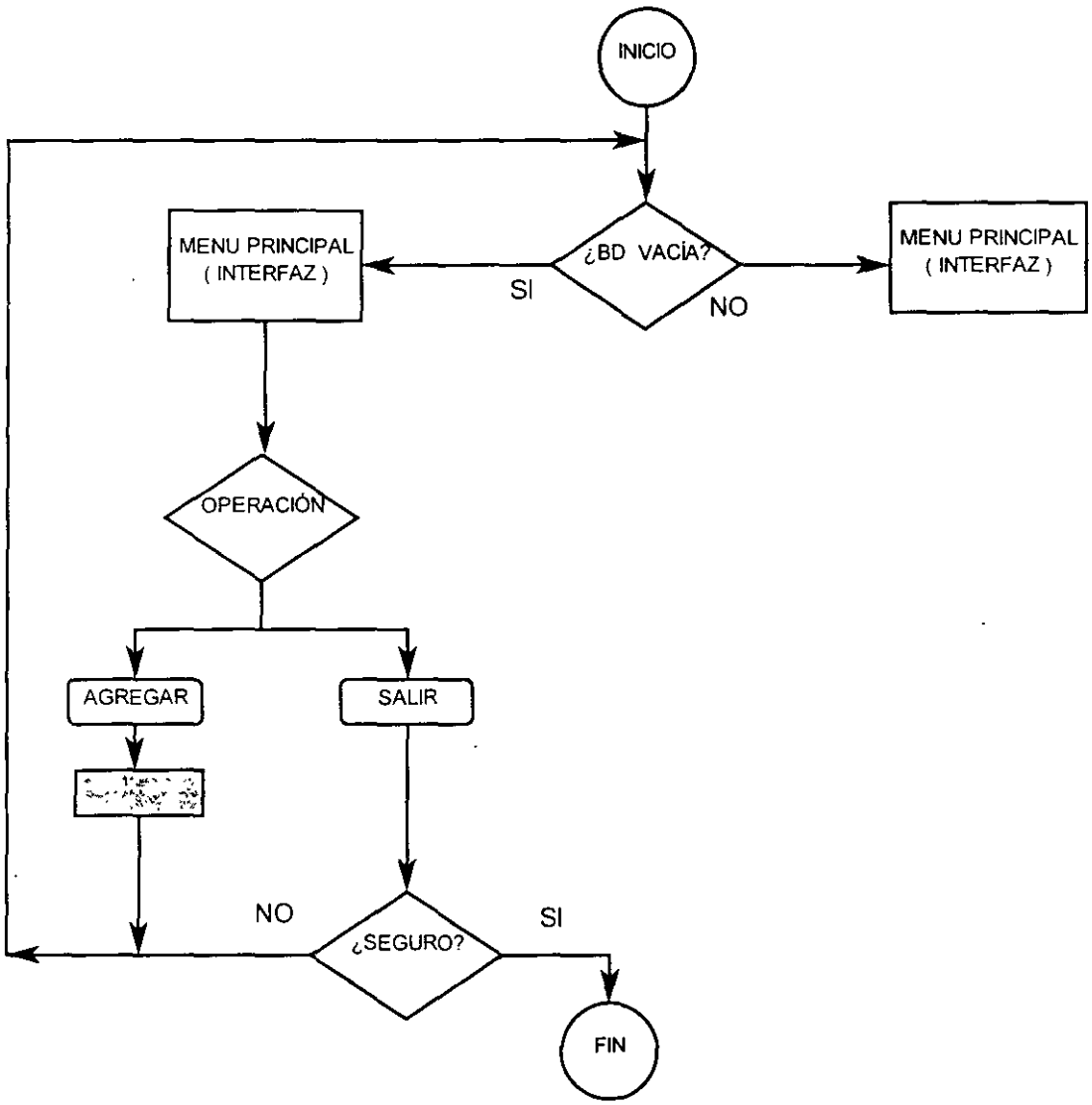
Una parte importante de la interfaz, es que ésta sea amigable con el usuario, en algunas operaciones se ha hecho costumbre verificar las acciones del usuario antes de ejecutarlas, con esto le damos al usuario la oportunidad de cancelar acciones que en verdad no requiera. Casos típicos son cuando vamos a borrar registros o cuando salimos de un sistema y el usuario no ha grabado su información.

- Añada el código que le despliegue cajas de confirmación al usuario.

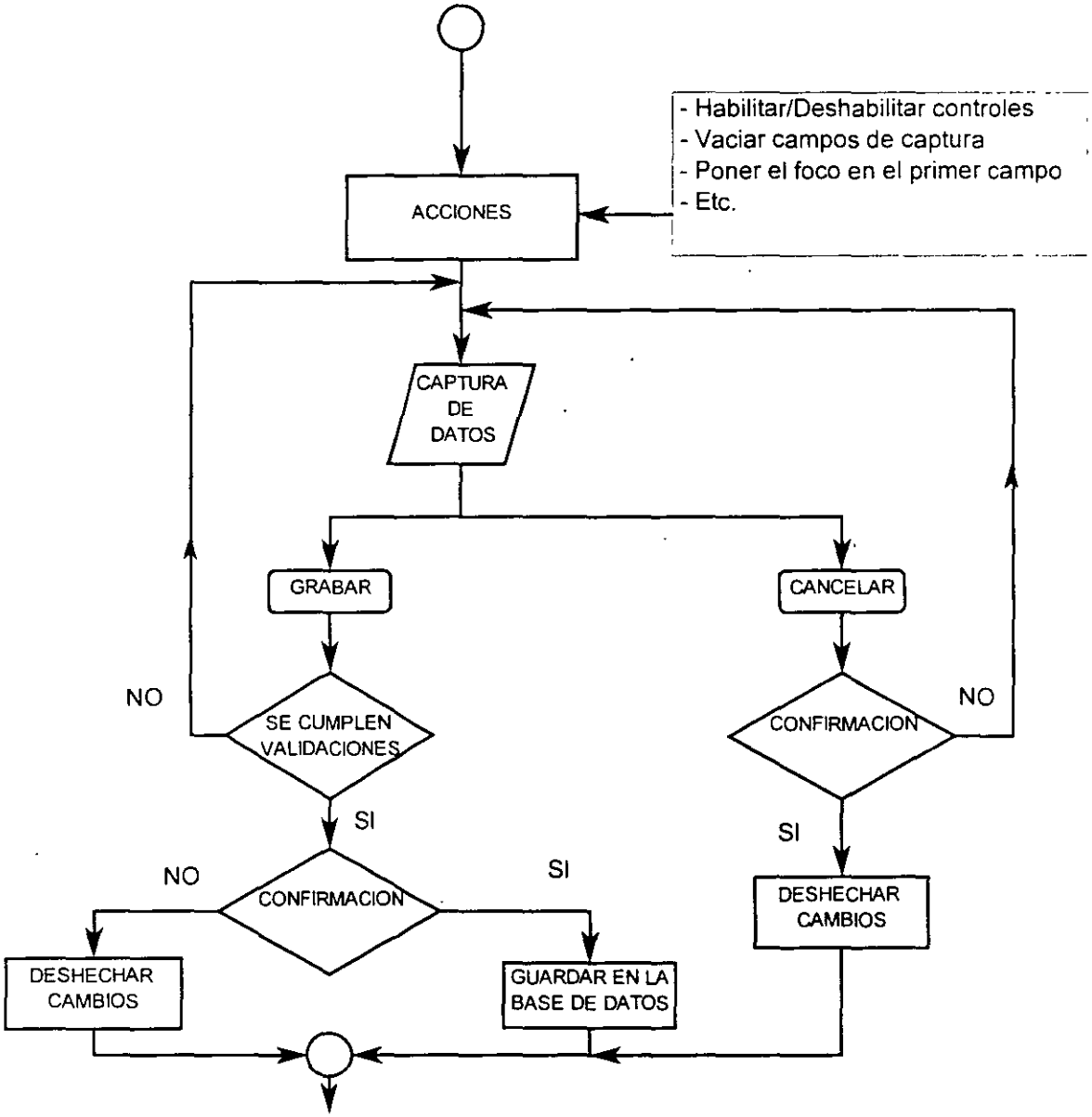
Al estar trabajando con el sistema, el usuario ejecutará acciones que por lo general deben deshabilitar otras, la interfaz debe reflejar estos cambios. Por ejemplo, el botón de "Grabar" sólo debería estar activo si hay datos que almacenar; como cuando el usuario está agregando o modificando un registro. Esta retroalimentación le indica al usuario más concretamente que acciones puede realizar. Otro ejemplo: si la base de datos está vacía, el usuario no podrá efectuar búsquedas, ni borrar registros; la única acción que podría hacer es agregar registros. Esto nos ayuda a guiar a nuestro usuario con respecto a que acciones puede realizar en el sistema, y esto es mucho mejor que desplegar una caja de mensajes que diga: "No puede borrar registros en este momento", a lo que el usuario se podría preguntar, ¿entonces para que está disponible esa función?

Para nuestra aplicación a desarrollar podemos definir el diagrama de flujo de las acciones que el usuario podrá ejecutar. En una situación real las opciones y operaciones serán seguramente un mayor número y con mayor complejidad, pero nuestro ejemplo de todos modos será muy útil. Una buena interfaz va a ir guiando al usuario por la serie de acciones que él podrá elegir y es lo que mostraremos.

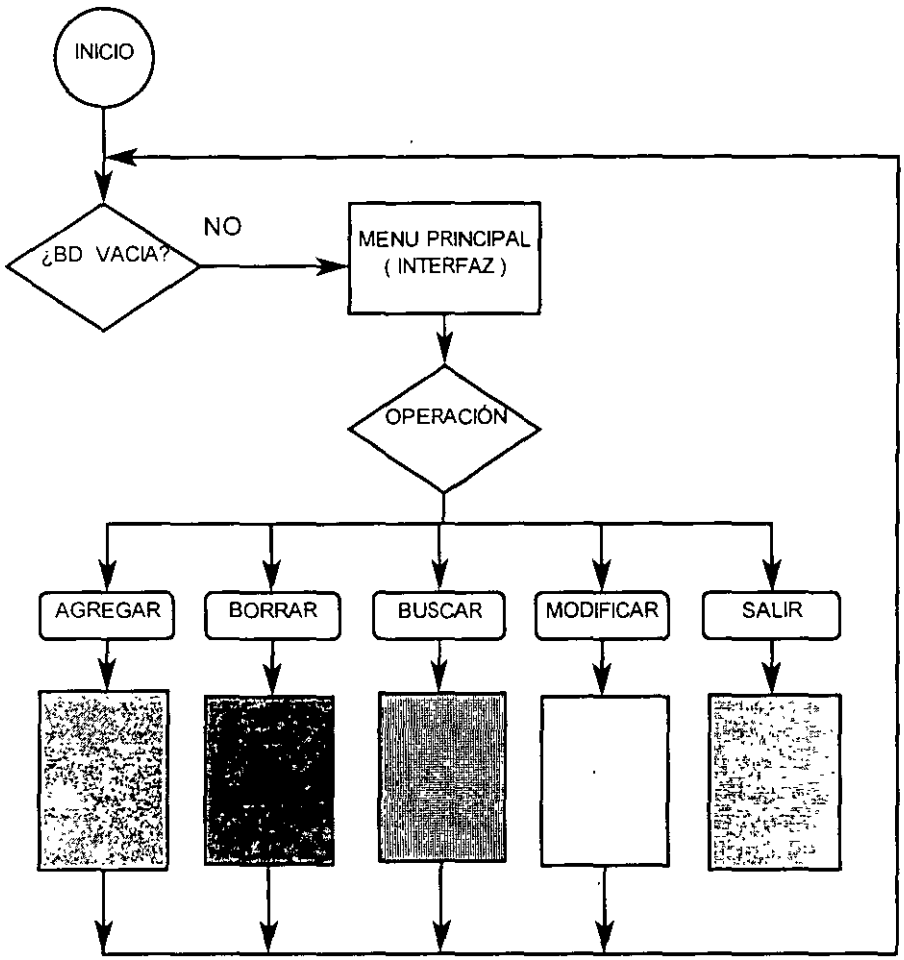
Por lo mencionado, nuestra aplicación presentará diferentes opciones dependiendo de si la base de datos está vacía o no. Si la base de datos está vacía sólo le permitiremos al usuario agregar un nuevo registro o salir del sistema.



Si el usuario selecciona la operación de "Agregar", las operaciones que podrá realizar serán las que especifiquemos con el siguiente flujo de acciones:



Si la base de datos no se encuentra vacía, el usuario tendrá más operaciones de donde escoger.



- Haga los diagramas de flujo para las operaciones de "Borrar", "Buscar" y " Modificar" y agregue esta funcionalidad a su interfaz.

- Realice la interfaz para las tablas de Autores y Editoriales. Antes es muy buen que se familiarice con el diseño de la base de datos.

Capítulo 2.

Codificación de eventos del ratón

Las aplicaciones en *Visual Basic* pueden responder a varios eventos disparados a través del ratón, como pueden ser los *clicks* hechos con los botones, el movimiento del ratón, y operaciones compuestas como es la de arrastrar y soltar (*drag and drop*).

Los eventos que dispara el ratón, son recibidos en los controles en que se encuentra nuestro apuntador en ese momento. Una forma atraparía los eventos del ratón si el apuntador se encuentra sobre una parte de la forma que no contenga controles. Un control atrapa el evento si el apuntador se encuentra sobre ese control.

Entre los eventos más comunes que dispara el ratón sobre los objetos están:

Evento	**Descripción**
MouseMove	Ocurre cada vez que el apuntador se mueve a una nueva posición en la pantalla.
MouseDown	Ocurre cada vez que se presiona algún botón del ratón.
MouseUp	Ocurre cada vez que el usuario libera algún botón del ratón.

Así, en cada objeto que pueda atrapar los eventos del ratón, se tiene los siguientes procedimientos:
Objeto_MouseMove
Objeto_MouseUp
Objeto_MouseDown

Por ejemplo, para los objetos "forma" tiene el evento:

```
Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    'Dentro de esta subrutina pondríamos el código que quisiéramos se
    'ejecutara cada vez que se mueve el apuntador del ratón sobre la forma.
End Sub

Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    'Dentro de esta subrutina pondríamos el código que quisiéramos se
    ejecutara cada vez que se presione algún botón del ratón sobre la forma.
End Sub

Sub Form_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    'Dentro de esta subrutina pondríamos el código que quisiéramos se
    'ejecutara cada vez que se libere algún botón del ratón sobre la forma.
End Sub
```

Estas tres subrutinas reciben los mismos argumentos:

Argumento	Descripción
Button	Determina el botón que se presionó. 1 – Izquierdo 2 – Derecho 4 – Central
Shift	Determina si las teclas *ALT, SHIFT y CTRL* están presionados. 1 – *SHIFT* 2 – *CTRL* 4 – *ALT*
X	La posición horizontal del apuntador utilizando el sistema de coordenadas del objeto que recibe el evento.
Y	La posición vertical del apuntador utilizando el sistema de coordenadas del objeto que recibe el evento.

Con el argumento *Shift* se pueden determinar combinaciones de las teclas presionadas. Si el argumento *Shift* tiene un valor de 5 (4 + 1), significa que las teclas de [*ALT*] y [*SHIFT*] están presionadas al momento de que se disparó el evento.

La operación de arrastre y soltar sirve para disparar acciones relacionando elementos gráficos. Por ejemplo: en el Explorador de Windows podemos seleccionar un archivo con el ratón, arrastrarlo y soltarlo sobre el ícono de una unidad de disco, con esto le decimos al sistema que queremos que copie ese archivo a esa unidad.

En *Visual Basic* se puede utilizar esta misma operación de arrastre y soltar dentro de nuestras aplicaciones. Para está operación, se utilizan los siguientes métodos, eventos y propiedades:

Método	Descripción
Drag	Inicia o cancela el arrastre. 0 – Cancela el arrastre 1 – Inicia el arrastre 2 – Termina el arrastre soltando el objeto

Eventos	Descripción
DragDrop	Se dispara cuando un objeto que está siendo arrastrado es soltado. Este evento lo recibe el objeto sobre el cuál se tira el control que está siendo arrastrado.
DragOver	Se dispara cuando un objeto está siendo arrastrado sobre el control.

Propiedades	Descripción
DragMode	Determina el modo de arrastre. Automático o manual.
DragIcon	Especifica el ícono que se despliega mientras el control es arrastrado.

2.1 Inicio del arrastre

Para iniciar el arrastre de un objeto se puede hacer especificando la propiedad *DragMode* o bien utilizando el método *Drag*. La propiedad *DragMode* acepta los siguientes valores:

DragMode	Descripción
0	Manual
1	Automático

Si la propiedad *DragMode* de un objeto está en Automático, el arrastre se inicia cuando el usuario presiona con el botón izquierdo del ratón sobre este objeto. Si la propiedad *DragMode* está en Manual, el arrastre del objeto se debe de iniciar desde código invocando el método *Drag*.

2.2 Responder a la operación de arrastre y soltar

Para terminar una operación de arrastre, se suelta el objeto que estamos arrastrando encima de otro objeto. Por ejemplo, si arrastramos un control *Picture* encima de la forma, al momento de soltar el botón soltamos el control *Picture* sobre la forma. El objeto sobre el cuál se suelta otro objeto recibe el evento *DragDrop*. En nuestro ejemplo, la forma recibiría el evento *DragDrop*.

El evento *DragDrop* tiene esta sintaxis:

```
Sub Form_DragDrop (Source As Control, X As Single, Y As Single)

End Sub
```

El evento *DragDrop* recibe tres argumentos, "*Source*", "X" y "Y". Los argumentos "X" y "Y" determinan la posición dentro del objeto donde fue soltado el objeto, el argumento "*Source*" es una referencia hacia el tipo de control que le es soltado encima y para determinar el tipo de objeto al que apunta este argumento, se utiliza una sentencia "*If*" con las estructura "*Typeof Is*".

La expresión *Typeof* verifica si el argumento (en este caso *Source*) es de un tipo de objeto determinado comparando con un nombre de clase. En este caso, el nombre de la clase de los objetos *Picture* es *PictureBox*. Puede determinar el nombre de la clase en la ventana de propiedades.

El apuntador del ratón también nos sirve para dar información al usuario. Por ejemplo, al momento de estar arrastrando un control le podemos indicar al usuario en que áreas o controles está permitido soltar cierto control. Al momento de arrastrar un control la propiedad *DragIcon* determina el ícono que se muestra, por lo que en código podemos modificar esta propiedad acorde a lo que le queramos indicarle al usuario.

Para cambiar de manera dinámica el ícono de arrastre, podemos utilizar el evento *DragOver*. Cuando una operación de arrastre pasa por encima de un control, se dispara este evento.

```
Sub Form_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
    If TypeOf Source is CommandButton then
        Source.Dragicon = picIncorrecto.picture
    Else
        Source.Dragicon = picCorrecto.picture
    End if
End Sub
```

La propiedad *MousePointer* nos permite cambiar el apuntador del ratón a varios valores por default, uno de los más usados es cambiar el ícono del apuntador por el de un reloj de arena para indicar que el sistema está ocupado con un proceso. Por ejemplo si una rutina toma mucho tiempo, cambiando el apuntador del ratón le podemos indicar al usuario que espere.

```
Sub CalculaMatriz( )
    Dim i as long
    Form1.MousePointer = 11          'El cursor toma forma de reloj de arena
        For i = 0 to 30000000            ' Rutinas que llevan tiempo
        Next i
    Form1.MousePointer = 1           'El cursor regresa a su forma normal
End sub
```

2.3 Ejemplo A. Arrastre y soltar.

El siguiente ejemplo deberá desplegar la imagen almacenada en un archivo. Para seleccionar el gráfico a desplegar se arrastrará y soltará el nombre del archivo hasta un control *Picture*.

Inserte un control *File List Box* (filGráficos) y un *Picture* (picIcono).

Especifique el tipo de archivos a desplegar.

```
Sub filGráficos ( )
  File1.Pattern = "*.bmp;*.ico"  'Se desplieguen únicamente archivos de tipo gráfico
End Sub
```

En el evento Form_Load indique el directorio va a desplegar el objeto *File List Box*.

```
Sub Form_Load ( )
  filGráficos.Path = "VB\icons\misc"  'Un directorio con archivos gráficos
End Sub
```

Inicie el arrastre manual en filGráficos_MouseDown y especificaremos el ícono de inicio de arrastre.

```
Sub filGráficos ( )
   filGráficos.Dragicon = "VB\icons\dragdrop\drag1pg.ico"
   filGráficos.Drag 1              'Inicio de arrastre
End Sub
```

El ícono de arrastre es muy útil para indicar donde se puede depositar un objeto. En la aplicación el único lugar válido donde se puede soltar el objeto es sobre el control *Picture*

Cambie el ícono de arrastre en cualquier objeto que no sea el *Picture*.

```
Sub Form_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   Source.DragIcon = LoadPicture("VB\icons\misc\misc06.ico")
End Sub
```

Para indicar que el objeto se puede soltar sobre el control *Picture*, también cambiamos el ícono de arrastre en el evento picIcono_DragOver.

```
Sub picIcono_Dragover(Source As Control, X As Single, Y As Single, State As Integer)
   Source.DragIcon = LoadPicture("VB\icons\dragdrop\drop1pg.ico")
End Sub
```

Se muestra el gráfico cuando se suelte un objeto sobre el control *Picture*.

```
Sub picIcono_DragDrop (Source As Control, X As Single, Y As Single)
   picIcono.Picture=LoadPicture(Source.Path + "\" + Source.FileName )
End Sub
```

¡ Pruebe el ejemplo !

2.4 Laboratorio

Codifique una aplicación de dibujo. El programa debe permitir dibujar líneas, rectángulos, círculos y dibujo a mano alzada. En el caso que se dibujen rectángulos y círculos, si el usuario al mismo tiempo mantiene presionada la tecla de SHIFT, el rectángulo o círculo tendrá que ser relleno con un color.

Capítulo 3.

Manejo de archivos

Visual Basic maneja tres diferentes tipos de acceso a archivos: acceso secuencial, acceso binario o acceso aleatorio. Cuando *Visual Basic* accesa un archivo, hay que especificar la manera como queremos que trabaje el archivo para que sepa como debe de interpretar la información que lee o escribe.

Para trabajar con archivos, no importa el tipo de acceso, se siguen estos pasos:

- Seleccionar el archivo (obtener la ruta hacia el archivo)
- Obtener un número de archivo válido
- Abrir el archivo
- Trabajar con el archivo (operaciones de lectura y/o escritura)
- Cerrar el archivo

3.1 Obtener un número de archivo válido

El número para uso de archivo es un valor entero que usa el sistema operativo para hacer referencia al archivo. Una vez abierto un archivo, todas las operaciones que hagamos se harán a través de este número de archivo. Para obtener un número para archivo válido se usa la función *FreeFile*.

```
Dim fhandle as integer ' Esta variable almacenará el número de archivo para hacer referencia al archivo
fhandle = Freefile
```

La función *FreeFile* regresa el valor del siguiente número de archivo disponible, por lo que entre la invocación de la función *FreeFile* y el abrir el archivo no debe de haber otras instrucciones para que otra aplicación no utilice ese número y en nuestra aplicación se presenten problemas.

3.2 Abrir un archivo

Para abrir un archivo se utiliza la sentencia *Open* que tiene la siguiente sintaxis:
***Open* archivoS *For* tipo *As* fhandle% [*Len*=longitud_registro]**

donde: **archivoS** es la ruta al archivo con el que queremos trabajar.
tipo determina la clase de acceso que haremos.
fhandle% es la variable que contiene el número de archivo que usaremos para hacer referencia al archivo.
Len se usa para indicar la longitud del registro.

La instrucción para abrir un archivo variará dependiendo del tipo de acceso deseado:

- Secuencial

El acceso secuencial está diseñado para ser usado con archivos de texto. Cada carácter en el archivo se considera representa un carácter de texto o una secuencia de formato para el texto, tales como la secuencia "nueva línea". Para abrir un archivo de tipo secuencial se tiene tres sentencias diferentes.

- Para abrir el archivo secuencial en forma de lectura
  Open archivo for input as fhandle

- Para abrir el archivo secuencial en forma de escritura
  Open archivo for output as fhandle

- Para abrir el archivo secuencial para concatenar información.
  Open archivo for append as fhandle

Si abrimos un archivo en forma de lectura no podemos escribir al archivo, primero tendríamos que cerrar el archivo y volverlo a abrir en forma de escritura; de la misma manera no puedo leer si el archivo lo abrí en forma de escritura.

Si al momento de abrir un archivo de lectura (*input*) , éste no existe, *Visual Basic* generará un error. Si abrimos un archivo en forma de escritura (*output*) y éste no existe, *Visual Basic* lo creará; si ya existe, *Visual Basic* lo sobreescribirá. En modo de concatenación (*append*) si el archivo no existe, lo crea; si ya existe agregará lo que grabemos al final del archivo.

- **Aleatorio**
  El acceso aleatorio considera que el archivo está compuesto por una serie de registros de igual longitud. La longitud de cada registro debe ser proporcionada al momento de abrir el archivo, sino, *Visual Basic* asume que el registro tiene una longitud de 128 caracteres. Esta longitud se usa para calcular la posición de cada registro dentro del archivo. Después lo único que se necesita para accesar un registro, es especificar el número de registro. Los archivos aleatorios se abren en forma de lectura y escritura.

  Open archivo for Random As fhandle Len=longitud_registro

- **Binario**
  El acceso binario permite almacenar datos como queramos, no se hacen suposiciones sobre el tipo de datos. Para recuperar la información, es necesario saber exactamente como es que la información está almacenada. Los archivos binarios se abren en forma de lectura y escritura.

  Open archivo for Binary as fhandle

3.3 Trabajar con el archivo

Para trabajar con archivos, las instrucciones dependerán del tipo de archivo que se haya abierto. Las instrucciones para leer información y grabar datos son diferentes si el tipo de archivo es secuencial, aleatorio o binario.

3.3.1 Archivos Secuenciales

Para trabajar con archivos de acceso secuencial tenemos que indicar si vamos a utilizar ese archivo para leer información, para grabar información o para anexar información. Con los archivos secuenciales no podemos ejecutar una operación de lectura y escritura con el mismo manejador de archivos, por ejemplo, si queremos escribir sobre un archivo que fue abierto para lectura, tenemos que cerrar ese archivo y volverlo a abrir en modo de escritura.

Si vamos a leer información del archivo, se utiliza la sentencia siguiente para abrir el archivo:

Open archivo for Input as fhandle

Para leer información del archivo se tienen las siguientes funciones:

- **cadenaS = Input(num, fhandle)**

Lee "num" número de caracteres del archivo asociado con el número de archivo fhandle y los almacena en una variable cadena.

```
' Leer 100 caracteres de un archivo
textoS = Input ( 100, fhandle)
```

- **Line Input #fhandle, cadenaS**

Esta instrucción lee información hasta que encuentra un carácter de fin de línea; es decir, lee una línea del archivo. La línea que lee se almacena en la variable cadena que se pasa como argumento.

```
' Leer una línea de un archivo
Line Input #fhandle, textoS
```

- **Input #fhandle, lista_de_variables**

Esta instrucción lee información que este separada por comas, y asigna cada valor que lee a una variable en "lista_de_variables". Esta sentencia se utiliza cuando la información en el archivo se grabó utilizando la sentencia *Write #*.

```
'Leer el nombre, antigüedad y puesto de un archivo que haya sido creado con Write #
Input #fhandle, nombreS, ant%, puestoS
```

Para almacenar datos en un archivo de tipo secuencial, se debe abrir el archivo con la cláusula *output* o *append.*

Open archivo for Output as fhandle
Open archivo for Append as fhandle

Para grabar información, se utiliza las siguientes instrucciones:

- **Print #fhandle, cadena$**

  Esta instrucción almacena lo que tenga la variable cadena en el archivo.

```
' Grabar lo que esta en la variable texto al archivo
Print #fhandle, texto$
```

- **Write #fhandle, [lista de expresiones]**

  Esta instrucción almacena el contenido de las expresiones en el archivo. Automáticamente separa cada expresión con una coma y encierra entre comillas las cadenas.

```
'Grabar lo que contenga las variables nombre, antigüedad y puesto al archivo.
Write #fhandle, nombre$, ant%, puesto$
```

3.3.2. Archivos de acceso aleatorio

Una vez que abrimos el archivo en modo de acceso aleatorio, para trabajar con la información utilizamos las sentencias *Put* para escribir registros y *Get* para leer registros.

- **Get fhandle, posición, variable**

  La función *Get* toma del archivo relacionado con "fhandle", el número de registro especificado por posición y lee ese registro hacia la variable que especifiquemos. Es común que la variable donde se almacena lo leído del archivo sea una estructura que contenga todos los campos que vamos a leer del registro.

```
'Leer el registro 10 de un archivo aleatorio a la variable empleado
Get fhandle, 10, empleado
```

- **Put fhandle, posición, variable**

  La función *Put* pondrá el contenido de la variable en el registro especificado por posición al archivo relacionado con "fhandle". Si no se especifica una posición, la función *Put* insertará un registro a partir de la posición actual. Para añadir registros, es suficiente especificar que la posición sea el número total de registros más uno.

```
'Grabar una variable al archivo
Put fhandle, 5, empleado
```

3.3.3. Archivos binarios

Para trabajar con estos archivos utilizamos de igual manera las instrucciones *Put* y *Get*. Al momento de grabar información al archivo, la sentencia *Put* almacena exactamente la longitud de las variables (en acceso aleatorio rellena con espacios en blanco las cadenas). Al momento de leer el archivo la instrucción *Get* lee exactamente la longitud de la variable (que va a almacenar el resultado de la lectura) en bytes del archivo. Por eso cuando se trabaja con archivos binarios, es muy importante saber exactamente como se almacenó la información y asegurarnos un método para poder recuperar la información.

Por ejemplo:

En el acceso aleatorio, todos los registros son de una misma longitud, estos registros pueden tener diferentes campos, pero cada campo debe ser de una longitud fija. Así, *Visual Basic* sabe donde empieza y acaba un registro y al momento de extraer la información es fácil de encontrar. En los archivos de acceso binario, este no es el caso, la información que almacenemos puede tener una longitud variable, el caso es, ¿cómo saber cuantos bytes tenemos que leer para recuperar la información? Si vamos a grabar cadenas de longitud variable, una solución sería almacenar justo antes de la cadena su longitud, esto representaría que almacenaríamos 2 bytes más por cada cadena, pero nos permite recuperar la información: esto lo hace automáticamente *Visual Basic* si grabamos una cadena de longitud variable en un archivo aleatorio, pero no en un archivo binario.

3.4 Cerrar el archivo

Para cerrar un archivo se utiliza la instrucción *Close*. Se puede indicar a la instrucción *Close* el archivo a cerrar pasando como argumento el número de archivo. Si no se especifica ningún número de archivo, se cerrarán todos los archivos abiertos.

Al utilizar la sentencia *Close* para un archivo abierto en modo de escritura (*output* o *append*), el sistema operativo escribe el contenido del buffer al archivo, libera el buffer y cierra el archivo. Por eso es siempre conveniente que cerremos nosotros los archivos en cuanto terminemos de trabajar con ellos.

Close fhandle

3.5 Comandos para trabajar con archivos

Comando	Descripción
ChDir directorio$	Cambia el directorio default para una unidad de disco. Recibe la ruta al directorio. **ChDir** "\proyecto"
ChDrive unidad$	Cambia la unidad de disco actual. Recibe la unidad de disco. **ChDrive** "d:"
ruta = **CurDir** [(unidad)]	Regresa la ruta default de la unidad especificada; si no se especifica la unidad se toma la actual. ruta_actual$ = **CurDir**
nombre$ = **Dir$**("*.txt",0)	Regresa el nombre del archivo o directorio que concuerde con el patrón especificado. Recibe como argumento un patrón y un entero que indica los atributos. archivo$ = **Dir$**("*.txt",0)
i=**FileAttr**(fhandle, modo)	Regresa el modo en que fue abierto un archivo (*input, output, append, random o binary*) o el manejador del sistema operativo asociado al archivo. Recibe el número asociado al archivo y un valor entero que indica si debe devolver el manejador del sistema operativo o un valor que indica el modo en que fue abierto el archivo. i = **FileAttr**(farch, 1)
FileCopy fuente$, destino$	Copia el archivo indicado por fuente$ a destino$. Recibe como argumentos las rutas a los archivos. **FileCopy** "c:\bd.mdb" "c:\bd.bak"
modif$=**FileDateTime**(ruta$)	Regresa en una cadena la fecha de última modificación o de creación de un archivo. Recibe la ruta hacia el archivo. fecha$=**FileDateTime**("c:\autoexec.bat")
tamaño = **FileLen**(ruta$)	Regresa el tamaño en bytes de un archivo que no esté abierto. Recibe la ruta al archivo como parámetro. i = **FileLen**("c:\autoexec.bat")
EOF (fhandle)	Regresa un valor verdadero o falso indicando si se ha alcanzado el fin de archivo. En archivos de acceso aleatorio o binario indica si la última lectura no pudo leer un registro completo.
GetAttr (ruta$)	Regresa un entero indicando los atributos de un archivo, directorio o volumen. Recibe la ruta hacia el archivo, directorio o volumen. i = **GetAttr**("c:\autoexec.bat")

Kill archivo$	Borra el archivo especificado. Recibe la ruta al archivo. **Kill** "c:\temp\prueba.txt"
i = **Loc** (fhandle)	Regresa la posición actual dentro de un archivo abierto. Recibe al número de archivo. Para archivos de acceso aleatorio devuelve el número de registro, para archivos secuenciales, devuelve la posición actual dividida entre 128, para archivos binarios, devuelve la posición del último byte leído o escrito.
tamaño = **Lof** (fhandle)	Regresa el tamaño en bytes de un archivo abierto. Recibe el número de archivo como parámetro.
Mkdir directorio$	Crea un nuevo directorio. Recibe la ruta hacia el nuevo directorio. **Mkdir** "c:\respaldo"
Name antes$ **As** después$	Renombra un archivo o directorio. Recibe la ruta al nombre actual y la ruta al nuevo nombre. Se puede utilizar *Name* para mover archivos pero no para mover directorios. **Name** "c:\respaldo.bak" As "bd.mdb"
Rmdir directorio$	Borra un directorio vacío. Recibe la ruta hacia el directorio. **Rmdir** "c:\temp\"
SetAttr ruta$, atributos%	Modifica los atributos a un archivo. Recibe la ruta al archivo o directorio y un entero indicado que atributos debe modificar. **SetAttr** "c:\respaldo.bak", 1

3.6 Ejemplo A. Archivos Secuenciales.

- Se codificará una aplicación que despliegue el contenido de un archivo secuencial en una caja de texto, y que grabe el contenido de la caja de texto a un segundo archivo.

- Inserte una caja de texto (txtArchivo) y dos botones de comandos (cmdAbrir y cmdGrabar).

- La caja de texto debe ser multilínea para que despliegue correctamente los renglones del archivo.

```
txtArchivo.Multiline = True
txtArchivo.Scrollbars = 3   'Ambas
```

```
Sub cmdAbrir_Click ( )
    Dim farch as integer      'Será la variable que utilizaremos como número de archivo
    Dim archivo as string     'Almacenará la ruta al archivo que deseamos abrir
    archivo = InputBox("Dame la ruta absoluta al archivo", "Leer del archivo")
    farch = Freefile          'Obtenemos un número de archivo válido
    Open archivo for input as farch 'Se abre el archivo secuencial en modo lectura
    txtArchivo.text = Input( Lof( farch ), farch )    'Se lee el contenido del archivo y
                                                      'despliega  la caja de texto
    Close farch               'Se cierra el archivo
    End Sub

Sub cmdGrabar_Click ( )
    Dim farch as integer      'Será la variable que utilizaremos como número de archivo
    Dim archivo as string     'Almacenará la ruta al archivo que vamos a grabar
    archivo = InputBox("Dame la ruta absoluta al archivo", "Grabar al archivo")
                              'Obtenemos la ruta al archivo
    farch = Freefile          'Obtenemos un número de archivo válido
    Open archivo for output as farch    'Se abre el archivo secuencial en modo escritura
    Print# farch, txtArchivo.text       'Se lee el contenido del archivo y se despliega en
                                        'la caja de texto
    Close farch               'Se cierra el archivo
    End Sub
```

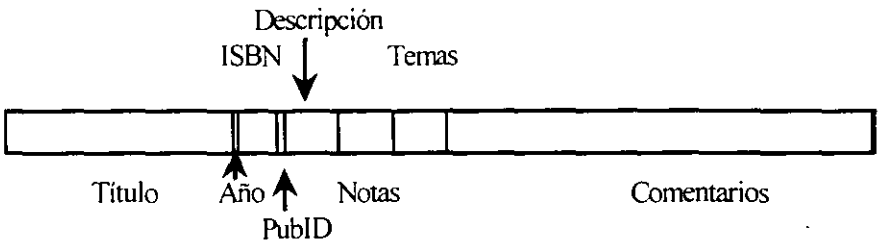
¡ Pruebe el ejemplo !

3.7 Ejemplo B. Archivos de Acceso Aleatorio

Codificar una aplicación que lea y grabe registros de un archivo de acceso aleatorio. Para este ejemplo se tomó la tabla de libros (*TITLES*) de la base de datos *Biblio.Mdb* que se distribuye con *Visual Basic* 4.0 ó 5.0 y se grabaron los registros al archivo *Títulos.Dat*. En el capítulo XI se muestra un ejemplo de cómo se creó el archivo *Títulos.Dat* a partir de la base de datos.

Cada registro dentro de este archivo contiene 8 campos:

- Nombre del libro. 255 caracteres.
- Año de publicación. Variable entera.
- ISBN. 20 caracteres.
- Identificador de Editorial. Variable entera larga.
- Descripción. 50 caracteres.
- Notas. 50 caracteres.
- Temas. 50 caracteres.
- Comentarios. 500 caracteres. El campo comentario en la base de datos es de tipo Memo. por lo que este campo debería ser de 64K, pero por facilidad lo trabajaremos de 500 caracteres.



Representación de un registro en *Títulos.Dat*

Inserte 8 cajas de texto (txtTítulo, txtAño, txtISBN, txtEditorial, txtDescripción, txtNotas, txtTemas y txtComentarios) y 4 botones de comandos (cmdLeer, cmdGrabar, cmdAnterior, cmdSiguiente).

Se creará un tipo de dato con los tipos de datos adecuados para leer la información del archivo. Dentro de la parte de declaraciones generales de un **módulo** declararemos el nuevo tipo de dato Registro_Título.

```
Type Registro_Título
        título as string * 255
        año as integer
        isbn as string * 20
        editorial as long
        descripción as string * 50
        notas as string * 50
        temas as string * 50
        comentarios as string * 500
End Type
```

Se utilizará un número de archivo y una variable de tipo Título en varios procedimientos, por lo que las declaramos a nivel forma en *general_declarations*.

```
Dim farch as integer
Dim libro as Registro_Título
```

Definiremos una subrutina que despliegue la información en la variable libro hacia las cajas de texto. Inserte una subrutina y llámela DesplegarLibro

```
Sub DesplegarLibro ( )
   txtTitulo = Rtrim$(libro.Título)
   txtAño = libro.Año
   txtISBN = Rtrim$(libro.ISBN)
   txtEditorial = libro.Editorial
   txtDescripción = Rtrim$(libro.Descripción)
   txtNotas = Rtrim$(libro.Notas)
   txtTemas = Rtrim$(libro.Temas)
   txtComentarios = Rtrim$(libro.Comentarios)
End Sub
```

- En el Form_Load abriremos el archivo *Títulos.Dat* y mostraremos el primer registro.

```
Sub Form_Load ( )
  farch=Freefile
  'Incluya la ruta donde tenga el archivo Títulos.Dat
  Open "RUTA\Títulos.Dat" for Random as farch Len = Len(libro)
  Get farch, 1, libro
  Call DesplegarLibro
End Sub
```

La función *Get* lee un registro del archivo y almacena el resultado en la variable que le indiquemos. En este ejemplo, la variable libro es una estructura, y *Visual Basic* hará lo siguiente:

La sentencia:

```
Get farch, 1, libro
```

la podemos conceptualizar como si pusiéramos:

```
Get farch 1, [libro.título, libro.año, libro.isbn, libro.editorial, libro.descripción.... ]
```

Entonces *Visual Basic* determina el tipo y longitud de la primera variable (libro.título), es de tipo cadena y longitud de 255 caracteres, por lo que lee del archivo 255 bytes y los almacena en esta variable. Después determina el tipo y longitud de la segunda variable (libro.Año), que es de tipo entero (los enteros ocupan 2 bytes), por lo que lee 2 bytes del archivo, los interpreta a valor entero y almacena este resultado en la variable libro.Año y así hasta terminar de llenar la variable.

- En el evento cmdSiguiente_Click aprovecharemos que si al utilizar la función *Get* no se determina el número de registro a leer, leerá el siguiente.

```
Sub cmdSiguiente_Click ( )
  Get farch, , libro
  If Not EOF (farch) then
      Call DesplegarLibro
  End if
End Sub
```

- En el evento cmdAnterior_Click desplegaremos el registro anterior.

```
Sub cmdAnterior_Click ( )
  IF Loc ( farch ) > 1 Then
      Get farch, Loc(farch) -1, libro
      Call DesplegarLibro
  End if
End Sub
```

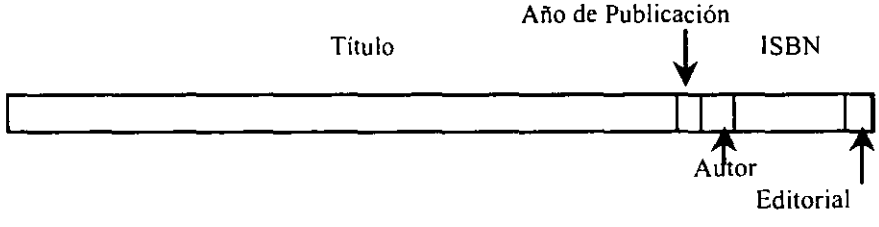
```
Sub cmdGrabar_Click ( )
    libro.Título = txtTítulo.text
    libro.Año = txtAño.text
    libro.ISBN = txtISBN.text
    libro.Editorial = txtEditorial.text
    libro.Descripción = txtDescripción.text
    libro.Notas = txtNotas.text
    libro.Temas = txtTemas.text
    libro.Comentarios = txtComentarios.text
    Put farch. Loc(farch), libro
End Sub
```

¡ Pruebe el ejemplo !

En *Visual Basic* 3.0 la base de datos es diferente, por lo que el archivo *Títulos.Dat* puede cambiar. Este mismo ejemplo es para un archivo *Títulos.Dat* creado a partir de *Biblio.Mdb* de *Visual Basic* 3.0.

Cada registro dentro de este archivo contiene 5 campos:

- Nombre del libro. 255 caracteres.
- Año de publicación. Variable entera.
- Identificador de Autor. Variable entera larga.
- ISBN. 20 caracteres.
- Identificador de Editorial. Variable entera larga.



Representación de un registro en *Títulos.Dat*

Inserte 5 cajas de texto (txtNombre, txtAño, txtAutor, txtISBN y txtEditorial) y 4 botones de comandos (cmdLeer, cmdGrabar, cmdAnterior, cmdSiguiente).

Se creará una estructura con los tipos de datos adecuados para leer la información del archivo. Dentro de la parte de declaraciones generales de un **módulo** declararemos la estructura Título.

```
Type Registro_Título
        título as string * 255
        año as integer
        autor as long
        isbn as string * 20
        editorial as long
End Type
```

- Se utilizará un número de archivo y una variable de tipo Título en varios procedimientos, por lo que las declaramos a nivel forma en *general_declarations*.
  Dim farch as integer, libro as Registro_Título

- Definiremos una subrutina que despliegue la información en la variable libro hacia las cajas de texto. Inserte una subrutina y llámela DesplegarLibro

```
Sub DesplegarLibro ( )
   txtTítulo = Rtrim$(libro.Título)
   txtAño = libro.Año
   txtAutor = libro.Autor
   txtISBN = Rtrim$(libro.ISBN)
   txtEditorial = libro.Editorial
End Sub
```

- En el Form_Load abriremos el archivo *Títulos.Dat* y mostraremos el primer registro.

```
Sub Form_Load ( )
   farch=Freefile
   ' Incluya la ruta donde se localice Títulos.Dat
   Open "RUTA\Títulos.Dat" for Random as farch Len = Len(libro)
   Get farch, 1, libro
   Call DesplegarLibro
End Sub
```

- En el evento cmdSiguiente_Click aprovecharemos que si al utilizar la función *Get* no se determina el número de registro a leer, leerá el siguiente.

```
Sub cmdSiguiente_Click ( )
   Get farch, , libro
   If Not EOF (farch) then
         Call DesplegarLibro
   End if
End Sub
```

- En el evento cmdAnterior_Click desplegaremos el registro anterior.

```
Sub cmdAnterior_Click ( )
   IF Loc ( farch ) > 1 Then
         Get farch, Loc(farch) -1, libro
         Call DesplegarLibro
   End if
End Sub
```

```
Sub cmdGrabar_Click ( )
    libro.Título = txtTítulo.text
    libro.Año = txtAño.text
    libro.Autor = txtAutor.text
    libro.ISBN = txtISBN.text
    libro.Editorial = txtEditorial.text
    Put farch, Loc(farch), libro
End Sub
```

¡ Pruebe el ejemplo !

3.8 Laboratorio

Agregue al ejemplo B lo necesario para poder grabar, borrar y modificar registros. Cuando borramos un registro en un archivo aleatorio, sólo dejamos el registro vacío, pero el registro sigue ahí. Si desea eliminar estos registros, deberá copiar todos los registros que contengan datos a un nuevo archivo y este nuevo archivo sustituirlo por el archivo original.

Capítulo 4.

Controles dinámicos

4.1 Arreglos de controles

Un arreglo de controles es un grupo de controles que comparten el mismo tipo y nombre. Los controles dentro de un arreglo comparten las mismas subrutinas de eventos, pero cada control posee diferentes propiedades y se les identifica a través de un índice. Cada arreglo de controles puede almacenar hasta 254 controles. Para hacer referencia a un control dentro del arreglo, se debe de utilizar el nombre del arreglo y el índice entre paréntesis.

Para crear un arreglo de controles es suficiente especificar que dos controles del mismo tipo tengan el mismo nombre. En este momento *Visual Basic* confirmará si deseamos construir un arreglo de controles.

Por ejemplo: inserte un botón de comandos (command1), copiélo (CTRL+C) y péguelo (CTRL+V); en este momento *Visual Basic* nos pide la confirmación, y decimos que "sí". Tendremos un arreglo de controles llamado command1, pero sus elementos tendrán índice (la propiedad *Index*) 0 e índice 1.

Los arreglos de controles son útiles cuando queremos que varios controles compartan el mismo código, cuando no sabemos el número de controles que necesitamos, o bien, *Visual Basic* tiene un límite de 254 controles por forma, y un arreglo de controles sólo cuenta como uno hacia este límite.

A un arreglo de controles se le pueden agregar o eliminar controles en tiempo de diseño y en ejecución, es decir, trabajamos con controles dinámicos. Para agregar un control a un arreglo de controles desde código se usa la sentencia:

```
Load nombre_control(índice)
Load cmdOperación(15)    'Esto agregaría al arreglo de controles cmdOperación, el control cuyo índice
                         'será el 15.
```

El nuevo control que se genera copia todas las propiedades, excepto por las de *Visible*, *Index* y *Tabindex*, del control cuyo índice sea el más bajo, por lo general, el control con índice 0.

El índice debe ser un valor entero único dentro del arreglo, si se intenta cargar un control con un índice repetido *Visual Basic* generará un error.

Para descargar controles se utiliza la sentencia *Unload* incluyendo el nombre e índice del control a descargar.

```
Unload nombre_control(índice)
Unload cmdOperación(14)
```

No se pueden descargar controles que se hayan incluido en tiempo de diseño, no importando si forman parte de un arreglo de controles.

Una ventaja que ofrecen los arreglos de controles, es que se pueden hacer referencia a controles dentro de bucles, lo que reduce el código para realizar la misma operación con un grupo de controles.

```
Dim cont as integer
For cont% = 0  to Ubound(cmdOperación)
        cmdOperación( cont ).Enabled = False
Next cont%
```

Como todos los controles dentro de un arreglo comparten los mismos eventos, *Visual Basic* pasará el valor del índice del control que disparó un evento como parámetro adicional a los eventos. Por ejemplo, si en el arreglo de controles llamado cmdOperación con 10 elementos presionamos un botón cuyo índice sea el 7, se dispara el evento Click del arreglo cmdOperación y en el parámetro Index se tendrá el valor de 7.

```
Sub cmdOperación( Index as integer )
        Print "Se presionó el botón cuyo índice es " & Index
End Sub
```

4.2 Colección de controles

Visual Basic define automáticamente un arreglo de controles llamada *Controls* que contiene todos los controles que hay dentro de una forma. Esta colección tiene una única propiedad que es la de *Count* que contiene el número total de controles dentro de la colección.

Por ejemplo, para cambiar el tamaño de letra de todas las cajas de texto dentro de una forma podemos ejecutar el siguiente código:

```
Sub command1_click ( )
For i = 0 to Controls.Count - 1
        If TypeOf Controls( i ) is textbox then
                Controls( i ).Fontsize = 12
        End if
Next i
```

4.3 Variables objeto

Una variable objeto se utiliza para hacer referencia a objetos utilizando una variable. Al momento de declarar las variables se especifica la clase del control al que van a hacer referencia y con la sentencia *Set* se relaciona un control con una variable. Una vez que la variable está relacionada, se puede usar ésta como un alias para hacer referencia al control.

```
Dim nom as textbox
Set nom = txtNombreEmpleado
nom.Text = "Empleado 1"
```

La relación que hacemos entre la variable y un control la podemos cambiar cuando lo necesitemos.

```
Set nom=txtDirecciónEmpleado
```

Estas variables se pueden usar para hacer referencia a controles incluso en otras formas.

```
Set nom = frmAltas txtNombreEmpleado
```

Se puede declarar una variable objeto genérica, es decir, puede hacer referencia a todos las clases de controles. El uso de una variable objeto genérica es menos eficiente que el usar la clase en particular. Para declarar una variable objeto genérica se utiliza la clase "control".

```
Dim gral as control
Set gral = cmbListaNombres
Set gral = txtNombreEmpleado
Set gral = lblConsulta
```

El usar variables genéricas puede llevarnos a errores en código. Por ejemplo, si la variable "gral" la usamos para hacer referencia a diferentes clases de controles, la siguiente sentencia generará un error si la variable "gral" no apunta a un control con lista.

```
gral.Additem "¡Asegúrese que la variable objeto gral apunta hacia una lista!"
```

Como se ha mencionado anteriormente, es a veces muy útil poder manipular a los controles a través de un arreglo, pero para crear un arreglo de controles se tiene la condición de que los controles sean de la misma clase (todos *TextBox* o todos *List*, etc.). Si queremos manipular un conjunto de controles de diferente clase, lo que se puede hacer en este caso es declarar un arreglo de variables objeto.

```
'Se declaran las variables
Dim  altas(3) as control

'después se relacionan los controles con las variables
Set altas(0) = cmdAltas
Set altas(1) = lblAltas
Set altas(2) = mnuAltas
Set altas(3) = picAltas

'podemos realizar una misma operación sobre estos controles dentro de un ciclo
For i = 0 to 3
      altas( i ) Enabled = False
Next i
```

4.4 Ejemplo A. Desplegar gráficos dinámicamente.

Codificar una aplicación que muestre todos los gráficos dentro de un directorio. Como el número de archivos es variable, añadiremos dinámicamente controles Picture.

Inserte un control *Picture* (picGráfico) , un control *File List Box* (filGráficos) y un botón de comandos (cmdDesplegar)

Declare el control picGráfico como arreglo de controles para poder crear controles picGráfico dinámicamente. Modifique su propiedad Index.
picGráfico.Index = 0
picGráfico.Autosize = True

En el evento Form_Load indique que directorio va a desplegar el objeto *File List Box* y el patrón de archivos a desplegar.

```
Sub Form_Load
  filGráficos.Pattern = "*.ico"
  filGráficos.Path = "VB\icons\misc"   'Un directorio con archivos gráficos
End Sub
```

```
cmdDesplegar_Click
    Dim i As Integer
    Dim archivo As String
    Dim picActual as Picture, picAnterior as Picture
    For i = 0 To filGráficos.ListCount - 1
        If i > 0 Then    'No podemos añadir el control con índice 0. Ese ya existe.
            Load picGráfico( i )
            Set picActual = picGráfico(i)
            Set picAnterior = picGráfico(i – 1)
            picActual.Left = picAnterior.Left + 700
            picActual.Top = picAnterior.Top
            If picActual.Left + 700 > Form1.Width Then
              picActual.Left = picGráfico( 0 ).Left
              picActual.Top = picAnterior.Top + 700
            End If
        End If
        archivo = filGráficos.Path + "\" + filGráficos.List( i )
        picActual.Picture = LoadPicture(archivo)
        picActual.Visible = True
    Next i
    End Sub
```

En los eventos utilizamos la propiedad *Index* para diferenciar que control fue el que recibió el evento. Para mostrarlo. haremos que cuando el usuario presione sobre uno de los gráficos, éste se despliegue en la forma.

```
Sub picGráfico_Click (Index as integer)
  Form1.Picture = picGráfico(Index).picture
End Sub
```

¡ Pruebe el ejemplo !

Para completar: Si presionamos el botón cmdDesplegar dos veces marcará error porque los controles ya están cargados. Implemente lo necesario para que antes de cargar los controles, descargue los previamente cargados. Recuerde que no es posible descargar los controles creados en tiempo de diseño.

Capítulo 5.

Aplicaciones MDI

La interfaz de documento múltiple (*Múltiple Document Interface*) permiten presentar varias formas dentro de otra forma contenedora. Una aplicación MDI permite al usuario desplegar múltiples documentos al mismo tiempo con cada documento presentado en una ventana por separado.

Dentro de *Visual Basic* para desarrollar una aplicación MDI el primer paso debe ser incluir una forma especial que actuará como contenedora de otras formas. Esta es la forma MDI. Dentro de cada proyecto puede haber una única forma MDI o forma *Padre*. Una forma *hija* es una forma normal, pero con su propiedad *MDIChild* en verdadero. En tiempo de ejecución, estas formas hijas son desplegadas dentro de la forma MDI; cuando una forma hija se minimiza, el ícono queda dentro de la forma MDI. Las aplicaciones MDI nos sirven para cuando sea útil presentar una misma pantalla varias veces que realiza las mismas funciones con una misma interfaz y el usuario es el que modificará la información. Por ejemplo, *Excel* utiliza esta interfaz (MDI y formas hijas) para presentar las hojas de cálculo y el usuario almacena diferentes datos en las hojas de cálculo pero siempre con una misma interfaz.

Las formas hijas se pueden cargar en tiempo de ejecución y podemos desplegar las que sean necesarias. Cada forma hija es en sí, la declaración de una clase de un objeto y una instancia de esa misma clase. En tiempo de ejecución, se pueden crear nuevas instancias de una forma hija.

```
Dim nuevoDoc As New frmDocumento
nuevoDoc.Show
```

Con estas sentencias, se crea una nueva instancia de la forma "frmDocumento". La nueva instancia copia todas las características de la forma que definimos en tiempo de diseño, y contiene exactamente el mismo código y responde a los mismos eventos. La segunda sentencia es para desplegar la forma.

Cada forma hija es idéntica y dentro de código no podemos utilizar el nombre de la forma para diferenciarlas (¡todas se llaman igual!). Se puede utilizar la palabra reservada "**Me**" para casos en los que hay que hacer mención explícita a la forma. Por ejemplo, la sentencia *Unload* que recibe como parámetro el nombre de la forma que debe descargarse.

```
Unload Me
```

Cuando se ejecuta la sentencia Unload en la forma MDI, el evento *QueryUnload* ocurre primero en todas las formas hijas, si ninguna hija cancela la operación de descarga de las formas, el evento *Unload* se dispara para todas las hijas. Si ninguna forma hija cancela la operación de *Unload*, se dispara el evento de *Unload* para la forma MDI.

Si una forma hija contiene un menú, este menú reemplaza al menú de la forma MDI. El menú del padre se puede seguir accesando desde código, pero el usuario no puede disparar ningún evento del menú MDI directamente. Cuando una forma hija es maximizada, se combina la propiedad *Caption* de la forma hija con el *Caption* de la forma Padre.

5.1 Colección de formas

Cada aplicación tiene una colección "*Forms*" de todas las formas cargadas en la aplicación. La colección incluye las formas MDI, las formas MDI hijas y las formas normales. Esta colección sólo tiene la propiedad Count.

Por ejemplo, para minimizar todas las formas hijas dentro de una aplicación, podemos utilizar el siguiente código para recorrer toda las formas dentro de la aplicación:

```
Dim i as integer
For i = 0 to Forms.Count -1
        If Typeof Forms( i ) is frmDocumento then
                forms( i ).Windowstate = 1
        End if
Next i
Msgbox "Tu aplicación tiene " & Forms.Count & " formas"
```

5.2 Barra de herramientas

Dentro de las aplicaciones MDI es muy común que la forma principal (la forma MDI) despliegue una barra de herramientas con botones que realicen las funciones más comunes dentro de la aplicación y son operaciones comunes a todas las formas. Las formas MDI sólo pueden contener controles que tengan la propiedad *Align*. El control Picture tiene está propiedad que indica cuál debe ser la alineación del control dentro de las formas, por lo que utilizaremos el control Picture como contenedor de los botones para construir una barra de herramientas. (Hay controles especiales para construir más fácilmente la barra de herramientas, este ejemplo sólo es ilustrativo).

En la barra de herramientas los botones de comando que ponemos indican su función a través de un ícono. También es importante señalar que algunos botones sirven para ejecutar alguna acción o bien para indicar el estado de algo. Por ejemplo, en la mayoría de las aplicaciones si vemos un botón con un disquete podemos asumir que sirve para grabar nuestra información a disco. Un ejemplo de botones que sirven para modificar o desplegar el estado de algo pueden ser los botones de la barra de formato de *Word*, si el texto seleccionado está en **negritas** este botón se muestra presionado; para esto será necesario utilizar los controles apropiados. Los controles 'Sheridan 3D' (o el archivo threed.vbx en *Visual Basic* 3.0) son controles con interfaz de tercera dimensión; se incluyen botones de comandos con capacidad de desplegar gráficos o botones de grupo que permiten dar la impresión de que permanecen presionados.

5.3 Ejemplo A. Barra de herramientas.

Construcción de una barra de herramientas. Desde la barra de herramientas agregaremos ventanas hijas a nuestro proyecto.

Añada una forma MDI a su proyecto (frmPadre), un módulo (Mdi.bas) y tenga dos formas normales (frmHijo y frmHija).

Agregue los controles Sheridan 3D (o el control THREED.VBX) a su proyecto.

Añada un botón de comandos (cmdSalir) a frmHijo y a frmHija.

Indique que las formas frmHijo y frmHija serán formas hijas.
frmHijo.MDIChild = True
frmHija.MDIChild = True

Cambie el color de fondo de la frmHijo a azul y la frmHija a rosa, el ícono que se despliega cuando se minimicen las formas y el gráfico de fondo de las formas.
frmHijo.Backcolor = &H00FF0000
frmHijo.Icon = "**VB**\icons\misc\misc27.ico"
frmHijo.Picture = "**VB**\icons\misc\misc27.ico"
frmHija.Backcolor = &H00FF00FF
frmHija.Icon = "**VB**\icons\misc\misc27.ico"
frmHija.Picture = "**VB**\icons\misc\misc27.ico"

Cambie la forma de inicio de su proyecto por la forma frmPadre. Puede especificar que su proyecto inicie de una forma hija. en este caso, al ejecutar su aplicación se cargarán la forma MDI y la forma hijo que haya especificado.

Inserte un control *Picture* (picBarra) a la forma MDI y ponga su propiedad de *Backcolor* en gris.

Inserte dos botones *SSCommand* (cmdMostrarHijo y cmdMostrarHija) en el control picBarra.

A los botones cmdMostrarHijo y cmdMostrarHija les especificaremos el ícono a desplegar y borraremos su propiedad *Caption*.
cmdMostrarHijo.Picture = "**VB**\icons\misc\misc27.ico"
cmdMostrarHijo.Caption = ""
cmdMostrarHija.Picture = "**VB**\icons\misc\misc26.ico"
cmdMostrarHijo.Caption = ""

```
Sub cmdMostrarHijo_Click ( )
   Static numHijo as integer
   numHijo = numHijo + 1
   Dim nuevoHijo as New frmHijo
   nuevoHijo.Caption = "Hijo # " & numHijo
   nuevoHijo.Show
End Sub
```

Complete el código para que se muestre una forma Hija cuando se presione el botón cmdMostrarHija y se cierre la forma si se presiona el botón cmdSalir.

Desde la barra de herramientas se realizan las operaciones comunes entre las formas hijas, pero ¿cómo saber cuál es la forma activa? Por ejemplo, si en la barra de herramientas presiono el botón de "Grabar" debo de determinar sobre que forma está trabajando el usuario para grabar la información respectiva.

Si el código se disparará desde una de las formas hijas, podríamos utilizar la sentencia *Me* para pasar como argumento una referencia de la forma que dispara cierta acción.

Agregue la subrutina CambiaColor en el Módulo. Esta subrutina recibirá como argumento una referencia a la forma que debe de cambiarle el color.

```
Sub CambiaColor ( formaY as form )
   formaY.Backcolor = RGB(255,255,255)
End Sub
```

Para invocar la subrutina, utilice el evento DblClick de las formas (frmHija y frmHijo)

```
Sub Form_DblClick ( )
   Call CambiaColor(Me)
End Sub
```

¡ Pruebe ! Haga doble *click* en las formas y el fondo debe cambiar a blanco.

Pero en el caso de las barras de herramientas, el botón de comando está sobre la forma MDI, por lo que no podemos obtener de esta manera una referencia de la forma hija. Sin embargo, la forma MDI tiene una propiedad que nos indica cuál de sus formas hijas está activa. Con esta propiedad obtenemos una referencia hacia la forma hija y así podemos trabajar con la forma hija activa. Esta propiedad es *ActiveForm*.

Inserte un botón SSCommand (cmdCuadricula) en picBarra.

```
cmdCuadricula.Picture = "VB\icons\office\rulers.ico"
```

```
Sub cmdCuadricula_Click ( )
    Dim i As Long, formaX As form
    Set formaX = frmPadre.ActiveForm      'Obtenemos una referencia a la
                                          'forma hija activa
    For i = 0 To formaX.ScaleWidth Step 500
        formaX.Line ( i , 0)-( i , formaX.ScaleHeight)
    Next i
    For i = 0 To formaX.ScaleHeight Step 500
        formaX.Line (0, i )-(formaX.ScaleWidth, i )
    Next i
```

¡ Pruebe su ejemplo! Al presionar el botón de cuadrícula cuadriculará la forma hija activa.

Capítulo 6.
Procesamiento de errores en tiempo de ejecución

El procesar los errores que se presentan en tiempo de ejecución nos permite presentar un programa robusto que no "truene" cuando se presenta un error. *Visual Basic* antes de compilar un programa verifica que no haya errores en la sintaxis del lenguaje y trata en lo posible de asegurar que todas las referencias a objetos sean válidas, pero no puede prever los errores que se pueden presentar en nuestra aplicación. Por mencionar algunos ejemplos, ¿qué pasa si el programa intenta grabar un archivo al disco duro y el disco duro está lleno?, ¿qué pasa si el archivo a leer se encuentra en la red y en ese momento la red no está disponible?, sin el procesamiento de errores nuestra aplicación le presenta un mensaje de error al usuario y termina su ejecución con la posible pérdida de información.

Con *Visual Basic* podemos atrapar los errores en tiempo de ejecución para determinar que debe hacer nuestra aplicación: pedirle ayuda al usuario, cerrar los programas y terminar, tratar de corregirlo automáticamente, ignorarlo, etc.

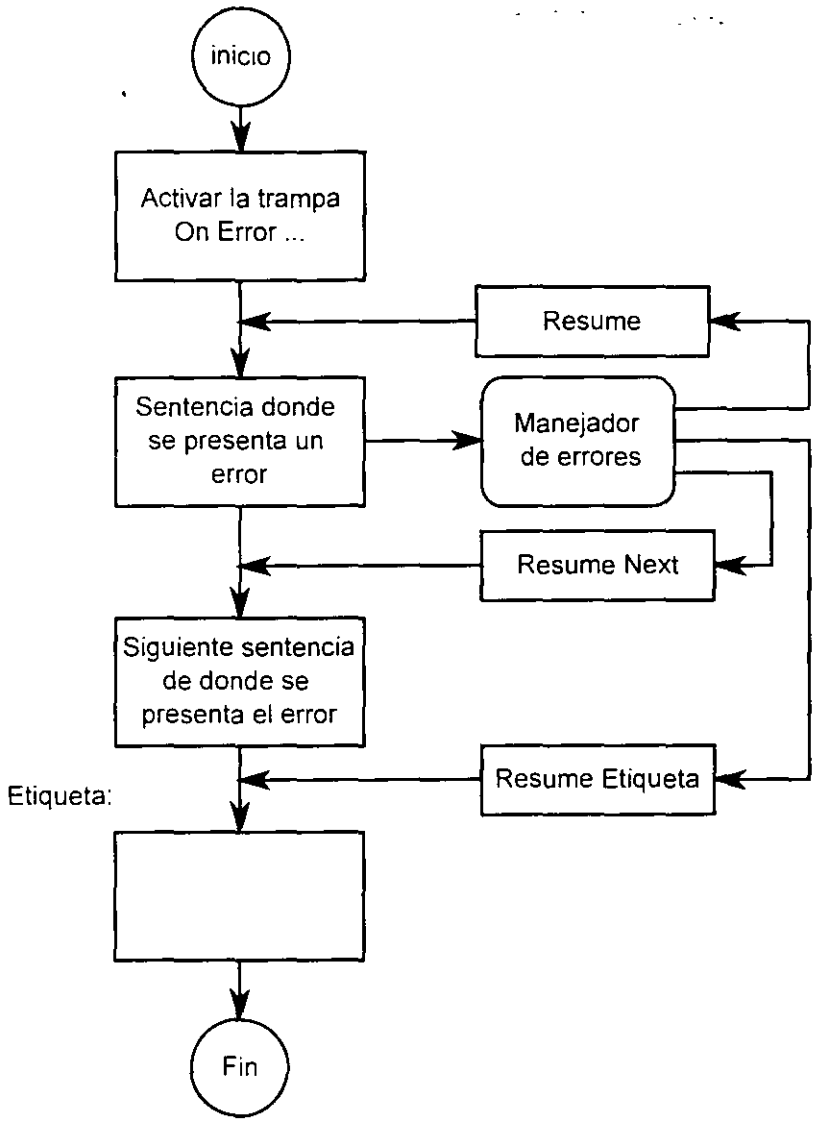
6.1 Manejo de errores en tiempo de ejecución

Para que *Visual Basic* atrape los errores en tiempo de ejecución se utiliza la sentencia: *On error* <acción a realizar>. Esto se le conoce como activar la trampa. Cuando *Visual Basic* detecta un error busca si se tiene una sentencia *On error* que le indique que acción debe de tomar. Como acciones a tomar se pueden especificar:

- *On error Resume Next*
  La acción "*Resume*" le indica que continúe la ejecución del programa en la siguiente línea a partir de donde se presentó el error.

- *On error goto* <etiqueta>
  La acción "*goto* <etiqueta>" le indica que cuando se presente un error brinque a la línea de código especificada por <etiqueta>. Esta sentencia nos permite implementar un manejador de errores, es decir, podemos escribir código que nos permita procesar el error.

El flujo que puede seguir nuestro procesamiento de errores es el que se muestra en el siguiente diagrama:



Las trampas para errores se especifican en cada subrutina que queramos atrape errores en tiempo de ejecución. Si se presenta un error en una subrutina que no tenga manejadores de errores, el error se pasa automáticamente por la lista de llamadas a funciones, si en la lista a llamadas a funciones no hay ningún manejador de errores, se despliega un error al usuario y la aplicación termina.

La lista de llamadas es todas las llamadas a funciones o subrutinas pendientes que tiene *Visual Basic* al llegar hasta nuestra actual línea de código. Considere el siguiente ejemplo:

Inserte un botón de comandos (cmdEjecutar)

Inserte tres subrutinas (procA, procB y procC)

Pondremos el código para el botón de comandos y para las tres subrutinas.

```
Sub cmdEjectuar ( )
  Call ProcA
End Sub

Sub ProcA ( )
  Print "Estoy en el procedimiento A"
  Call ProcB
  Print "Saliendo del procedimiento A"
End Sub

Sub ProcB ( )
  Print "Estoy en el procedimiento B"
  Call ProcC
  Print "Saliendo del procedimiento B"
End Sub

Sub ProcC ( )
  On Error Resume Next
  Dim i as integer
  Print "Estoy en el procedimiento C"
  i = i / 0
  Print "Saliendo del procedimiento C"
End Sub
```

Pruebe su ejemplo. Al presionar el botón de comandos tendrá algo similar a esto en la pantalla:

```
Estoy en el procedimiento A
Estoy en el procedimiento B
Estoy en el procedimiento C
Saliendo de C
Saliendo de B
Saliendo de A
```

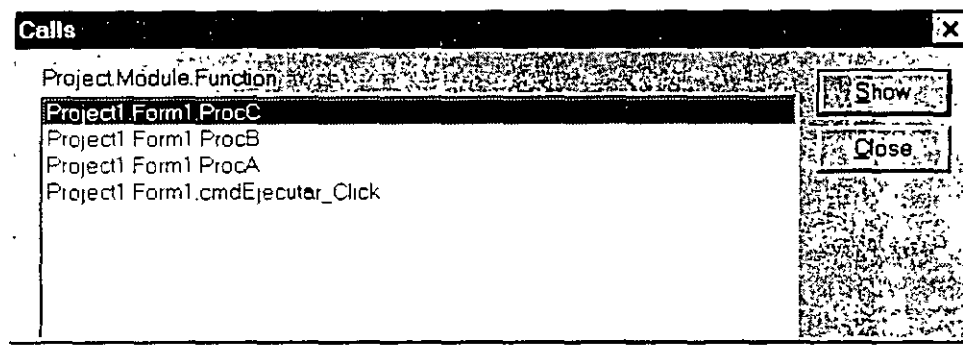
Ahora modifique su procC de la siguiente manera (incluya la cláusula *Stop*):

```
Sub ProcC ( )
  On Error Resume Next
  Dim i as integer
  Print "Estoy en el procedimiento C"
  Stop
  i = i / 0
  Print "Saliendo del procedimiento C"
End Sub
```

Pruebe su ejemplo. La sentencia *Stop* detiene la ejecución del programa y nos deja en modo de depuración. En su barra de herramientas tendrá activado el botón de "Llamadas a funciones."

También puede desplegar las llamadas de funciones con el menú *Calls* de su menú herramientas o del menú de Depuración.

Presione este botón y se desplegarán la lista de llamadas:



Esta ventana nos dice que en las llamadas a funciones pendientes, la primera llamada a función fue la de cmdEjecutar_Click, después sigue ProcA, después ProcB y la más reciente es el ProcC.

Ahora agregaremos una trampa de errores al ProcA y quitaremos de ProcC la cláusula de *Stop* y su trampa de errores. Su código quedará de la siguiente manera:

```
Sub ProcA ( )
  On Error Resume Next
  Print "Estoy en el procedimiento A"
  Call ProcB
  Print "Saliendo del procedimiento A"
End Sub
```

```
Sub ProcC ( )
  Dim i as integer
  Print "Estoy en el procedimiento C"
  i = i / 0
  Print "Saliendo del procedimiento C"
End Sub
```

- Si ahora prueba su aplicación, en su forma tendrá algo como lo siguiente:

```
Estoy en el procedimiento A
Estoy en el procedimiento B
Estoy en el procedimiento C
Saliendo de A
```

Sucede que al dispararse un error en la subrutina *ProcC* y no haber trampa de errores, *Visual Basic* revisa las llamadas a funciones y va "pasando" el error. En nuestra lista de funciones la llamada anterior a la subrutina *ProcC* es la subrutina *ProcB*, por lo que pasa el error a *ProcB*. En *ProcB* tampoco tenemos trampa de errores y el error pasa a *ProcA*. En esta subrutina si tenemos manejador de errores y la acción a ejecutar es *Resume Next*. Pero para la subrutina *ProcA* la sentencia que provocó el error fue la llamada al procedimiento *ProcB* (como para *ProcB* el error lo generó la llamada a *ProcC*).

```
                                Sub ProcA ( )
                                  On Error Resume Next
                                  Print "Estoy en el procedimiento A"
Aquí se generó el error para ProcA →  Call ProcB
                                  Print "Saliendo del procedimiento A"
                                End Sub
```

Sin una muy buena adecuada planeación, este paso del error por las llamadas a funciones puede hacer muy difícil de prever o entender como estarían trabajando nuestros manejadores de errores y dónde hacer que continúe o dónde está continuando la ejecución de mi programa. Entonces, puede ser conveniente habilitar un manejador de errores en cada función que nos interese procesar los errores que se presenten.

Todos los manejadores de errores deben contener forzosamente la cláusula *Resume* o *End* para que *Visual Basic* sepa que hacer en cuanto termine el manejador de errores, en caso contrario, recibimos un error. Si se presenta un error dentro del manejador de errores, *Visual Basic* automáticamente pasa el error por la lista de llamadas de funciones.

Cuando se presenta un error en la aplicación, *Visual Basic* inicializa la variable *Err%* con un número asociado al error y la variable *ErrorS* con una cadena descriptiva asociada al error (si existe esta cadena descriptiva). Estos valores almacenados en *Err%* y *ErrorS* nos sirven para determinar el tipo de error que se presentó y como tratar de solucionarlo.

Con la ayuda que trae *Visual Basic* podemos ver los errores que podemos atrapar. En su archivo de ayuda puede buscar por "Error" o bien por "Trappable Errors", o "Errores Procesables".

Por ejemplo, en el evento click de un botón de comando tenemos el siguiente código:

```
Sub cmdGrabar_Click( )
    On error goto ManejadorErrores    ' Habilitar la trampa de errores
    Dim respuesta as integer, mensaje as string, título as string
    Const Exclamación = 48, Interrogación = 32, Info = 64,  ReintentarCancelar = 5, Reintentar = 4
    Filecopy "c:\autoexec.bat" , "a:\"   ' sentencia en la que se puede presentar un error
    Msgbox "El archivo se copió satisfactoriamente" , 64, "Copia de archivos"
    NoCopiar:            ' etiqueta con la cuál brincamos un grupo de expresiones
    Exit Sub             ' Salir de la subrutina
    ManejadorErrores:    ' etiqueta donde brincará el flujo del programa si hay un error
    Select Case Err
        Case 55:
            mensaje = "El archivo está en uso, cierre el archivo antes de copiarlo"
            título = "Error en la copia del archivo"
            Msgbox mensaje, Exclamación , título
            Resume NoCopiar
        Case 71:
            mensaje = "La unidad no está lista, verifique que tenga un disco adentro."
            título = "Error en la copia del archivo"
            respuesta = Msgbox( mensaje, Interrogación + ReintentarCancelar, título )
            If respuesta = Reintentar then
                Resume
            Else
                Resume NoCopiar
            End if
        Case Else
            mensaje = "No se completó la operación " + chr$(13) + Str$(Err) + chr$(13) + Error$
            título = "Error no previsto"
            Msgbox mensaje, Exclamación, título
            Resume NoCopiar
    End Select
End Sub
```

En el ejemplo anterior, lo primero es activar la trampa, con esto le indicamos que cuando se presente un error el flujo del programa pase a la etiqueta "ManejadorErrores". Es importante recalcar que la etiqueta debe estar dentro de la misma función o subrutina. En nuestro ejemplo, la función *Filecopy* puede presentar errores al ejecutarse que no dependen del código, dependen de factores externos que el programador no puede controlar.

Si una trampa para errores se activo en un procedimiento, es automáticamente desactivada cuando este procedimiento termina. Para desactivar la captura de errores dentro de una subrutina se utiliza la sentencia:

```
On Error Goto 0
```

6.2 Laboratorio

Desarrolle las trampas de error necesarias para el manejo de una base de datos. Detecte los errores que pueden ocurrir al grabar un registro a la base de datos (índices vacíos, índices repetidos, tipos de datos incorrecto, disco lleno, etc.) y escriba las rutinas necesarias para corregir estas situaciones.

2.0 Fundamentos de la programación de bases de datos.

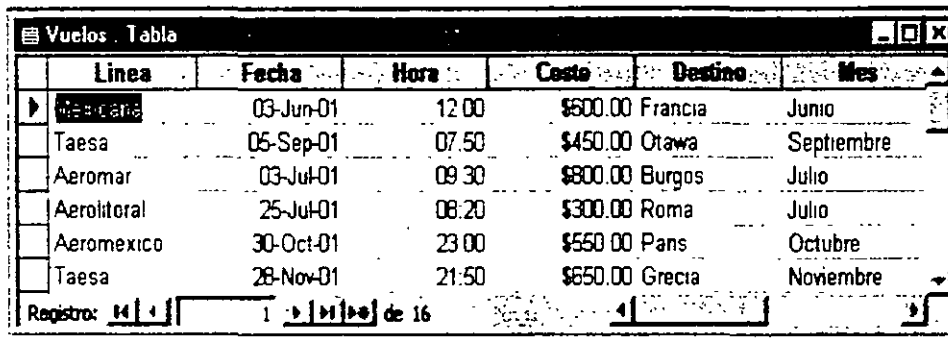
2.1 ¿Qué es una base de datos?

Una base de datos es una colección de datos clasificados y estructurados que son guardados en uno o varios archivos pero referenciados como si de un único archivo se tratara. En la versión 6 de Visual Basic, que es la que nosotros estudiaremos, incluye un Administrador visual de datos que nos permitirá crear y manipular bases de datos *Microsoft Access, Dbase, Foxpro, Paradox, ODBC* y archivos de texto.

Los datos de una base de datos están compuestas por campos y registros. El conjunto de todos los registros forman la base de datos. Una base de datos puede estar formada por una o más *tablas*.

Una tabla es una colección de datos presentada en forma de una matriz bidimensional, donde las filas son los *registros* y las columnas los campos.

Figura 2.1 Tabla de una base de datos



En la figura se puede ver una tabla compuesta por los siguientes campos `Linea, Fecha, Hora, Costo, Destino, Mes.` Así un campo seria Mexicana, y un registro podria ser:

```
Taesa 05-Sep-01 07:50 $600.00 Otawa Septiembre
```

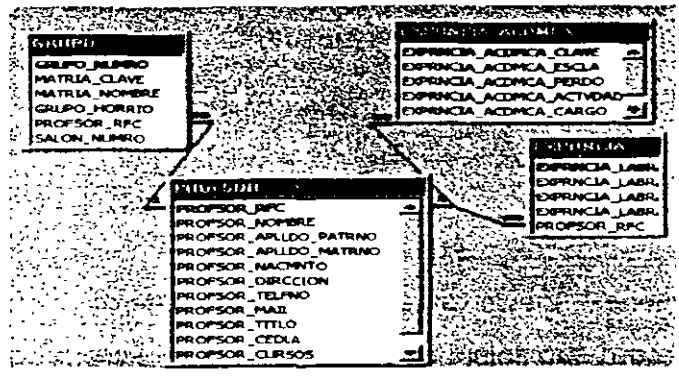
2.2 PROGRAMACIÓN DE BASES DE DATOS

Por lo general en aplicaciones fuertes se vuelve necesario almacenar información, de manera permanente, para esto nos pueden servir los archivos, sin embargo cuando se necesita llevar un orden mas rigido, o bien es necesario hacer consultas de la información almacenada de una manera rápida y segura, no hay nada mejor que una base de datos para almacenar nuestra información.

Las bases de datos actuales, en general, son relacionales. El acceso a

las mismas se hace generalmente utilizando el lenguaje SQL *(Structured Query Language* - Lenguaje de consulta estructurado).

Figura 2.2 Ejemplo de una base de datos relacional



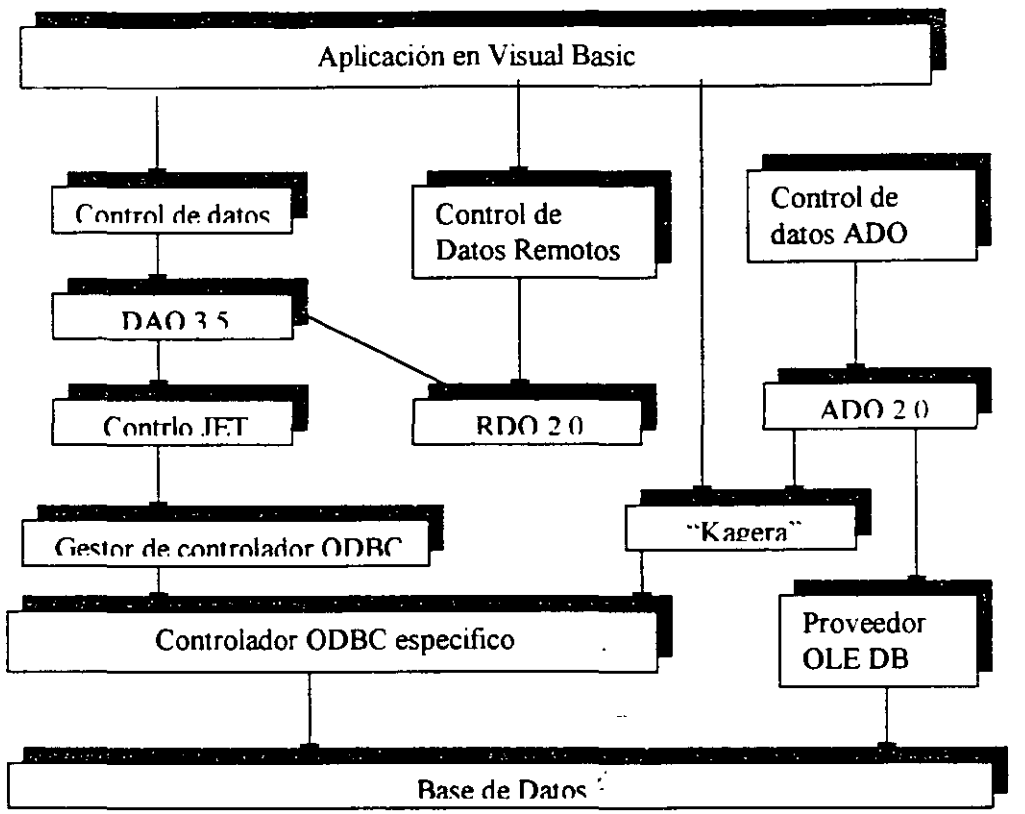
Por otra parte, dada la gran cantidad de situaciones diferentes en las que se puede utilizar una base de datos, es lógico que existan muchos tipos y muchas formas de acceder a ellas. Por eso, para que los programadores puedan acceder de forma estándar a una base de datos, los fabricantes suelen desarrollar junto con la base de datos el controlador ODBC *(Open Database Connectivity* - Conectividad abierta de bases de datos) de acceso correspondiente, que proporciona al programador un conjunto de funciones estándar (API - *Application Programming Inteiftice;* Interfaz de programación de aplicaciones) para acceder al motor de la base.

Algunos otros fabricantes, dotaron a sus herramientas de desarrollo de una nueva capa de software, conocida como motor de bases de datos, intercalada entre el código de la aplicación y el controlador ODBC. Un ejemplo es MS Jet, que mas adelante estudiaremos. El motor de base de datos que proporciona Microsoft en muchos de sus productos (Visual Basic, Visual C++, Excel, Word, etc.). Se trata de una capa de software independiente de la aplicación que lo utiliza para acceder a los datos de la base.

Finalmente, en las herramientas de desarrollo actuales contamos con los controles y los objetos de acceso a datos, los cuales establecen un puente entre la interfaz del usuario y el motor de acceso a datos.

Visual Basic nos proporciona varias formas de acceso a bases de datos remotas, las cuales se describen a continuación:

2.3 Controles para acceso a bases de datos.



Frfgura 2.4 esquema de los diferentes caminos para un enlace a una base de datos.

- Objetos ActiveX *de acceso a datos* (ADO - *ActiveX Data Objects).* Microsoft, a partir de la versión 6 de Visual Basic, introduce el modelo de objetos ADO para el acceso a bases de datos. Este modelo es más sencillo y proporciona mejor integración con las tecnología de Microsoft y con otras tecnologías, una interfaz común para acceso a datos locales y remotos, conjuntos de registros remotos y locales, una interfaz de enlace con los datos accesible para el usuario y un conjuntos de registros jerárquicos.

- Control de datos ADO (ADODC - *ADO Data control).* Permite crear una conexión con una base de datos de una forma fácil y rápida mediante objetos *ActiveX* de acceso a datos.
- Objetos de acceso a datos (DAO Data Access Objects). El modelo de obje tos DAO admite dos entornos diferentes de bases de datos o espacios de tra bajo: *MS Jet* y *acceso directo a ODBC (ODBCDirect).*

Microsoft Jet permite acceder a bases de datos Microsoft *Access,* o bien a bases de datos Microsoft conectadas a ODBC y a orígenes de datos conectados a ISAM *(Indexed Sequential Access Method* - Método de acceso secuencial indexado) para permitir el acceso a formatos de bases de datos externas como son dBASE, Microsoft Excel y Paradox. *MS Jet* carga estos controladores ISAM cuando se hace referencia a ellos en una aplicación.

El *acceso directo a ODBC (ODBCDirect)* permite tener acceso a servidores de bases de datos a través de ODBC sin cargar el motor de base de datos *MS*

- *Control de datos* (DC - *Data Control).* El control *Data* implementa el acceso a los datos mediante el motor de bases de datos *MS Jet,* el mismo motor de bases de datos de *Microsoft Access.*

- *RDO.* Es una interfaz de acceso a datos mediante ODBC orientada a objetos, que incorpora un estilo sencillo de DAO y cuya interfaz expone prácticamente toda la flexibilidad y eficacia de bajo nivel de ODBC. Sin embargo, RDO presenta limitaciones al no proporcionar un acceso apropiado a las bases de datos *MS Jet,* y al sólo permitir el acceso a bases de datos relacionales a través de los controladores ODBC existentes. El control remoto de datos (RDC - *RemoteData Control)* implementa el acceso mediante RDO.

- *Conectividad abierta de bases de datos* (ODBC - Open Database Connectivity). Se trata de una interfaz de programación para acceso a servidores de base de datos que proporciona un lenguaje común para las aplicaciones Windows que necesiten acceder a una base de datos en una red. Para utilizar

esta interfaz con distintas bases de datos debe instalar previamente los correspondientes controladores ODBC.

- *VB SQL.* Es una biblioteca de Visual Basic para *SQL Server.* Concretamente se trata de una interfaz de programación (API) para *DB-Library.*

Básicamente las ventajas dependen del tipo de usuario para el que se realice la aplicación.

El motor *Jet* funciona bien en monousuario y en redes pequeñas; tiene sus problemas de bloqueo cuando varios usuarios acceden a una misma información o incluso a informaciones colindantes.

Por esta razón cuando trabajemos con bases de datos grandes es aconsejable trabajar con ODBC. La ventaja de trabajar con el motor *Jet* es que no necesitamos depender de ningún *driver* ODBC externo, que las aplicaciones son más sencillas y dan menos problemas de configuración.

Utilizar RDO es similar en muchos aspectos a utilizar DAO; no obstante hay algunas diferencias, ya que RDO está implementado y diseñado para utilizarlo estrictamente con bases de datos relacionales. RDO no tiene ningún procesador de consultas propio; depende del origen de datos para procesar todas las consultas y crear los conjuntos de resultados. Los objetos de datos propiamente dichos se generan a partir de los conjuntos de resultados y los cursores devueltos por el controlador ODBC.

Finalmente, el acceso a datos basado en OLE DB y ADO es adecuado para una gama amplia de aplicaciones cliente/servidor. Las principales ventajas son fácil utilización, gran velocidad, uso de poca memoria y poca utilización de disco.

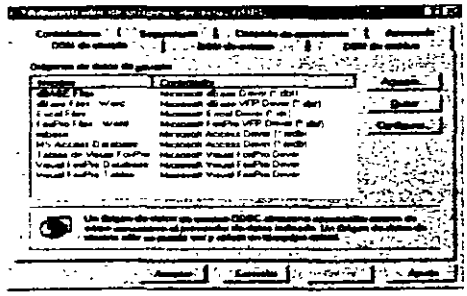
Recordemos, como principio de diseño, que aunque la tecnología posibilite múltiples y enrevesadas posibilidades, las aplicaciones que deben funcionar cada día deben ser lo más sencillas posibles, tanto para el usuario como para el programador que debe soportarlas. Por lo tanto, el modelo de objetos ADO es el que se aconseja utilizar de ahora en adelante.

2.4 Conexión ODBC

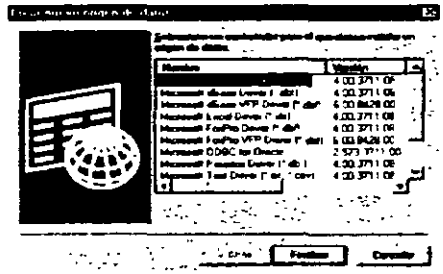
Para poder utilizar una base de datos desde ASP esta debe estar dada de alta para emplearla con controladores ODBC.

Vamos a ver como hacer esta conexión paso a paso.

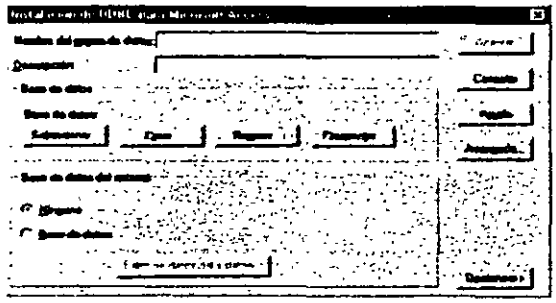
Lo primero será ir a **Inicio -> Configuración -> Panel de Control ->Fuentes ODBC.** Entonces estaremos en una pantalla como la siguiente:



En esta pantalla se encuentran algunos de los controladores para los diferentes tipos de bases de datos que soportan controladores ODBC. Estando ahí damos clic en el botón Agregar..., y se nos mostrará una pantalla como la siguiente:



En ella seleccionamos el tipo de base de datos que tenemos, es decir Acces, FoxPro, Oracle, etc. Y damos clic en el botón Finalizar, entonces estaremos en la siguiente pantalla. En ella tendremos que hacer clic en el botón que dice Seleccionar, en el recuadro que dice Base de Datos. Para que se nos muestre una ventana para seleccionar la base de datos que deseamos agregar.



Después le damos un nombre, en la primera caja de dialogo donde dice: Nombre del origen de datos, con este nombre nosotros haremos la conexión a la base de datos, no tiene que ser el mismo nombre de la base de datos.

Entonces damos clic en Aceptar, y nos regresara a la primera pantalla en ella ahora podremos ver el nombre del origen de datos que le dimos a la conexión acompañado de los controladores para el tipo de base de datos que tenemos. Ahora nuestra base de datos esta lista para ser utilizada desde ASP. Si ahora vamos al explorador de Windows podremos ver que un hay un icono que nos indica que la conexión se realizo satisfactoriamente. En la misma carpeta donde se encuentre la base de datos sobre la que vamos a trabajar encontraremos un icono parecido a esto:

2.5 Objetos ADO

El modelo de bases de datos basado en Actives Data Objects, está formado por objetos, estos objetos proporcionan una serie de métodos y propiedades con los que podemos acceder fácilmente a las bases de datos .

Para manejar este tipo de bases de datos tenemos siete objetos. De estos siete objetos hay que ver con especial cuidado a tres de ellos que son los mas importantes Connection, Recordset y Command. El resto de los objetos son Field, Parameter, Property y Error, sin embargo para poder emplear estos últimos necesitamos de los primeros.

La descripción de cada uno de los objetos seria:

- Connection: Representa la conexión con una base de datos. Este objeto lo utilizaremos para crear un enlace directo entre nuestra página Web y el servidor de bases de datos. Mientras dure la conexión podremos realizar todas las operaciones que deseemos sobre la base de datos. La conexión terminará cuando nosotros así lo indiquemos con el método Close del mismo objeto.
- RecordSet: representa una tabla de datos. En este objeto serpá donde almacenemos las consultas realizadas a la base de datos a la que

estemos conectados. Estará formada por filas, y por columnas a los que podremos acceder para exponer la información adquirida.

- Command: Representa un comando SQL. Con este objeto podremos ejecutar sentencias SQL sobre la base de datos a la que estemos conectados.

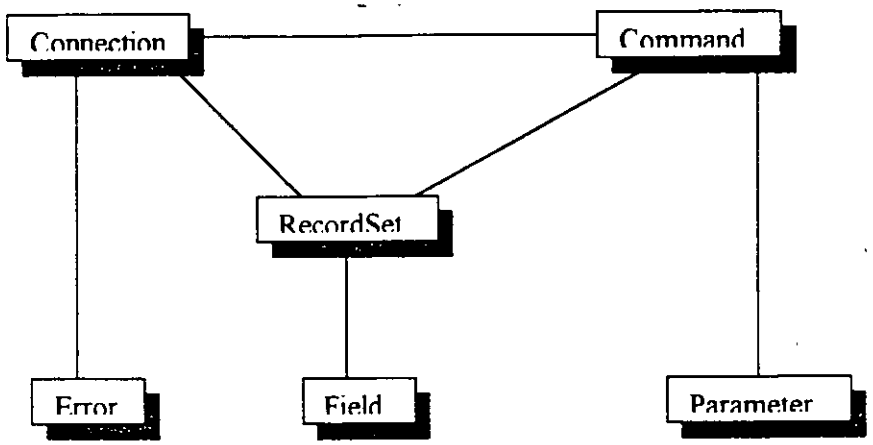
- Field: Representa un campo de un objeto RecordSet. Este objeto solo existirá si existe el correspondiente objeto RecordSet. El objeto RecordSet lleva implícito la colección Fields que representa todos sus campos. Cada elemento de esa colección es un objeto Field.

- Parameter: Representa un parámetro de un procedimiento o cuestión. Nos será de gran ayuda al utilizar el objeto Command para lanzar procedimientos o cuestiones sobre una base de datos.

- Error: Representa un error ADO. Se puede producir un error al realizar la conexión sobre la base de datos. Esto quiere decir que este objeto solo existirá si previamente se ha intentado conectar con una base de datos erróneamente.

- Property: Representa una propiedad específica de un proveedor de datos. Este objeto se encuentra un poco al margen de lso demás, ya que no tiene relación alguna con ellos.

Como se puede suponer entre estos objetos existe cierta relación.



Relaciones:

Connection-RecordSet: Una cualidad que tiene el objeto Connection es la de poder ejeutar comandos SQL. Al hacer esto pueda darse el caso de que el comando devuelva cierto resultado, por ejemplo, si el comando realiza una consulta sobre la base de datos. En tal caso el resultado vendrá dado dentro de un objeto RecordSet.

Command-RecordSet: Al igual que en el caso anterior, al ejecutar una consulta con el objeto Command sobre una base de datos el resultado viene dado en un objeto RecordSet.

Command-Connection: Cuando creamos un objeto Command no será utilizable hasta que no lo relacionamos con una base de datos. Para esto es necesario el objeto Connection. Con este objeto abriremos una sesión con una base de datos y entonces se lo asignaremos al objeto Command como conexión activa. Solo de este modo podremos ejecutar los camandos de este objeto sobre una base de datos.

Connection-Error: El objeto Error solo aparece cuando se produce una conexión errónea a una base de datos. Con este objeto y sus métodos y propiedades podremos determinar la causa del error.

RecordSet-Field: Todo objeto RecordSet tiene entre sus atributos la colección Fields. Esta colección esta formada por objetos del tipo Field. Estos objetos representan ada campo de cada registro del objeto RecordSet. El objeto Field nos permitirá un acceso sencillo a la información del objeto RecordSet.

Command-Parameter: Un comando SQL puede tener parámetros. Esos parámetros pertenecerán al objeto Parameter. Además no sólo podemos ejecutar sentencias simples de SQL con el objteo Command, sino también procedimientos que probablemente llevarán parámetros, tanto de entrada como de salida.

6. Selección de datos mediante SQL

6.1 Introducción

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft Jet. SQL se utiliza para crear objetos QueryDef, como el argumento de origen del método OpenRecordSet y como la propiedad RecordSource del control de datos. También se puede utilizar con el método Execute para crear y manipular directamente las bases de datos Jet y crear consultas SQL de paso a través para manipular bases de datos remotas cliente - servidor.

6.2 Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- Los DLL que permiten crear y definir nuevas bases de datos, campos e índices.
- Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL:

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices

ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML:

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados
DELETE	Utilizado para eliminar registros de una tabla de una base de datos

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo

ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

Operadores lógicos

Operador	Uso
AND	Es el "y" lógico. Evalua dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdar si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Consultas de Selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

Consultas Básicas

La sintaxis básica de una consulta de selección es la siguiente:

SELECT Campos FROM Tabla;

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

SELECT Nombre, Telefono FROM Clientes;

Esta consulta devuelve un recordset con el campo nombre y teléfono de la tabla clientes.

Ordenar los Registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER
BY Nombre;
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY
    CodigoPostal, Nombre;
```

Incluso se puede especificar el orden de los registros: ascendente mediante la claúsula ASC (se toma este valor por defecto) ó descendente (DESC).

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY
    CodigoPostal DESC , Nombre ASC;
```

Consultas con Predicado

El predicado se incluye entre la claúsula y el primer nombre del campo a recuperar, los posibles predicados son:

Predicado	Descripción
ALL	Devuelve todos los campos de la tabla
TOP	Devuelve un determinado número de registros de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente
DISTINCTROW	Omite los registros duplicados basandose en la totalidad del registro y no sólo en los campos seleccionados.

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No se conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados. SELECT ALL FROM Empleados; SELECT * FROM Empleados;

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

SELECT TOP 25 Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes. El predicado TOP no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un

rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

```
SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes ORDER
BY Nota DESC;
```

El valor que va a continuación de TOP debe ser un Integer sin signo. TOP no afecta a la posible actualización de la consulta.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.

Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT Apellido FROM Empleados;
```

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

DISTINCTROW

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campo indicados en la cláusula SELECT.

```
SELECT DISTINCTROW Apellido FROM Empleados;
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López, el ejemplo del predicado DISTINCT devuleve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados;
```

Bases de Datos Externas

Para concluir este capítulo se debe hacer referencia a la recuperación de registros de bases de datos externa. En ocasiones es necesario la recuperación de información que se encuentra contenida en una tabla que no se encuentra en la base de datos que ejecutará la consulta o que en ese momento no se encuentra abierta, esta situación la podemos salvar con la palabra reservada IN de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados
IN 'c:\databases\gestion.mdb';
```

En donde c:\databases\gestion.mdb es la base de datos que contiene la tabla Empleados

Criterios de Selección

En el capítulo anterior se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de este capítulo se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan una condiciones preestablecidas.

Antes de comenzar el desarrollo de este capítulo hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda es que no se posible establecer condiciones de búsqueda en los campos memo y; la tercera y última hace referencia a las fechas. Las fechas se deben escribir siempre en formato mm-dd-aa en donde mm representa el mes, dd el día y aa el año, hay que prestar atención a los separadores -no sirve la separación habitual de la barra (/) - hay que utilizar el guión (-) y además la fecha debe ir encerrada entre almohadillas (#). Por ejemplo si deseamos referirnos al día 3 de Septiembre de 1995 deberemos hacerlo de la siguente forma; #09-03-95# ó #9-3-95#.

Operadores Lógicos

Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not. A excepción de los dos últimos todos poseen la siguiente sintaxis:

<expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso
Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso
Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad

Null	Imp	Falso	Null
Null	Imp	Null	Null

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.

El último operador denominado IS se emplea para comparar dos variables de tipo objeto <Objeto1> Is <Objeto2>. Este operador devuelve verdad si los dos objetos son iguales.

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo
= 100;
SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR
        (Provincia = 'Madrid' AND Estado = 'Casado');
```

Intervalos de Valores

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

campo [Not] Between valor1 And valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;
(Devuelve los pedidos realizados en la provincia de Madrid)
SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial', 'Nacional')
FROM Editores;
(Devuelve el valor 'Provincial' si el código postal se encuentra en el intervalo,
```

'Nacional' en caso contrario)

El Operador Like

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

expresión Like modelo

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador Like para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (Like An*).

El operador Like se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce Like C* en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

Like 'P[A-F]###'

Este ejemplo devuelve los campos cuyo contenido empiece con una letra de la A a la D seguidas de cualquier cadena.

Like '[A-D]*'

En la tabla siguiente se muestra cómo utilizar el operador Like para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

El Operador In

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:

expresión [Not] In(valor1, valor2, . . .)

SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');

La cláusula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Depués de escribir esta cláusula se deben especificar las condiciones expuestas en los dos primeros apartados de este capítulo. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM.

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario > 21000;
    SELECT Id_Producto, Existencias FROM Productos
    WHERE Existencias <= Nuevo_Pedido;

SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;
  SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos = 'King';

  SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos Like 'S*';
  SELECT Apellidos, Salario FROM Empleados WHERE Salario Between
  200 And 300;

SELECT Apellidos, Salario FROM Empl WHERE Apellidos Between 'Lon' And
'Tol';
SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE Fecha_Pedido
    Between #1-1-94# And  #30-6-94#;

SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE Ciudad
    In ('Sevilla', 'Los Angeles', 'Barcelona');
```

Agrupamiento de Registros y Funciones Agregadas

La cláusula GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupoGROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no esta incluido en la instrucción SELECT, siempre y cuando la instrucción SELECT incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

SELECT Id_Familia, Sum(Stock) FROM Productos GROUP BY Id_Familia;

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

```
SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia
        HAVING Sum(Stock) > 100 AND NombreProducto Like BOS*;
```

AVG (Media Aritmética)

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente:

```
Avg(expr)
```

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;
```

Count (Contar Registros)

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

```
Count(expr)
```

En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*) AS Total  FROM Pedidos;
```

Si expr identifica a múltiples campos, la función Count cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

```
SELECT Count(FechaEnvío & Transporte) AS Total FROM Pedidos;
```

Max y Min (Valores Máximos y Mínimos)

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

```
Min(expr)
Max(expr)
```

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Min(Gastos) AS ElMin FROM Pedidos WHERE Pais = 'España';
SELECT Max(Gastos) AS ElMax FROM Pedidos WHERE Pais  = 'España';
```

StDev y StDevP (Desviación Estándar)

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es:

StDev(expr)
StDevP(expr)

En donde expr representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

StDevP evalúa una población, y StDev evalúa una muestra de la población. Si la consulta contiene menos de dos registros (o ningún registro para StDevP), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

SELECT StDev(Gastos) AS Desviacion FROM Pedidos WHERE Pais = 'España';
SELECT StDevP(Gastos) AS Desviacion FROM Pedidos WHERE Pais= 'España';

Sum (Sumar Valores)

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

Sum(expr)

En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla,

Consultas de Eliminación

DELETE crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula FROM que satisfagan la cláusula WHERE. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Su sintaxis es:

DELETE Tabla.* FROM Tabla WHERE criterio

DELETE es especialmente útil cuando se desea eliminar varios registros. En una instrucción DELETE con múltiples tablas, debe incluir el nombre de tabla (Tabla.*). Si especifica más de una tabla desde la que eliminar registros, todas deben ser tablas de muchos a uno. Si desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Se puede utilizar DELETE para eliminar registros de una única tabla o desde varios lados de una relación uno a muchos. Las operaciones de eliminación en cascada en una consulta únicamente eliminan desde varios lados de una relación. Por ejemplo, en la relación entre las tablas Clientes y Pedidos, la tabla Pedidos es la parte de muchos por lo que las operaciones en cascada solo afectaran a la tabla Pedidos. Una consulta de borrado elimina los registros completos, no únicamente los datos en campos específicos. Si desea eliminar valores en un campo especificado, crear una consulta de actualización que cambie los valores a Null.

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

DELETE * FROM Empleados WHERE Cargo = 'Vendedor';

Consultas de Datos Añadidos

una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

SELECT Sum(PrecioUnidad * Cantidad) AS Total FROM DetallePedido;

Var y VarP (Varianza)

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

Var(expr)
VarP(expr)

VarP evalúa una población, y Var evalúa una muestra de la población. Expr el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

Si la consulta contiene menos de dos registros, Var y VarP devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar Var y VarP en una expresión de consulta o en una Instrucción SQL.

SELECT Var(Gastos) AS Varianza FROM Pedidos WHERE Pais = 'España';
SELECT VarP(Gastos) AS Varianza FROM Pedidos WHERE Pais = 'España';

Consultas de Acción

Las consultas de acción son aquellas que no devuelven ningún registro, son las encargadas de acciones como añadir y borrar y modificar registros.

INSERT INTO agrega un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipos: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla.

Insertar un único Registro

En este caso la sintaxis es la siguiente:

```
INSERT INTO Tabla (campo1, campo2, .., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente. Hay que prestar especial atención a acotar entre comillas simples (') los valores literales (cadenas de caracteres) y las fechas indicarlas en formato mm-dd-aa y entre caracteres de almohadillas (#).

Insertar Registros de otra Tabla

En este caso la sintaxis es:

```
INSERT INTO Tabla [IN base_externa] (campo1, campo2, ..., campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2, ..., TablaOrigen.campoN
FROM TablaOrigen
```

En este caso se seleccionarán los campos 1,2, ..., n de la tabla origen y se grabarán en los campos 1,2,.., n de la Tabla. La condición SELECT puede incluir la cláusula WHERE para filtrar los registros a copiar. Si Tabla y TablaOrigen poseen la misma estrucutra podemos simplificar la sintaxis a:

```
INSERT INTO Tabla  SELECT TablaOrigen.* FROM TablaOrigen
```

De esta forma los campos de TablaOrigen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de TablaOrigen estén contenidos

con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de TablaOrigen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementandose como le corresponde.

Se puede utilizar la instrucción INSERT INTO para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código especifica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar INSERT INTO para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula SELECT ... FROM como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula SELECT especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta.

Si la tabla destino contiene una clave principal, hay que segurarse que es única, y con valores no-Null ; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador , no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula IN para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las

tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula VALUES. Si se omite la lista de campos, la cláusula VALUES debe incluir un valor para cada campo de la tabla, de otra forma fallará INSERT.

```
INSERT INTO Clientes SELECT Clientes_Viejos.* FROM Clientes_Nuevos;
INSERT INTO Empleados (Nombre, Apellido, Cargo)
VALUES ('Luis', 'Sánchez', 'Becario');

INSERT INTO Empleados SELECT Vendedores.* FROM Vendedores
WHERE Fecha_Contratacion < Now() - 30;
```

Consultas de Actualización

UPDATE

crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico. Su sintaxis es:

```
UPDATE Tabla SET Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN
WHERE Criterio;
```

UPDATE es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez. El ejemplo siguiente incrementa los valores Cantidad pedidos en un 10 por ciento y los valores Transporte en un 3 por ciento para aquellos que se hayan enviado al Reino Unido:

```
UPDATE Pedidos SET Pedido = Pedidos * 1.1, Transporte = Transporte *
1.03
WHERE PaisEnvio = 'ES';
```

UPDATE

no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

UPDATE Empleados SET Grado = 5 WHERE Grado = 2;

UPDATE Productos SET Precio = Precio * 1.1 WHERE Proveedor = 8 AND Familia = 3;

Si en una consulta de actualización suprimimos la cláusula

WHERE

todos los registros de la tabla señalada serán actualizados.

UPDATE Empleados SET Salario = Salario * 1.1

Tipos de datos

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos.

Tipos de datos primarios:

Tipo de **os**	**Longitud**	**Descripción**
BINARY	1 byte	Para consultas sobre tabla adjunta de productos ases de datos que definen un tipo de datos Binario.
BIT	1 byte	Valores Si/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de Long)
CURRENCY	8 bytes	Un entero escalable entre 337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y .

SINGLE	4 bytes	Un valor en punto flotante de precisión simple con rango de -3.402823*1038a -1.401298*10-45para es negativos, 1.401298*10-45a 3.402823*1038para es positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con rango de -1.79769313486232*10308a -065645841247*10-324para valores negativos, 065645841247*10-324a 769313486232*10308para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 7,483,647.
LONGTEXT	1 byte por cter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se site	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por cter	De cero a 255 caracteres.

La siguiente tabla recoge los sinonimos de los tipos de datos definidos:

Tipo de Dato	**Sinónimos**
BINARY	VARBINARY
BIT	BOOLEAN ICAL ICAL1 NO
BYTE	INTEGER1

COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE E ESTAMP
SINGLE	FLOAT4 ISINGLE L
DOUBLE	FLOAT AT8 IDOUBLE IBER IERIC
SHORT	INTEGER2 LLINT
LONG	INT IGER IGER4
LONGBINARY	GENERAL OBJECT
LONGTEXT	LONGCHAR IO E
TEXT	ALPHANUMERIC R RACTER ING CHAR

VARIANT (No Admitido)	VALUE

Subconsultas

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta.

Puede utilizar tres formas de sintaxis para crear una subconsulta:

comparación [ANY | ALL | SOME] (instrucción sql)
expresión [NOT] IN (instrucción sql)
[NOT] EXISTS (instrucción sql)

En donde:

comparación

Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta

expresión

Es una expresión por la que se busca el conjunto resultante de la subconsulta.

instrucción sql

Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING.

Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE PrecioUnidad > ANY
(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);
```

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE IDProducto IN
(SELECT IDProducto FROM DetallePedido WHERE Descuento >=  0.25);
```

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo

siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título. A la tabla Empleados se le ha dado el alias T1:

```
SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1
WHERE Salario >= (SELECT Avg(Salario) FROM Empleados
WHERE T1.Titulo = Empleados.Titulo) ORDER BY Titulo;
```

En el ejemplo anterior , la palabra reservada AS es opcional. Otros ejemplos:

```
SELECT Apellidos, Nombre, Cargo, Salario FROM Empleados
WHERE Cargo LIKE "Agente Ven*" AND Salario > ALL (SELECT Salario
FROM
Empleados WHERE (Cargo LIKE "*Jefe*") OR (Cargo LIKE "*Director*"));
```

Obtiene una lista con el nombre, cargo y salario de todos los agentes de ventas cuyo salario es mayor que el de todos los jefes y directores.

```
SELECT DISTINCTROW NombreProducto, Precio_Unidad FROM Productos
WHERE (Precio_Unidad = (SELECT Precio_Unidad FROM Productos
WHERE
Nombre_Producto = "Almíbar anisado");
```

Obtiene una lista con el nombre y el precio unitario de todos los productos con el mismo precio que el almíbar anisado.

```
SELECT DISTINCTROW Nombre_Contacto, Nombre_Compañia,
Cargo_Contacto,
Telefono FROM Clientes WHERE (ID_Cliente IN (SELECT DISTINCTROW
ID_Cliente FROM Pedidos WHERE Fecha_Pedido >= #04/1/93#
<#07/1/93#);
```

Obtiene una lista de las compañías y los contactos de todos los clientes que han realizado un pedido en el segundo trimestre de 1993.

```
SELECT Nombre, Apellidos FROM Empleados AS E WHERE EXISTS
(SELECT * FROM Pedidos AS O WHERE O.ID_Empleado =
E.ID_Empleado);
```

Selecciona el nombre de todos los empleados que han reservado al menos un pedido.

```
SELECT DISTINCTROW Pedidos.Id_Producto, Pedidos.Cantidad,
(SELECT DISTINCTROW Productos.Nombre FROM Productos WHERE
Productos.Id_Producto = Pedidos.Id_Producto) AS ElProducto FROM
Pedidos WHERE Pedidos.Cantidad > 150 ORDER BY Pedidos.Id_Producto;
```

Recupera el Código del Producto y la Cantidad pedida de la tabla pedidos, extrayendo el nombre del producto de la tabla de productos.

Consultas de Referencias Cruzadas

Una consulta de referencias cruzadas es aquella que nos permite visualizar los datos en filas y en columnas, estilo tabla, por ejemplo:

Producto / Año	1996	1997
Pantalones	1.250	3.000
Camisas	8.560	1.253
Zapatos	4.369	2.563

Si tenemos una tabla de productos y otra tabla de pedidos, podemos visualizar en total de productos pedidos por año para un artículo determinado, tal y como se visualiza en la tabla anterior.

La sintaxis para este tipo de consulta es la siguiente:

```
TRANSFORM función agregada instrucción select PIVOT campo pivot
[IN (valor1[, valor2[, ...]])]
```

En donde:

función agregada

Es una función SQL agregada que opera sobre los datos seleccionados.

instrucción select

Es una instrucción SELECT.

campo pivot

Es el campo o expresión que desea utilizar para crear las cabeceras de la columna en el resultado de la consulta.

valor1, valor2

Son valores fijos utilizados para crear las cabeceras de la columna.

Para resumir datos utilizando una consulta de referencia cruzada, se seleccionan los valores de los campos o expresiones especificadas como cabeceras de columnas de tal forma que pueden verse los datos en un formato más compacto que con una consulta de selección.

TRANSFORM es opcional pero si se incluye es la primera instrucción de una cadena SQL. Precede a la instrucción SELECT que especifica los campos utilizados como encabezados de fila y una cláusula GROUP BY que especifica el agrupamiento de las filas. Opcionalmente puede incluir otras cláusulas como por ejemplo WHERE, que especifica una selección adicional o un criterio de ordenación.

Los valores devueltos en campo pivot se utilizan como encabezados de columna en el resultado de la consulta. Por ejemplo, al utilizar las cifras de ventas en el mes de la venta como pivot en una consulta de referencia cruzada se crearían 12 columnas. Puede restringir el campo pivot para crear encabezados a partir de los valores fijos (valor1, valor2) listados en la cláusula opcional IN.

También puede incluir valores fijos, para los que no existen datos, para crear columnas adicionales.

Ejemplos

TRANSFORM Sum(Cantidad) AS Ventas SELECT Producto, Cantidad FROM

Pedidos WHERE Fecha Between #01-01-98# And #12-31-98# GROUP BY Producto

ORDER BY Producto PIVOT DatePart("m", Fecha);

Crea una consulta de tabla de referencias cruzadas que muestra lasn ventas de productos por mes para un año específico. Los meses aparecen de izquierda a derecha como columnas y los nombres de los productos aparecen de arriba hacia abajo como filas.

TRANSFORM Sum(Cantidad) AS Ventas SELECT Compania FROM Pedidos

WHERE Fecha Between #01-01-98# And #12-31-98# GROUP BY Compania

ORDER BY Compania PIVOT "Trimestre " & DatePart("q", Fecha) In ('Trimestre1',

'Trimestre2', 'Trimestre 3', 'Trimestre 4');

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por trimestre de cada proveedor en el año indicado. Los trimestres aparecen de izquierda a derecha como columnas y los nombres de los proveedores aparecen de arriba hacia abajo como filas.

Un caso práctico:

Se trata de resolver el siguiente problema: tenemos una tabla de productos con dos campos, el código y el nombre del producto, tenemos otra tabla de pedidos en la que anotamos el código del producto, la fecha del pedido y la cantidad pedida. Deseamos consultar los totales de producto por año, calculando la media anual de ventas.

Estructura y datos de las tablas:

1. Articulos:

ID	Nombre

1	Zapatos
2	Pantalones
3	Blusas

2. Pedidos:

Id	Fecha	Cantidad
1	11/11/1996	250
2	11/11/1996	125
3	11/11/1996	520
1	12/10/1996	50
2	04/05/1996	250
3	05/08/1996	100
1	01/01/1997	40
2	02/08/1997	60
3	05/10/1997	70
1	12/12/1997	8
2	15/12/1997	520
3	17/10/1997	1250

Para resolver la consulta planteamos la siguiente consulta:

TRANSFORM Sum(Pedidos.Cantidad) AS Resultado SELECT Nombre AS Producto,

Pedidos.Id AS Código, Sum(Pedidos.Cantidad) AS TOTAL, Avg(Pedidos.Cantidad)

AS Media FROM Pedidos INNER JOIN Artículos ON Pedidos.Id = Articulos.Id

GROUP BY Pedidos.Id, Artículos.Nombre PIVOT Year(Fecha);

y obtenemos el siguiente resultado:

Producto	Código	TOTAL	Media	1996	1997
Zapatatos	1	348	87	300	48
Pantalones	2	955	238,75	375	580
Blusas	3	1940	485	620	1320

Comentarios a la consulta:

La clásula TRANSFORM indica el valor que deseamos visualizar en las columnas que realmente pertenecen a la consulta, en este caso 1996 y 1997, puesto que las demás columnas son opcionales.

SELECT especifica el nombre de las columnas opcionales que deseamos visualizar, en este caso Producto, Código, Total y Media, indicando el nombre del campo que deseamos mostrar en cada columna o el valor de la misma. Si incluimos una función de cálculo el resultado se hará en base a los datos de la fila actual y no al total de los datos.

FROM especifica el origen de los datos. La primera tabla que debe figurar es aquella de donde deseamos extraer los datos, esta tabla debe contener al menos tres campos, uno para los títulos de la fila, otros para los títulos de la columna y otro para calcular el valor de las celdas.

En este caso en concreto se deseaba visualizar el nombre del producto, como el tabla de pedidos sólo figuraba el código del mismo se añadió una nueva columna en la cláusula select llamada Producto que se corresponda con el campo Nombre de la tabla de artículos. Para vincular el código del artículo de la tabla de pedidos con el nombre del misma de la tabla artículos se insertó la cláusula INNER JOIN.

La cláusula GROUP BY especifica el agrupamiento de los registros, contrariamente a los manuales de instrucción esta cláusula no es opcional ya que debe figurar siempre y debemos agrupar los registros por el campo del cual extraemos la información. En este caso existen dos campos del cual extraemos la información: pedidos.cantidad y artículos.nombre, por ellos agrupamos por los campos.

Para finalizar la cláusula PIVOT indica el nombre de las columnas no opcionales, en este caso 1996 y 1997 y como vamos al dato que aparecerá en las columnas, en este caso empleamos el año en que se produjo el pedido, extrayéndolo del campo pedidos.fecha.

Otras posibilidades de fecha de la cláusula PIVOT son las siguientes:

1. Para agrupamiento por Trimestres

PIVOT "Tri " & DatePart("q",[Fecha]);

2. Para agrupamiento por meses (sin tener en cuenta el año)
3. PIVOT Format([Fecha],"mmm") In ("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic");

4. Para agrupar por días

PIVOT Format([Fecha],"Short Date");

APÉNDICE

El control 'Data' y DAO (objeto para acceso a datos)

Capítulo 10.

Acceso a base de datos

Visual Basic nos permite manipular la información en bases de datos muy fácilmente utilizando controles que se relacionan directamente con las bases de datos.

Los datos en una base de datos relacional se almacenan en tablas, que se construyen con base a renglones y columnas. Cada renglón dentro de una tabla almacena información sobre un elemento relacionada a la tabla. Así, si tenemos la tabla "Empleados" cada renglón almacena información sobre un empleado. A los renglones se les conoce como registros. A su vez los registros se componen de varios campos (las columnas), en las que se almacena una pieza única de información. En nuestra tabla de "Empleados", podríamos tener campos para almacenar el nombre, apellidos, dirección, puesto y sueldo de cada empleado.

Visual Basic se puede comunicar con tres tipos de bases de datos:

- Bases de datos JET. Las bases de datos son creadas y manipuladas por el mecanismo JET. El mismo mecanismo JET es usado por *Microsoft Access* y *Microsoft Visual Basic*.

- Bases de datos ISAM (Indexed Sequential Access Method), por ejemplo, dBase, Paradox y Btrieve. *Visual Basic* se comunica con estas bases de datos a través de *drivers* específicos. Estos *drivers* se pueden instalar dependiendo de la base de datos que se vaya a trabajar.

- Bases de datos ODBC (Open Database Connectivity). Estos incluyen manejadores de bases de datos con arquitectura cliente/servidor como son SQL Server y ORACLE. Para comunicarse con estas bases, *Visual Basic* necesita los *drivers* ODBC adecuados.

Para consultar la información en la base de datos se puede utilizar SQL (Structured Query Language), que es un lenguaje que facilita la comunicación con la base de datos. SQL se ha convertido en el medio más utilizado para conversar con una base de datos y *Visual Basic* trae integrado una implementación de SQL. A través de SQL se pasan las consultas de información que queremos y la base de datos nos regresa todos los registros que concuerden con nuestra consulta.

10.1 *Data Control* 

El *data control* nos permite ligarnos a una base de datos muy fácilmente. Esto hará que el *data control* tenga una referencia a la base de datos que estamos trabajando y cualquier método que utilicemos con el *data control* se reflejará en la base de datos. El *data control* permite desplazarnos por los registros de una base de datos y desplegar un registro a la vez en controles ligados al *data control*.

10.1.1 Propiedades del *data control*

Propiedades	Descripción
DataBaseName	El *data control* localiza la base de datos a través de esta propiedad. Determina el nombre del archivo de la base de datos, o la ruta donde se localizan los archivos de base de datos.
Exclusive	Determina si la base de datos se abrirá de modo exclusiva para la aplicación. Si *Exclusive* es verdadero se le negará el acceso a la base de datos a cualquier otra aplicación. Si otra aplicación ya tiene abierta la base de datos cuando nuestra aplicación la intenta abrir en modo exclusivo, nuestra aplicación recibe un error.
ReadOnly	Determina si se puede o no escribir a la base de datos.
RecordSource	Indica el nombre de la tabla o una cadena con la sentencia SQL con la cuál se selecciona el conjunto de registros que trabajará el *data control*.
Connect	Indica el tipo de base de datos con el que se conectará el *data control*

Dependiendo del valor de la propiedad *Connect*, la propiedad DataBaseName apuntará a un directorio, archivo, o fuente de datos.

Base de Datos	***DatabaseName***	***Connect***
Microsoft Access	drive:\ruta\archivo.mdb	
dBase	drive:\ruta\	"dbase III" o "dbase IV"
Paradox	drive:\ruta\	"Paradox 3.x"
Btrieve	drive:\ruta\	"btrieve
ODBC	Fuente de datos registrada (servidor)	"odbc;dsn=*datasource*: uid=*usuario*,pwd=*contraseña*"

- Crear una aplicación que despliegue el contenido de una base de datos. Para este ejemplo se ligarán controles a la base de datos.

- Agregue un *data control* (dtaAutores) y dos cajas de texto (txtClaveAutor, txtNombre).

- dtaAutores.DataBasename = "**VB**\Biblio.Mdb"
  dtaAutores.RecordSource = "Authors"

Al establecer la propiedad *DataBasename Visual Basic* intentará comunicarse con la base de datos para leer su estructura. Si logra comunicarse con la base de datos, al momento de seleccionar la propiedad *Recordsource* tendremos una lista predeterminada de los nombres de las tablas que podemos acceder en la base de datos.

Los controles que podemos ligar a un *data control* son: cajas de texto (*textbox*), etiquetas (*labels*), cuadros de dibujo (*picture box* e *image* control), y cuadros de selección (*check box*). Para ligar un control a un *data control* se utilizarán las propiedades *DataSource* y *DataField* del control a ligar.

Propiedades	Descripción
DataSource	Determina con que data control se ligará el control. Esta propiedad sólo se puede modificar en tiempo de diseño.
DataField	En esta propiedad se especifica el nombre del campo que se desplegará en el control.

Entonces, para ligar nuestras cajas de texto con el *data control* haremos:

txtClaveAutor.DataSource = "dtaAutores"
txtClaveAutor.DataField = "Au_ID"
txtNombre.DataSource = "dtaAutores"
txtNombre.DataField = "Author"

La propiedad *DataField* tendrá predeterminados una lista de los campos de una tabla si ya se tiene apuntada la propiedad *DataSource* a un data control que a su vez ya esté ligado con una base de datos.

Pruebe la aplicación. Utilice las flechas en el *data control*, esto nos permite desplazarnos por los registros de la base de datos.

10.1.2 Eventos del *data control*

Eventos	Descripción
Error	Se dispara cuando ocurre un error en el acceso a la base de datos y no hay código ejecutándose. Tiene dos argumentos *DataErr* y *Response*. El primero almacena el número de error que ocurrió. El argumento *Response* determina la acción a tomar. El default es *Response* en 1, que desplegará el mensaje de error, con *Response* en 0 el programa continuará sin desplegar el error.
Reposition	Cada vez que el *data control* se posiciona en un nuevo registro, se dispara este evento.
Validate	Se dispara antes de que un registro se convierta en el registro actual de trabajo. Este evento tiene dos argumentos *Action* y *Save*. *Action* contiene un valor numérico que determina la instrucción que se ejecutó para hacer que el *data control* cambie de registro, el valor de *Action* se puede cambiar para que al salir de este evento se ejecute una acción diferente a la que disparo el evento *Validate*. *Save* indicará si al final del evento se deben grabar datos a la base de datos. Si algún control ligado ha sido modificado, *Save* tomará el valor de verdadero que indica que se deben grabar los datos; para evitar la operación se puede poner *Save* en falso. Inicialmente *Save* está en verdadero si la información en algún control ligado ha cambiado, para determinar esto, *Visual Basic* revisa la propiedad *DataChanged* de los controles ligados. Esta propiedad sólo existe en tiempo de ejecución.

10.2 *Recordset*

El *data control* para comunicarse con la base de datos crea un objeto dinámico llamado *recordset*. Basándose en la consulta utilizada (con SQL) el *recordset* contendrá la estructura para hacer referencia a los datos en la base de datos y es a través del *recordset* que podemos accesar y modificar la información ahí almacenada. El *recordset* será entonces una estructura que apunte hacia la información en la base de datos.

10.2.1 Métodos del *recordset*

El objeto *recordset* tiene varios métodos que nos permiten realizar las operaciones básicas en el manejo de base de datos fácilmente:

Métodos	Descripción
AddNew	Agrega un registro nuevo. Entonces, para agregar un registro a un *data control* llamado "dtaAutores", utilizaríamos la sentencia: dtaAutores.Recordset.AddNew
Delete	Borra el registro actual. Cuando borramos un registro, el *recordset* queda apuntando a una posición inválida (apunta a un registro que ya no existe), por lo que debemos mover nuestro apuntador a un registro válido antes de intentar otra operación.
MoveFirst, MoveLast, MoveNext, MovePrevious	Para movernos al principio, final, siguiente o anterior del *recordset*.
FindFirst, FindLast, FindNext, FindPrevious	Estos métodos realizan una búsqueda en *recordsets* de tipo *Dynaset* o *Snapshot*. Es necesario indicarle el criterio de búsqueda que se construye como la parte de la sentencia "WHERE" en SQL pero sin la palabra "WHERE". Por ejemplo para buscar un autor del cuál tenemos su clave, utilizaríamos la sentencia: dtaAutores.Recordset.FindFirst "Au_ID = 5" recuerde que en la sentencia WHERE para pasar cadenas se deben encerrar entre apóstrofes: dtaAutores.Recordset.FindFirst "[Authors]='Flavin, Matt.'".
Update	Graba los registros nuevos o cambios a la base de datos.
Edit	Selecciona el registro actual para ser editado.
UpdateControls	Obtiene la información del registro actual de la base de datos y la despliega en los registros ligados. Tiene el mismo efecto que redefinir el registro actual como el registro actual pero sin disparar eventos.
CancelUpdate	Cancela la operación de edición que se esté llevando a cabo, ya sea edición o alta de un registro.

10.2.2 Propiedades del *recordset*

Propiedades	Descripción
NoMatch	Si la propiedad está en verdadero indica que el último método de búsqueda (*FindFirst, FindLast*, etc.) no encontró un registro que igualara el criterio especificado.
Bookmark	Un *bookmark* es un separador de libros. Utilizamos los separadores de libros para encontrar rápidamente una hoja. En los *recordsets* está propiedad se puede almacenar en una variable (tipo cadena) para almacenar posiciones de registros. Después podemos asignar el valor de una variable que contenga una posición válida para regresar rápidamente a ese registro. 'Almacena la posición en una variable posición= dtaAutores.Recordset.Bookmark La siguiente línea establecería el registro actual del *recordset* a la posición almacenada en la variable "posición". dtaAutores.Recordset.Bookmark = posición
BOF	Indica que el apuntador del registro actual ha rebasado el inicio del *recordset* cuando está propiedad toma el valor de verdadero.
EOF	Cuando se encuentra en verdadero, indica que se ha rebasado el fin del *recordset*.
Recordcount	Determina el número de registros que se han accesado. Para que la propiedad contenga el número total de registros. es necesario accesar todos; esto se puede hacer con el método *MoveLast*.
EditMode	Determina el estado de edición en que se encuentra el *recordset*. Este puede ser: 0 - No hay edición en progreso. 1 - Editando un registro. 2 - Añadiendo un nuevo registro.
A partir de *Visual Basic* 4.0 se incluyen las siguientes propiedades:	
PercentPosition	Indica la posición actual como porcentaje del recordset. Para que este porcentaje sea del total de registros es necesario haber accesado hasta el último registro del recordset. Esta propiedad también me permite moverme a un registro. 'Moverme a la mitad de mi *recordset*. Data1.Recordset.PercentPosition = 50
AbsolutePosition	Indica o especifica el número de registro en el que estoy posicionado. El primer registro tiene la posición absoluta 0.

10.3 Colección de campos

El *recordset* crea dinámicamente una colección de los campos que lo forman. De esta forma se puede accesar desde código los campos del *recordset*. Hay varias sintaxis para lograrlo:

```
clave = dtaAutores.Recordset Fields("Au_ID").Value
clave = dtaAutores.Recordset.Fields!Au_ID.Value
clave = dtaAutores.Recordset.Fields[Au_ID].Value
```

Estos comandos leerían el contenido del campo Au_ID y lo almacenarían en la variable "clave"; es importante hacer notar que la variable debe de ser un tipo adecuado para almacenar la información del campo que leemos. Las siguientes sentencias almacenarían el contenido de una variable en los campos especificados del *recordset*, pero faltaría un *Update* para que se guardaran en la base de datos.

```
dtaAutores.Recordset.Fields!Authors = nombre$
dtaAutores.Recordset.Fields[Authors] = "Asimov, Isaac."
dtaAutores.Recordset("Au_ID") = clave
```

Todos estos comandos darían el mismo resultado. La sintaxis de paréntesis con comilla y la de corchete la podemos utilizar para especificar nombres de campos que contengan espacios; por ejemplo:

```
clave = dtaAutores.Recordset.Fields("Clave del Autor").Value
clave = dtaAutores.Recordset.Fields[Clave del Autor].Value
```

También se pueden accesar a los campos indicando con número la posición que ocupan. El primer campo tiene la posición cero, el segundo la posición uno, etc. Este orden se determina por el diseño de la tabla o el orden en que yo pida los campos en la consulta SQL. La siguiente instrucción obtiene el contenido del primer campo dentro del *recordset*:

```
clave = dtaAutores.Recordset.Fields(0).Value
```

La propiedad *Value* es la propiedad default del objeto campo (*Field object*), por lo que podemos omitirla. Entonces las sentencias de acceso a la base de datos con los objetos campos es frecuente encontrarlos de la siguiente manera:

```
clave = dtaAutores.Recordset.Fields!Au_ID
clave = dtaAutores.Recordset Fields[Clave del Autor]
clave = dtaAutores.Recordset.Fields(1)
```

La colección por default del objeto *recordset* es la colección de campos, por lo que todavía podemos reducir las sentencias. Por ejemplo:

```
clave = dtaAutores.Recordset!Au_ID
clave = dtaAutores.Recordset("Au_ID")
```

10.4 Laboratorio

- Agregar la funcionalidad para trabajar con base de datos a la interfaz diseñada en el capítulo I.

Agregue a la interfaz que se diseño en el capítulo I, la funcionalidad para trabajar con base de datos. Agregué un *data control* (dtaTítulos) y ligue este control con la base de datos y a su vez ligue las cajas de texto con el *data control*.

Agregue a los botones de Agregar, Borrar, Cancelar, Grabar las sentencias necesarias para realizar su función. Agregue un botón de comandos (cmdBuscar) para poder realizar búsquedas de registros.

Capítulo 11.
Acceso a base de datos usando variables objeto

Visual Basic permite comunicarse a la base de datos sin necesidad de utilizar objetos como el *data control* ya que tiene tipos de datos que se utilizan para representar objetos de bases de datos. Es a través de estas variables que podemos accesar la información en la base de datos.

11.1 Variable tipo base de datos

La variable que se utiliza para representar una base de datos, se declara de tipo *database*. Utilizando está variable y aplicando una función para abrir una base de datos. obtenemos una referencia válida a la base de datos.

- **Declaración de un objeto tipo base de datos:**

```
Dim db as database
```

- **La función *OpenDatabase***

En la función le indicamos el nombre de la base de datos que deseamos abrir, el tipo de base de datos con la cuál se está conectando y si será de lectura o exclusiva la base de datos. Los valores para los argumentos de "archivo$" y "connect$" son los mismos que los mostrados en la tabla para el *data control*.

```
Set db = OpenDatabase( archivo$ , Exclusiva ,Modo Lectura , connect$ )
Set db = OpenDatabase("f:\usr\maq39\Biblio.Mdb",true,false)
Set db = OpenDatabase("f:\usr\maq39\bd\",false,false,"dbase III")
```

- **Para cerrar la base de datos:**

```
db Close
```

Ya que tenemos una referencia válida a la base de datos, se utilizan otras funciones para crear los objetos con los cuales modificaremos la información. Como prefijo a estas funciones le indicamos la variable u objeto hacia la base de datos que queremos trabajar. En los siguientes ejemplos la variable "db" es una referencia válida hacia una base de datos.

11.2 Variable tipo tabla

Las variables tipo tabla harán referencia a las tablas dentro de una base de datos. A través de estas variables podemos modificar los campos de una tabla en particular. No podemos abrir objetos tipos tabla sobre tablas vinculadas a nuestra base de datos o con conexiones ODBC. Este tipo de objeto es el que más "consciente" está de los datos en una base de datos.

- **Declaración de un objeto tipo tabla**

```
Dim tb as Table
```

- **La función *OpenTable*:**

En la función le indicamos el nombre de la tabla que queremos abrir, y también podemos especificarle algunas opciones adicionales al momento de abrir la tabla.

```
Set tb = db.OpenTable(tabla$, opciones% )
Set tb = db.OpenTable("Empleados")
```

El campo de opciones se utiliza para especificar el modo en que se abrirá la tabla:

Opciones	Descripción
1	Negar escritura a los demás
2	Negar lectura a los demás
4	Abrir en modo de sólo lectura

Para combinar las opciones, se suman los valores; por ejemplo:

```
Set tb = db OpenTable("Empleados", 3)
```

Abrirá la tabla denegando permisos de lectura y escritura a los demás.

- **Ordenar un objeto tabla por sus índices:**

Se utiliza la propiedad *'Index'* del objeto tabla:

```
Set tb = db.OpenTable("Empleados")
tb.Index = "Apellido"
```

Para utilizar tablas de dBase con índices, se debe crear un archivo .INF que liste los índices. Por ejemplo, si se tiene un archivo de dBase llamado EMPL.DBF con varios índices en los archivos EMPL.NDX, RFC.NDX, NOMBRE.NDX se debe crear un archivo de texto llamado EMPL.INF que contenga:

```
[dbase]
NDX1=empl.ndx
NDX2=rfc.ndx
NDX3=nombre.ndx
```

para que *Visual Basic* sepa que archivos son los que tiene que actualizar al momento de dar de alto o de baja un registro.

11.3 Variable tipo *Dynaset*

El objeto Dynaset me permite crear el recordset a partir de una tabla o de una consulta en SQL. Este objeto se crea a partir de una referencia a una base de datos válida.

- **Declaración de un objeto tipo *Dynaset***

```
Dim ds as Dynaset
```

- **La función *CreateDynaset***

A esta función le indicamos el nombre de la tabla que va a abrir, o bien la sentencia SQL con la cuál deseamos crear el *recordset*; y en otro argumento le podemos indicar otras opciones que alteran el modo en que creamos nuestro Dynaset.

```
Set ds = db.CreateDynaset( tabla$ | sentencia_SQL$ , opciones )
```

```
Set ds = db.CreateDynaset ( "Empleados" )
Set ds = db.CreateDynaset ( "Select * from Authors, Titles where Author.AU_ID = Titles.AU_ID" )
```

Las opciones nos permiten modificar el modo con el que se crea el *Dynaset*, por ejemplo, con:

```
Set ds = db.CreateDynaset ( "Select * from Authors, Titles where Author.AU_ID = Titles.AU_ID" , 4 )
```

Indicamos que cree el *Dynaset* a partir de esa sentencia SQL en modo sólo lectura, lo cual es más rápido si sólo vamos a realizar consultas y no vamos a realizar algún cambio.

Si la sentencia_SQL es demasiado complicada para la implementación de SQL que tiene *Visual Basic*, se puede indicar a la función *CreateDynaset* que pase la sentencia_SQL a la base de datos para que sea ésta la que ejecute la sentencia.

```
Set ds = db.CreateDynaset ( sentencia_SQL_muy_complicada, 64 )
```

- **Ordenar un objeto *Dynaset***

Para ordenar un *Dynaset* se puede especificar en la propiedad *Sort* del *Dynaset*

```
ds.Sort = "Apellido Asc"
```

Esto ordenaría el *Dynaset* por el campo apellido en orden ascendente; sin embargo, para que refleje los cambios es necesario reconstruir el *Dynaset*.

```
Dim ds as Dynaset, dsOrdenado as Dynaset
ds.Sort = "Apellido Asc"
Set ds = db.CreateDynaset( "Select * from Authors")
Set dsOrdenado = ds.CreateDynaset( )
```

Si se sabe el orden que queremos antes de crear el *Dynaset*, es más eficiente ordenarlo desde la sentencia_SQL

```
Set ds = db.CreateDynaset ( "Select * from Autores order by Apellido" )
```

11.4 Variable tipo *SnapShot*

El objeto *SnapShot* es una vista de la información en la base de datos en ese momento, es como si tomamos una fotografía de la información en ese momento: es una copia estática de la información. Al crear un objeto *SnapShot* toda la información del recordset queda almacenada en memoria por lo que en consultas que regresen mucha información esto puede ocupar muchos recursos de una máquina. Por ser sólo son una vista de la información, no podemos realizar ningún cambio en nuestro *recordset*, y no veremos ningún cambio que se haga en la información en la base de datos en nuestro *snapshot* después de que lo hayamos creado.

- **Declaración de un objeto tipo *Snapshot***

```
Dim sn as Snapshot
```

- **La función *CreateSnapshot***

A la función *CreateSnapshot* le podemos indicar la tabla, la sentencia SQL o un recordset ya existente del cuál queremos crearlo. Y podemos especificar como prefijo de esta función no sólo un objeto de base de datos, también le podemos indicar que cree el snapshot a partir de un recordset que ya exista. Esta función también contempla un argumento de opciones que alteran el modo en que se crea el snapshot.

```
Set sn = db.CreateSnapshot(tabla$ | dynaset | sentencia_SQL$, opciones%)
Set sn = recordset.CreateSnapshot( [ options ] )
```

Para crear un *SnapShot* de una Tabla en la Base de Datos:

```
Set sn = db.CreateSnapshot("Empleados")
```

Para crear un *SnapShot* de un *Dynaset* ya existente:

```
Set sn = ds.CreateSnapshot()
```

11.5 Variable tipo *Recordset*

A partir de *Visual Basic* 4.0, los objetos para manejar las bases de datos mejoran en sus características y tienen más funcionalidad, pero cambian las funciones aunque los conceptos se conserven. Las funciones para trabajar con *Dynasets, Tablas y SnapShots* se mantienen por compatiblidad en las librerías disponibles para *Visual Basic*, pero en versiones futuras se descartaran.

En la versión 4.0 se agrega el tipo *Recordset* a partir del cuál se definen las variables *Table, Dynaset* y *Snapshot*.

- **Declaración de un objeto tipo *Recordset***

```
Dim rs as Recordset
```

- **La función *OpenRecordset***

En esta función se define la fuente de los datos, como puede ser el nombre de una tabla o una consulta SQL; se define el tipo del recordset, si será de tipo tabla, *dynaset* o *snapshot*; y opciones que altera el modo de abrir el recordset.

```
Set rs = db.OpenRecordset( fuente$, tipo%, opciones%)
```

El argumento **fuente$** es una cadena especificando la sentencia o fuente para construir los registros del *recordset*. Puede ser un nombre de tabla o una sentencia SQL o una consulta predefinida. Para *recordset* tipo tabla sólo puede ser un nombre de tabla.

El parámetro ***tipo*%** determina la clase de recordset que se creará:

Tipo	**Descripción**
dbOpenTable	Recordset tipo tabla.
dbOpenDynaset	Recordset tipo *dynaset*.
dbOpenSnapShot	Recordset tipo *snapshot*.

Si no se especifica el tipo, la función crea un recordset tipo tabla si es posible.

El parámetro de opciones nos especifican las características del nuevo recordset.

Opción	**Descripción**
dbDenyWriteOther	Los usuarios no pueden modificar o añadir registros.
dbDenyReadOther	Los usuarios no pueden ver los registros; sólo para tipo tabla.
dbReadOnly	Sólo se pueden ver registros, otros usuarios pueden modificarlos.
dbAppendOnly	Sólo se pueden agregar nuevos registros; sólo para tipo *dynaset*.
dbInconsistent	Actualizaciones inconsistentes son permitidas; sólo para tipo *dynaset*.
dbConsistentOnly	Actualizaciones consistentes son las permitidas; sólo para tipo *dynaset*. Mutuamente excluyente con la opcion dbInconsistent.
dbForwardOnly	El *recordset* será de tipo *snapshot* y sólo se puede recorrer hacia adelante. Sólo acepta el método *movenext* para movernos por los registros.
dbSQLPassThrough	La sentencia para crear el recordset es pasada directamente al servidor ODBC para ser procesada.
dbSeeChangesGenerate	Se generará un error si otro usuario modifica datos que estemos editando.

Para crear un *Recordset* tipo Tabla:

```
Dim rs as Recordset
Set rs = db.OpenRecordset("Empleados",dbOpenTable)
Set rs = db.OpenRecordset("Empleados")
```

Para crear un *Recordset* tipo *Dynaset*:

```
Set rs = db.OpenRecordset("Select * from Author", dbOpenDynaset)
```

Comparación de objetos para acceso a base de datos

Objeto	Permite la modificación de registros	Refleja altas y bajas de otros usuarios	Se puede obtener como resultado de una sentencia SQL
Table	Se puede añadir, borrar y modificar	Sí	No
Dynaset	Se puede añadir, borrar y modificar	No	Sí
SnapShot	No	-	Sí

El *data control* crea por default un *recordset* de tipo *dynaset*; por lo tanto, los métodos y propiedades mencionadas para el *recordset* del *data control* también se aplica para estos objetos - con sus diferencias -. Por ejemplo, los métodos para añadir o modificar registros no son aplicables con el *SnapShot* por su propia naturaleza (no modificable) y los métodos *Find* no se pueden utilizar en objetos tabla.

Y las variables objeto no se pueden vincular automáticamente a controles, esto se tendría que realizar desde código.

Entonces podemos tener sentencias dentro de nuestra aplicación como en los siguientes ejemplos:

```
tb.Movelast
tb.EOF
apellido = tb![Apellido]
clave = ds.Fields(3) Value
ds Update
tb.Addnew
sn MoveNext
```

11.6 Otras operaciones

- **Filtrando registros en un *Dynaset* o *SnapShot***

Se utiliza la propiedad *Filter* de estos objetos. La propiedad de *Filter* se aplicará a los *dynasets* que se creen subsecuentemente.

```
Set ds = db.CreateDynast( "Empleados" )
ds.Filter = "Apellido = 'Fernández'"
Set dsFiltrado = ds.CreateDynaset( "Empleados" )
```

- **Accesando base de datos de *Access* con un esquema de permisos:**

Si la aplicación debe accesar una base de datos de *Microsoft Access* que tenga un esquema de permisos, se debe incluir en el archivo .INI de nuestra aplicación una línea que indique donde encontrará el archivo de seguridad .MDA.

```
[Options]
SystemDB=c:\access\system.mda
```

Además, antes de iniciar cualquier interacción con la base de datos se debe incluir en código la sentencia:

```
SetDefaultWorkspace nombre_de_usuario$, contraseña$
```

En las nuevas versiones del mecanismo JET, la extensión MDW sustituye a MDA, y se utiliza la propiedad SystemDB del objeto DBEngine para indicar la ruta hacia este archivo. Después se utiliza el método *CreateWorkspace* para indicar el usuario y la contraseña que usaremos para realizar la conexión.

- **Accesando una base de datos de *SQL Server***

La base de datos ya debe estar dada de alta en el Administrador de ODBC.

```
Dim dbconnect as string
' Las fuentes de datos registrada se listan en el archivo ODBC.INI
' o se encuentran en el registro de Windows.
' Se utiliza las herramientas de Administración de ODBC para modificar el
' ODBC.INI o la parte de base de datos en el registro de Windows.

dbconnect="ODBC;UID="juan";PWD="qwerty97";Database=pubs;"
Set db=OpenDatabase("trabajo", False, False, dbconnect)

dbconnect="ODBC;UID="juan";PWD="qwerty97";Database=pubs;DSN=trabajo;"
Set db=OpenDatabase( "", False , False , dbconnect)
```

11.7 Ejemplo A. Crear Títulos.Dat.

Crear el archivo Títulos.Dat que se utiliza en el capítulo 3. Este programa abrirá la tabla Títulos de la base de datos *Biblio.Mdb* y grabará los primeros 50 registros a un archivo de tipo aleatorio.

Declararemos el tipo de dato para que almacene la estructura de los registros de la tabla títulos. El campo Memo lo definimos de 500 caracteres. Esto lo ponemos en la parte de *general_declarations* de un módulo.

```
Type Registro_Libro
   título As String * 255
   año As Integer
   isbn As String * 20
   pubid As Long
   descripción As String * 50
   notas As String * 50
   tema As String * 50
   comentarios As String * 500
End Type
```

Agregue un botón de comandos (cmdCrear).

```
Sub cmdCrear_Click ( )
'Definir nuestros objetos de base de datos
Dim db As Database, rs As Recordset
'Definir un contador de numero de registros, el manejador de archivos,
'y una variable del tipo Registro_Libro
Dim numRegistro%, arch_Libros%, libro As Registro_Libro

'Abrir la base de datos
Set db = OpenDatabase("VB\Biblio.Mdb")
'Crear el recordset con la tabla de Titles
Set rs = db.OpenRecordset("Titles")
'Obtener el manejador de archivo disponible
arch_Libros = FreeFile
'Abrir el archivo aleatorio de salida
Open "Títulos.Dat" For Random As arch_Libros Len = Len(libro)
'Un ciclo para grabar los primeros 50 registros
For numRegistro = 1 To 50
      'Asignar lo que está en el recordset a la variable libro
      libro.título = rs("Title")
      libro.año = rs("Year Published")
      libro.isbn = rs("ISBN")
      libro.pubid = rs("PubID")
      libro.descripción = rs("Description") & ""
      libro.notas = rs("Notes") & ""
      libro.tema = rs("Subject") & ""
      libro.comentarios = rs("Comments") & ""
      'Grabar el registro del libro al archivo aleatorio
      Put arch_Libros, numRegistro, libro
      'Movernos al siguiente registro en la tabla
      rs.MoveNext
Next numRegistro
End Sub
```

11.8 Ejemplo B. Mejorar la interfaz para la tabla Títulos.

Mejoraremos la interfaz de Títulos para hacer que al momento de dar de alta un libro, el usuario seleccione la editorial a través de su nombre.

Hasta el momento la interfaz de la tabla Títulos debe poder navegar por los registros y va mostrando la información en las cajas de texto. Pero en el campo editorial se despliega el número de identificador de las editoriales (PubID). Este número no le da prácticamente nada de información a nuestro usuario ya que él trabaja con nombres de editoriales y no con números de identificación.

Esto se puede hacer relacionando dos controles *dbList* con dos *data controls*, pero utilizaremos este ejercicio para mostrar el uso de algunos eventos y las variables objeto.

Asegúrese que tiene referencia a las librerías DAO 3.0 o superior. Si cuenta con librerías de objetos anteriores, modifique las funciones, variables y objetos que trabajan con base de datos.

Inserte a su interfaz de la tabla Títulos un control combo (cmbEditorial). Utilizaremos este combo para desplegar todos los nombres de las Editoriales.

Definiremos las variables de base de datos en el *general_declarations* de la forma.

```
Dim db as database
Dim rs as recordset
```

Apuntaremos la variable de base de datos. Para que este disponible desde el principio la apuntaremos al cargar la forma. Es importante que este sentencia preceda a cualquiera que utilice el control dtaTítulos para evitarnos errores posteriores.

```
Sub Form_Load
  Set db = OpenDataBase("VB\Biblio.Mdb")
  Call CargarCombo
  dtaTitulos.Refresh
End Sub
```

Cree una subrutina llamada CargarCombo en la cuál llenaremos el control cmbEditorial con los nombres de Editoriales registrados en la base de datos.

```
Sub CargarCombo ( )
  Dim consulta$
  consulta$="Select [Name] from Publishers"
  Set rs = db.OpenRecordset(consulta$,dbOpenDynaset,dbReadOnly)
  Do Until rs.EOF
        cmbEditorial.Additem rs("Name")
        rs.MoveNext
  Loop
End Sub
```

- Pruebe su ejemplo, al cargar su interfaz el combo ya debe contener todos los nombres de las editoriales. Modifique la propiedad *Sort* del control cmbEditorial para que los nombres de las editoriales aparezcan en orden alfabético.

Lo siguiente ha realizar será que al desplazarnos por los registros, el combo despliegue el nombre de la editorial. Para esto tendremos que hacer una consulta tomando el PubID que se despliega en la caja txtEditorial y obteniendo el nombre asociado.

El evento que nos sirve en este caso, será el evento *Reposition* de dtaTítulos. Cada vez que nos coloquemos en un registro, haremos la consulta para obtener el nombre de la Editorial.

```
Sub dtaTítulos_Reposition ( )
    Dim consulta$
    consulta$ = "Select [Name] from Publishers where PubID=" & Val(txtEditorial)
    Set rs = db.CreateDynaset(consulta$, dbOpenDynaset,dbReadOnly)
    cmbEditorial.Text = rs("Name")
End Sub
```

Para asegurar la integridad en las tablas, sabemos que el usuario no debe poder escoger alguna editorial que no exista en la base de datos; lo más práctico es hacer que el usuario sólo pueda seleccionar una editorial de las que aparecen en el control cmbEditorial. Al seleccionar el usuario una editorial, nosotros haremos una consulta y obtendremos el identificador de editorial (PubID) y lo insertaremos en txtEditorial. Tomaremos el nombre que haya seleccionado el usuario del control cmbEditorial y con él haremos la consulta; recuerde que al ser cadena debemos pasar el argumento entre apóstrofes.

```
Private Sub cmbEditorial_Click()
    Dim consulta$
    consulta = "Select PubID from Publishers where Name='" + cmbEditorial.Text + "'"
    Set rs = db.OpenRecordset(consulta$, dbOpenDynaset, dbReadOnly)
    txtEditorial.Text = rs("PubId")
End Sub
```

Pero podemos tener todavía algunos problemas, si intentamos dar de alta un nuevo registro se generará un error de "*No Current Record*". Esto porque al agregar un registro, se dispara también el evento *Reposition* y se ejecuta la consulta ahí especificada. Como la caja txtEditorial está en blanco, la función Val(txtEditorial) me regresa un valor de cero y la consulta busca por una editorial cuyo PubID sea cero. El *recordset* que me regresa la base de datos está vacío y al momento de tratar de leer el contenido del *recordset* con la instrucción:

```
cmbEditorial.Text = rs("Name")
```

marca error porque el *recordset* "rs" no contiene registros.