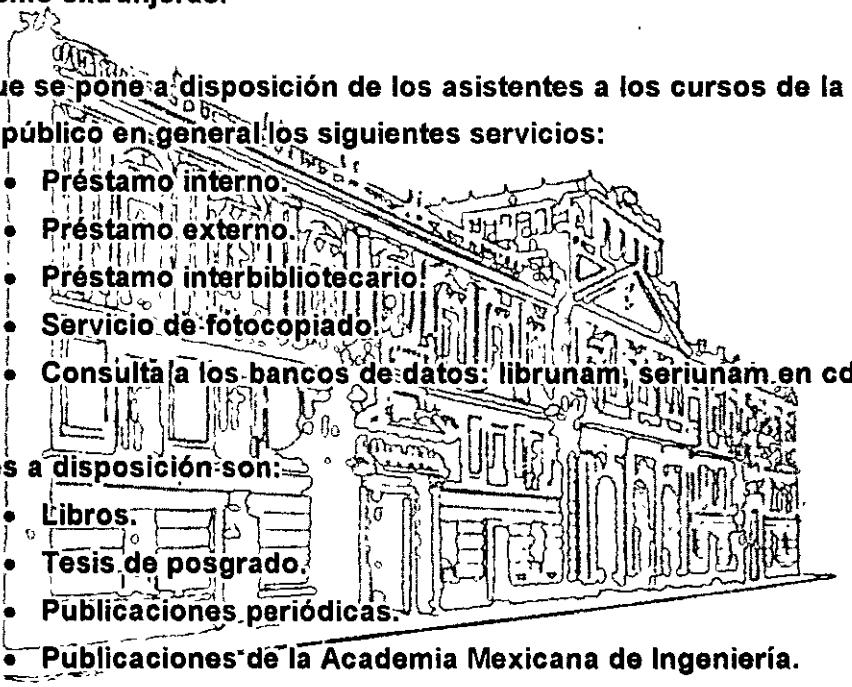
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN
" 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.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1988 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 14:30 y 16:00 a 17:30 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.

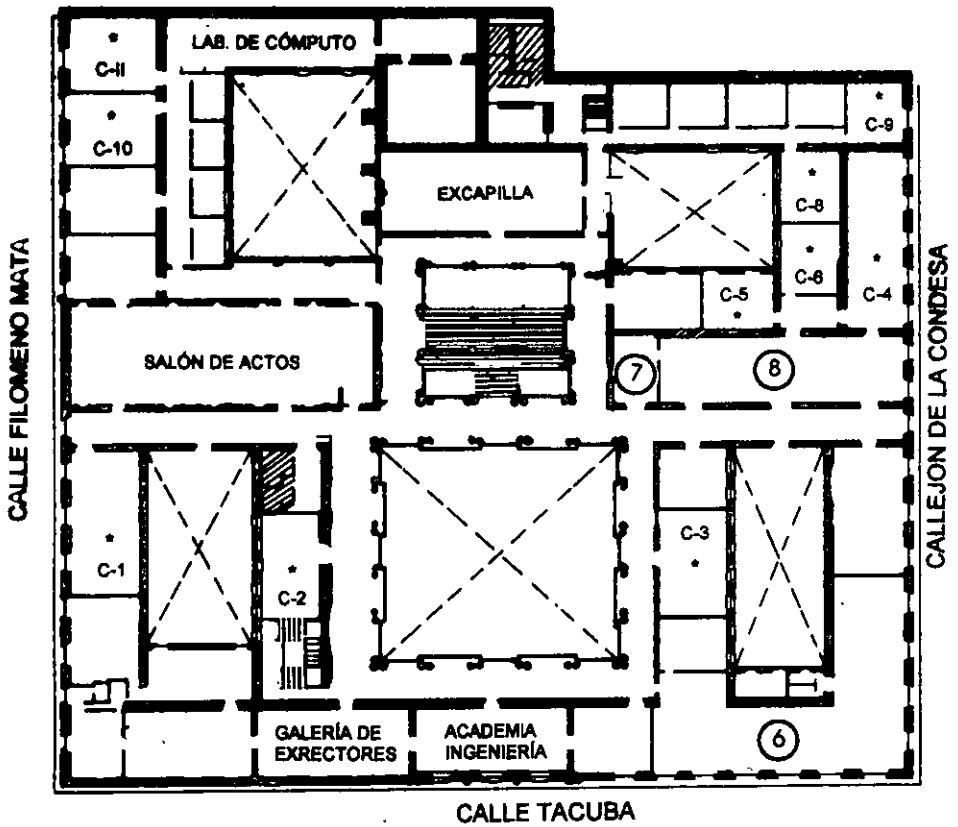
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

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: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

PALACIO DE MINERIA



Ier. 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

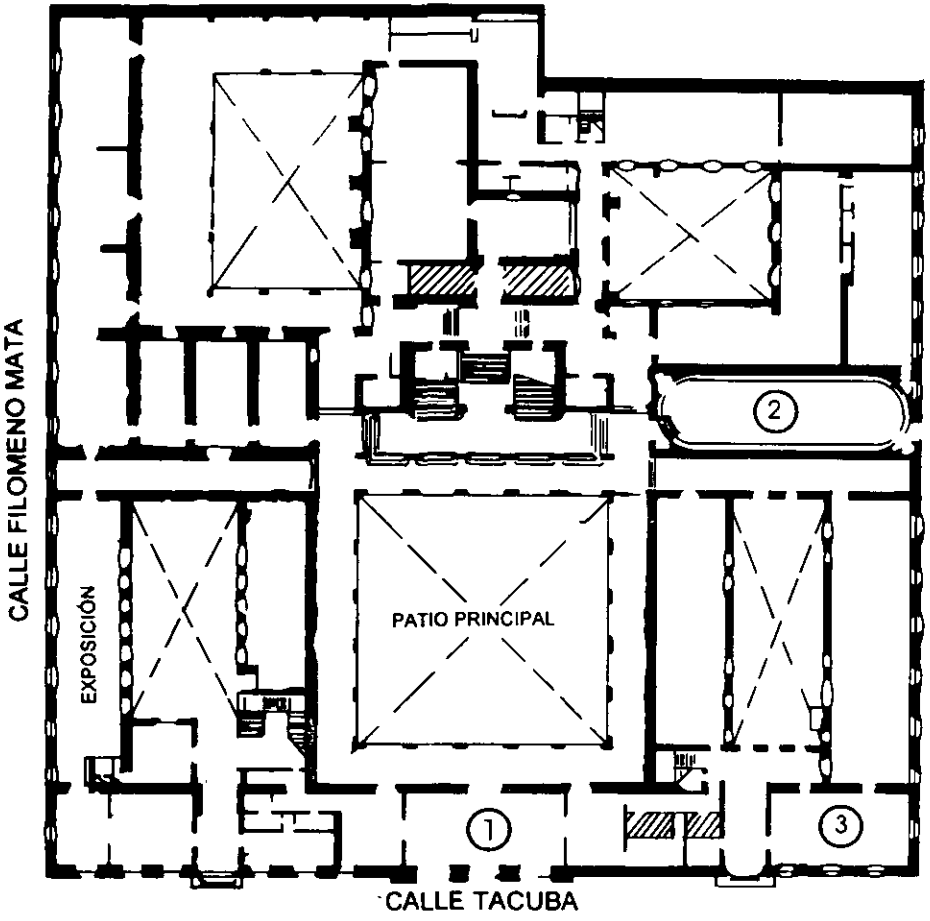
SANITARIOS

* AULAS

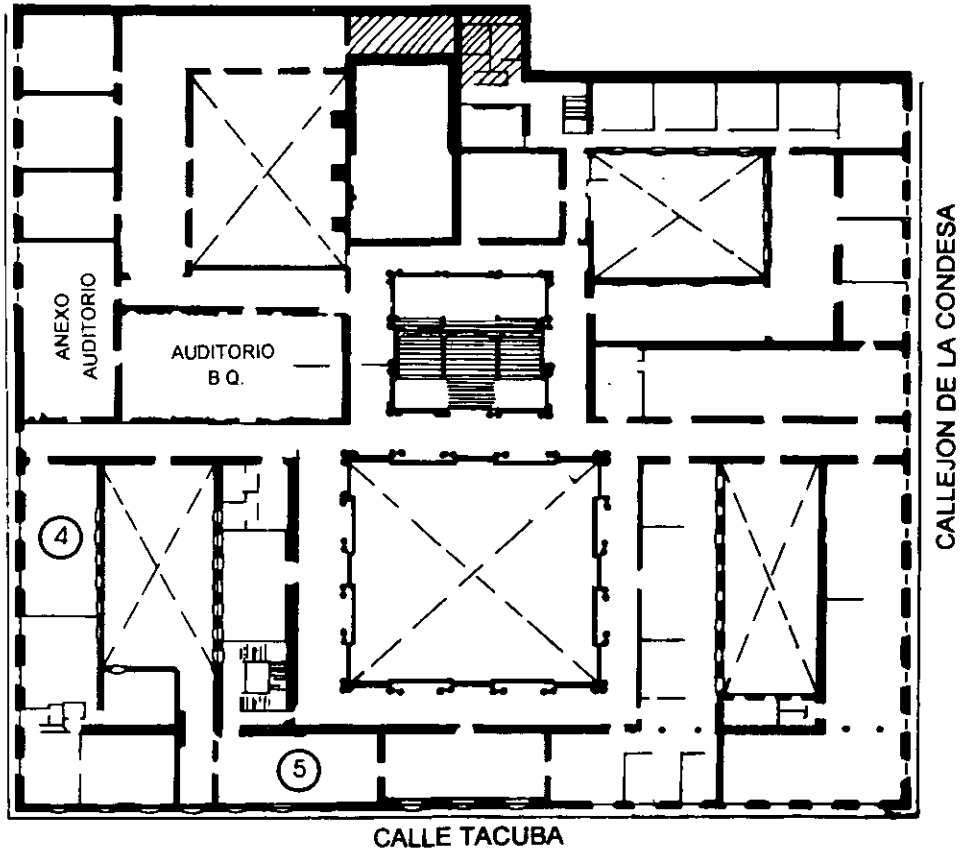
DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA

PALACIO DE MINERIA



PLANTA BAJA



MEZZLANINNE

Microsoft Access 97

Curso avanzado

Manual del usuario

Contenido

CAPÍTULO 1: INTRODUCCIÓN A VISUAL BASIC PARA APLICACIONES EN MICROSOFT ACCESS. 4

1.1 El ambiente de Visual Basic para Aplicaciones. 5
1.2 Variables. 9
1.3 Estructuras de control. 12
1.4 Cualidades de procedimientos Sub. 16
1.5 Cualidades de procedimientos Function. 17
1.6 Alcance y visibilidad. 18
1.7 Preguntas de Repaso. 19

CAPÍTULO 2: PROGRAMACIÓN ORIENTADA A OBJETOS Y ACCESO A DATOS EXTERNOS. 22

2.1 Definiendo objetos y colecciones. 23
2.2 Propiedades y métodos. 24
2.3 Modelo de objetos de Microsoft Access. 25
2.4 Modelo de objetos de acceso a datos (DAO). 28
2.5 DAO: objetos y colecciones de alto nivel. 30
2.6 Tablas, campos e índices. 36
2.7 Colección Properties y Objeto Property. 43
2.8 Consultas, contenedores y relaciones. 44
2.9 Preguntas de repaso 49

CAPÍTULO 3: DEPURACIÓN Y MANEJO DE ERRORES. 51

3.1 Tipos de errores. 52
3.2 La ventana de depuración. 53
3.3 Herramientas y técnicas de depuración. 56
3.4 Manejo de errores. 59
3.5 Repaso. 66

CAPÍTULO 4: TRABAJANDO CON CONJUNTO DE REGISTROS. 68

4.1 Definiendo conjuntos de registros. 69
4.2 Creación de Recordsets. 72
4.3 Manipulación de Recordsets. 76
4.4 Actualización de Recordsets. 81
4.5 Procesamiento de transacciones. 85
4.6 Preguntas de repaso 87

CAPÍTULO 5: MÓDULOS DE CLASE. 89

5.1 ¿Qué es una clase?. 90
5.2 Módulos de clase en Microsoft Access. 91

5.3	Herramientas para trabajar con objetos.	96
5.4	Preguntas de repaso	98
CAPÍTULO 6:	**INTEGRANDO APLICACIONES DE MICROSOFT OFFICE.**	**99**
6.1	Aplicaciones centradas en documentos.	100
6.2	Referenciando un objeto usando una librería de objetos.	101
6.3	Accesando objetos ligados e incrustados.	102
6.4	Automatización OLE.	106
6.5	Microsoft Access como servidor de automatización OLE.	110
6.6	Preguntas de repaso.	111
CAPÍTULO 7:	**CONTROLES OLE.**	**113**
7.1	Introducción a los controles OLE.	114
7.2	Tipos de controles OLE.	115
7.3	Registrando y agregando controles OLE.	117
7.4	Trabajando con controles OLE.	118
7.5	Preguntas de repaso.	121
CAPÍTULO 8:	**LIBRERÍAS DE VÍNCULOS DINÁMICOS (DLL).**	**122**
8.1	Introducción a los DLL's.	123
8.2	Uso de DLL's con VBA.	125
8.3	Información general acerca del uso de DLL's.	128
8.4	Ejemplo de llamadas a DLL.	132
8.5	Preguntas de repaso.	133
CAPÍTULO 9:	**REPLICACIÓN.**	**135**
9.1	Introducción a la replicación de bases de datos.	136
9.2	Replicación a través de la interface del usuario.	139
9.3	Conflictos y errores de la replicación.	140
9.4	Replicación usando DAO.	143
9.5	Preguntas de repaso.	146
CAPÍTULO 10:	**SEGURIDAD.**	**148**
10.1	¿Por qué asegurar bases de datos?.	149
10.2	El modelo de seguridad de Microsoft Jet.	150
10.3	Asegurando bases de datos.	153
10.4	Eliminando seguridad de una base de datos.	155

Capítulo 1: Introducción a Visual Basic para aplicaciones en Microsoft Access.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Interpretar y editar código de Visual Basic para Aplicaciones.
- ☑ Describir las diferencias entre procedimientos Sub y Function.
- ☑ Llamar procedimientos desde otros procedimientos y módulos.
- ☑ Almacenar datos en variables.
- ☑ Crear tipos de datos personales.
- ☑ Usar el asistente de Microsoft Office para obtener ayuda de Visual Basic.

1.1 El ambiente de Visual Basic para Aplicaciones.

Esta sección presenta el lenguaje ***Visual Basic para Aplicaciones*** como el ambiente de desarrollo para diversas aplicaciones de ***Microsoft Office***, incluido Microsoft Access.

Interface del usuario: la ventana de Módulo.

La ventana de Módulo es donde se escribe, prueba y edita el código. Esta ventana en Microsoft Access el ejemplo mas obvio de los cambios realizados a la interface de programación de Microsoft Access, con la capacidad de dar formato automático y revisión de sintaxis incluidas.

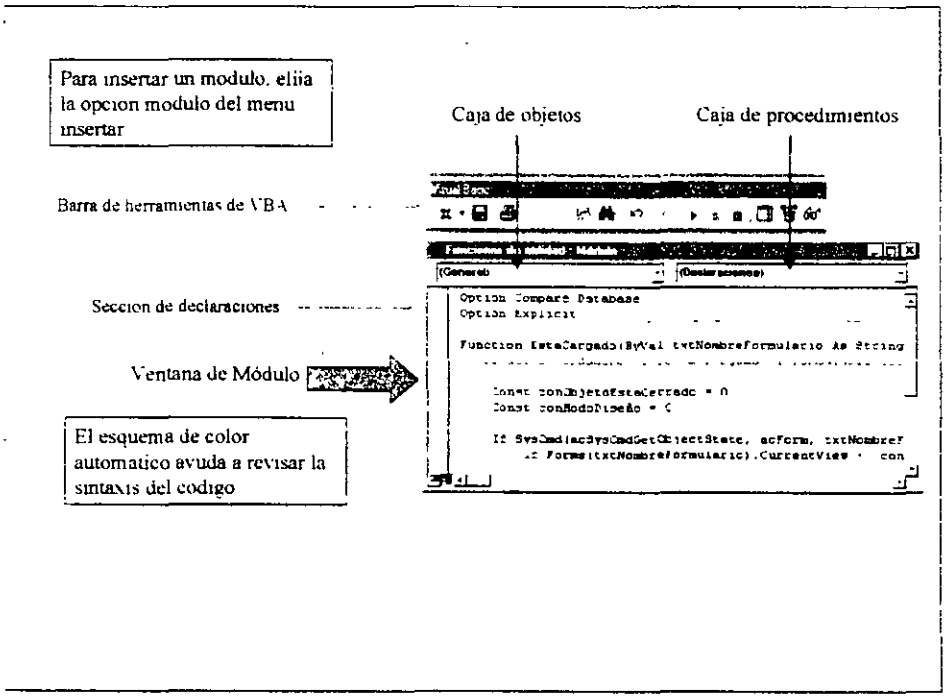
Estableciendo opciones: el separador Módulo.

En Microsoft Access existen diversas opciones para personalizar el ambiente de desarrollo. Se establecen muchas de las opciones, tales como se desea que se visualice el código, en el separador Módulo del cuadro de diálogo Opciones.

Desplazándose por el código.

El editor de ***Visual Basic para Aplicaciones*** incluye diversas opciones de navegación para desplegar ampliamente el código.

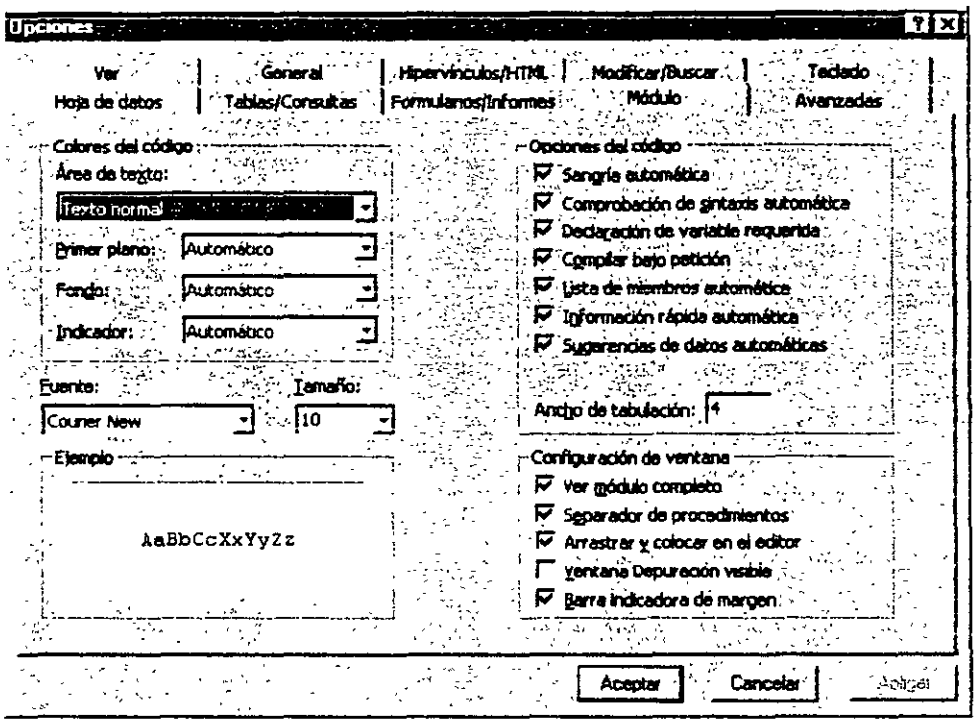
Interface del usuario: la ventana de Módulo.



La ventana de Módulo es la ventana en la cual se escribe, edita y despliega el código de ***Visual Basic para Aplicaciones***. Los elementos que incluye el módulo son:

Componente de la interface	Tiene este propósito	Para usar este componente
Caja de objetos.	Muestra el nombre del objeto seleccionado	De click en la flecha a la derecha de la caja de objeto para desplegar una lista de todos los objetos asociados con el módulo.
Caja de procedimientos.	Lista todos los eventos reconocidos por **Visual Basic** para la forma o control mostrado en la caja de objeto.	De click en la flecha a la derecha de la caja de procedimientos y seleccione un evento. La ventana de módulo mostrara el procedimiento de evento asociado con ese evento.
Barra de división.	Divide la ventana de Módulo en dos paneles horizontales, cada uno de los cuales puede desplazarse independientemente. Puede con esto visualizar diferentes partes del código al mismo tiempo. La información que aparece en las cajas de Objetos y Procedimientos aplica al código del panel que tiene el foco.	Arrastre esta barra hacia abajo para dividir la ventana del Módulo Arrastrar la barra a la parte superior de la ventana cierra el panel. La barra de división se localiza debajo de la barra de titulo en la parte superior de la barra de desplazamiento vertical.
Sección de declaraciones	Contiene instrucciones que definen variables, constantes, tipos definidos por el usuario y procedimientos externos. La sección de declaraciones de un módulo esta separada de los procedimientos, y las declaraciones en esta sección son accesibles a cada procedimiento dentro del módulo.	Teclee instrucciones en esta sección
Barra de herramientas de Visual Basic	Los botones en la barra de herramientas de **Visual Basic** corresponden a los comandos de programación mas comúnmente usados	La ayuda esta disponible para cada botón en esta barra. Solo coloque el puntero del ratón encima del botón y se mostrara la información especifica

Estableciendo opciones: el separador Módulo.



El separador Módulo en el cuadro de dialogo Opciones (Menú Herramientas) es donde se pueden establecer las opciones ambientales y de edición para el código. Esto proporciona una opción para poder implementar o no situaciones tales como:

- Revisión sintáctica automática.
- Declaración de variables requerida.
- Vista completa del módulo.
- Cambiar los colores predeterminados.
- Otras opciones

▶ **Para tener mas información sobre las distintas opciones en este separador:**

1. Entre al submenú Opciones del menú Herramientas
2. Seleccione el separador Módulo.
3. De click al botón de ayuda y luego de click a la opción.

Desplazándose por el código.

Puede desplazarse a través del código de los módulos una línea o una pagina a la vez, mediante búsqueda o mediante brincos directos a la definición del procedimiento. Las opciones de navegación son:

Comandos estándar de edición: las funciones típicas de edición de textos, tales como teclas de flechas, CTRL + PageUp y CTRL + PageDown, y las barras de desplazamiento, están todas disponibles en los módulos de código.

El comando Buscar en el menú Edición puede buscar un texto especifico. Puede limitar la búsqueda a secciones especificas del código, tales como el procedimiento o el módulo actual, o el libro de trabajo completo.

SHIFT + F2 para encontrar la definición de Procedimientos. En cualquier momento que se encuentre una llamada a un procedimiento y se desee visualizar el código de dicho procedimiento, solo presione SHIFT + F2. El editor de ***Visual Basic para Aplicaciones*** salta a la definición de dicho procedimiento, se encuentre o no en el mismo módulo (el alcance de la búsqueda esta limitada a los módulos disponibles)

Examinador de objetos para visualizar procedimientos. El Examinador de objetos puede mostrar los procedimientos disponibles en cualquier módulo o librería activo o referenciado.

▶ **Para mostrar el Examinador de Objetos:**

Del menú Ver, elegir Examinador de Objetos (también se puede dar click en el icono del Examinador de objetos en la barra de herramientas de **Visual Basic** o presionar F2)

Comando Opciones en el menú Herramientas. El separador Módulo en la caja de dialogo Opciones contiene varias opciones que permiten personalizar la forma de ver el código. El separador Módulo se analizara mas adelante en este capitulo.

Código de colores de sintaxis. Usando las opciones para Código de Colores en el separador Módulo del cuadro de dialogo Opciones, puede especificar la apariencia del código de ***Visual Basic para Aplicaciones***. Las opciones de Colores de Código determinan los colores del fondo y primer plano usados para el tipo de texto seleccionado dentro de un módulo.

Carácter de continuación de línea. Ahora puede usarse el carácter subrayado (_) para continuar una línea en el código Esto facilita la lectura del código debido a que no se tiene que desplazar hasta el final de la línea.

1.2 Variables.

Instrucciones empleadas para declarar variables.

Aunque ***Visual Basic para Aplicaciones*** no requiere que se declaren las variables antes de usarlas, el código será mas fácil de depurar y mantener si se declaran todas las variables.

`Option Explicit`. La instrucción `Option Exlicit` obliga a declarar explícitamente todas las variables en un módulo. Esta instrucción debe aparecer antes de cualquier otra instrucción que declare variables o defina constantes (instrucciones `Dim`, `Private`, `Public`, `ReDim` o `Static`)

Si trata de usar el nombre de una variable no declarada, ocurrirá un error en tiempo de compilación. El uso de **Option Explicit** evita que se teclee incorrectamente el nombre de una variable existente, o evita la confusión en el código cuando el alcance de una variable no es claro.

Se tiene la instrucción **Option Explicit** agregada de manera automática al inicio de cada nuevo módulo.

▶ **Para implementar automáticamente la instrucción** `Option Explicit`

1. Del menú Herramientas, elegir Opciones y el separador Módulo.
2. En la lista de Opciones de Código, seleccionar la opción Declaración de variable requerida
3. De click en el botón Aceptar.

Tipos de datos. La característica de la variable que determina que tipo de datos puede contener. Los tipos de datos incluyen:

- `Boolean` (Lógicos)
- `Integer` (Enteros)
- `Long` (Entero largo)
- `Currency` (Moneda)
- `Single` (Simple)
- `Double` (Doble)
- `Date` (Fechas)
- `String` (Cadenas de caracteres)
- `Variant` (Variable) tipo predeterminado
- `Object` (Objeto)
- Tipos definidos por el usuario

`Dim` declara una variable como un tipo particular de datos y asigna un espacio de almacenamiento para ella.

Sugerencia: por convención, las instrucciones `Dim` aparecen al inicio de un procedimiento o modulo.

`ReDim` o `ReDim Preserve`. `ReDim` declara un arreglo dinámico de variables y asigna o reasigna espacio de almacenamiento para variables de cualquier tipo `ReDim` Preserve también declara un arreglo dinámico de variables y asigna o reasigna espacio de almacenamiento, además preserva cualquier dato que exista previamente en el arreglo.

`Arrays`. Un conjunto de elementos indexados secuencialmente el cual tiene el mismo tipo de datos. Cada elemento de un arreglo tiene un numero de índice único que lo identifica. Los cambios realizados a un elemento de un arreglo no afecta a los otros elementos.

`Static`. Cuando la instrucción `Static` se coloca frente a una declaración de variable a nivel de procedimiento, la variable retiene su valor mientras el código es ejecutado, no solo cuando se ejecuta el procedimiento. Las variables no declaradas como `Static` pierden su valor entre llamadas al procedimiento.

`Public/Private`. Las instrucciones `Public` y `Private` son empleadas al nivel de módulo para determinar la visibilidad de una variable o constante en otros módulos. Las variables o constantes declaradas como `Public` son visibles desde otros módulos; aquellas declaradas como `Private` no.

Note que la instrucción `Option Private Module` puede reemplazar a la instrucción `Public`.

`Const`. Aunque la palabra reservada `Const` no trata con variables per se, esta es asociada. La instrucción `Const` declara constantes para emplearse en lugar de valores literales. Las constantes pueden ser publicas o privadas, y de cualquier tipos de datos excepto objetos.

Ejemplos de declaración de variables.

```
Option Explicit ' Declaracion de Variable requerida
Const MinWage =4 25
Dim MiNombre As String * 20 ' Cadena de 20 caracteres
Dim ListaDeNombres( ) As String
Dim Edad as Integer
Dim TieneAutoGrande As Boolean

Sub MuestraVariables( )
        ReDim Preserve ListaDeNombres(1 To 20) As String
        Static TimesRun As Integer
        Dim cualquiera ' Declarada como variant
        TimesRun=TimesRun + 1
        MsgBox "El procedimiento se ha ejecutado " & TimesRun & " veces"
        MiNombre = "Joe"
        ListaDeNombres(1) = MiNombre
End Sub
```

El código de ejemplo mostrado arriba muestra como se usan varias de las instrucciones de declaración de variables en ***Visual Basic para Aplicaciones***. Cuando se usa la instrucción **Option Explicit**, debe aparecer siempre en la primera línea

Convenciones de nombres

Las convenciones para los nombres ayudan a los programadores de **Visual Basic** para:

- Estandarizar la estructura, estilo de codificar y lógica de una aplicación.
- Crear código fuente preciso, legible y no ambiguo.

- Ser consistente con otras convenciones de lenguajes (como importantes, la Guía de Programadores de Visual Basic y la notación Húngara de Windows C).
- Definir los requerimientos mínimos necesarios para realizar lo anterior.

Tipos definidos por el usuario.

Cuando el conjunto estándar de tipos de datos no satisface nuestras necesidades, pueden crearse tipos definidos por el usuario para que hagan exactamente lo que uno necesita.

Agrupar variables múltiples en una estructura combinada simple. Dependiendo de la información en un programa, puede ser mas útil el crear tipos definidos por el usuario que tratar de usar uno o mas combinaciones de los tipos estándar de datos de **Visual Basic**. En el ejemplo mostrado arriba, el tipo de datos definido por usuario "EmpRecord" consiste del tipo de información que se espera encontrar en un registro de empleado: nombres y apellidos, sueldo y fecha inicial de información.

Definición con la instrucción Type. Se definen tipos definidos por el usuario mediante la instrucción **Type**, la cual debe aparecer dentro de la sección de declaraciones de un módulo estándar.

```
Type FontDisplay
      FontType As String
      FontSize as Long
      FontColor as String
End Type
```

Nota: solo se puede usar la instrucción **Type** a nivel de módulo

Declaración con la instrucción Dim. Se usa la instrucción **Dim** para declarar variables con el nuevo tipo, como se muestra en el siguiente ejemplo:

```
Dim MiLetra As FontDisplay
```

Se pueden incluir arreglos de cualquier tipo dentro de los tipos definidos por el usuario

1.3 Estructuras de control.

Esta sección presenta nuevas instrucciones, estructuras de control y palabras reservadas que son mejorados en ***Visual Basic para Aplicaciones***.

Instrucción With...End With

Esta sección presenta a la instrucción **With...End With** como una forma eficiente de realizar varios cambios a un objeto simple. Empleando **With...End With** involucra menor procesamiento para el código, y así resulta en una ejecución mas rápida del código.

Argumentos opcionales.

Las siguientes dos secciones cubren algunas mejoras del lenguaje que proveen de mayor flexibilidad cuando emplean argumentos. En Microsoft Access, pueden ser creados procedimientos con argumentos opcionales. Esta sección da ejemplos de como se emplea la palabra reservada **Optional**.

Palabra clave ParamArray.

La palabra clave **ParamArray** no es nueva en Microsoft Access (se introdujo en la versión 95). Provee de un camino para pasar un numero dinámico de argumentos a una función.

For...Each Loop

Esta sección revisa el empleo del ciclo **For...Each** como un medio para repetir un grupo de instrucciones para cada elemento de un arreglo o colección.

Instrucción With...End With

La instrucción estructura **With...End With** ejecuta una serie de instrucciones sobre un objeto simple o un tipo definido por el usuario. Esto permite desarrollar una serie de instrucciones sobre el objeto especifico sin volver a cualificar el nombre del objeto.

Facilita la lectura y escritura del código. Si de desea cambiar diferentes propiedades de un objeto simple, es mas conveniente colocar las instrucciones de asignación de propiedades dentro de la estructura de control **With**. Sintácticamente, el bloque de código **With...End With** es mas fácil de leer, debido a que las instrucciones de asignación se encuentran sangradas.

Ejecución mas rápida de muchas instrucciones individuales. La instrucción **With...End With** incremente la eficiencia de programación debido a que especifica el objeto solo un vez. Esto elimina las referencias redundantes y acelera la ejecución del código.

El siguiente ejemplo ilustra el uso de la instrucción **With** para asignar valores a diferentes propiedades del mismo objeto.

```
Function UsoDeWithEndWith( )
      Dim f As Form
      Set f = Form_Employee

      With f
            .caption = "emp"
```

```
            .scr lbars = 3
            With .detail
                  .backcolor = 255
                  .height = 1440
            End With
      End With

End Function
```

Anidamiento. Pueden anidarse las instrucciones **With**, colocando un bloque **With** dentro de otro. Sin embargo, debido a que los miembros de los bloques **With** externos están mascarados por los bloques internos, debe proveerse una referencia totalmente cualificada de objeto en el bloque interno con respecto a los objetos del bloque externo. Por ejemplo:

```
Function WithAnidado (f as Form)
      With f
            .caption = "Nuevo titulo"
            With f.LastNameTextBox
                  .value = "Smith"
            End With
      End With
End Function
```

Nota: una vez que se entra a un bloque **With**, no puede cambiarse la variable del objeto. Así pues, no puede usar una instrucción **With** simple para afectar un numero de objetos diferentes.

Importante: no salte dentro o fuera de los bloques **With**. Si las instrucciones dentro de un bloque **With** son ejecutadas, pero no se ejecutan las instrucciones **With** o **End With**, pueden ocurrir errores y un comportamiento no predecible.

Argumentos opcionales.

```
Function OptionalArguments(CustName As String, Optional CustAge)
' Esta funcion demuestra el uso de arguemntos opcionales

If IsMissing(CustAge) Then
   MsgBox ("Nombre es " & CustName)
   MsgBox ("Edad no disponible")
Else
   MsgBox ("Nombre es " & CustName & " y la edad es " & CustAge)
End If

End Function
```

Permite elegir cuando y cuando no se proporciona un argumento. Use la palabra clave **Optional** para indicar que un argumento no es requerido.

Cuando se use la palabra clave **Optional**, deben observarse las siguientes reglas:

- Deben declararse los argumentos opcionales al final de la lista de argumentos requeridos.
- Declare todos los argumentos opcionales como tipo **Variant**.
- No se puede usar **Optional** para ningún argumento si se esta usando la palabra clave **ParamArray**.

Los siguientes ejemplos muestran el uso correcto de los argumentos opcionales.

```
Function DevuelveNombre(Optional Fname, Optional Lname)
Function DevuelveNombre(Fname, Optional Lname)
Function DevuelveNombre(Fname As String, Optional Lname)
Function DevuelveNombre(Fname As String, Optional Lname As Variant)
Function DevuelveNombre(Fname, Optional Lname)
```

Los siguientes ejemplos muestran el uso incorrecto de los argumentos opcionales

```
Function DevuelveNombre(Optional Fname, Lname)
Function DevuelveNombre(Fname As String, Optional Lname As String)
Function DevuelveNombre(Fname , Optional ParamArray Horas() As String)
Function DevuelveNombre(ParamArray As Hours(), Optional Lname As String)
```

Palabra clave ParamArray.

Para pasar un numero dinámico de variables a un procedimiento en Microsoft Access puede usarse la palabra clave **ParamArray**.

Usar la palabra clave ParamArray para pasar arreglos como argumentos en procedimientos

Solo use **ParamArray** como el ultimo argumento en la lista de argumentos de un procedimiento. Esto indica que el argumento final es un arreglo opcional de elementos **Variant**. No puede usar **ParamArray** con las palabras clave **By Val**, **By Ref** u **Optional**.

La siguiente función toma dos argumentos. Toma LastName y un numero de horas por día para cada empleado. Emplea la función **Ubound** para determinar cuantos elementos hay en el arreglo. Entonces se suman esos elementos y muestra el resultado. Cada empleado puede tener un numero variable de horas durante un periodo de pago.

```
Function UsingParamArray(LastName As String, ParamArray Horas())
      Dim I As Integer
      Dim SunningSum as Double

      For I = 0 to Ubound(Horas)
            RunningSum = Runningsum + Horas(I)
      Next I
      Debug.Print "Total de horas para " & LastName & " = " &" RunningSum
End Function
```

For...Each Loop

Repite un grupo de instrucciones para cada elemento en un arreglo o colección.

For...Each es útil cuando se trabaja con objetos y colecciones., no se necesita conocer exactamente el numero de elementos en el grupo.

El siguiente ejemplo muestra como se emplea **For...Each** para ciclar a través de los elementos en un arreglo.

```
Function ForEachLoop()
      Dim MumberList(3) As Integer
      Dim Number As Variant

      NumberList(0) = 5
      NumberList(1) = 12
      NumberList(2) = 23
      NumberList(3) = 34

      For Each Number in NumberList
            Debug.Print Number
      Next
End Function
```

1.4 Cualidades de procedimientos Sub.

Los procedimientos **Sub** desarrollan una operación o una serie de operaciones, mas no regresan valores.

Unidad de código Los procedimientos **Sub** son unidades de código debido a que contienen puntos de inicio y fin distintivos, además de que el cuerpo del código realiza tareas discretas.

Encerrado por instrucciones Sub y End Sub. Todos los procedimientos **Sub** inician con la instrucción **Sub** y terminan con la instrucción **End Sub**. El nombre del procedimiento siempre se encuentra seguido de un paréntesis, aun si no hay argumentos. Por ejemplo:

```
Sub GetData()
...
End Sub
```

Puede tomar argumentos. Puede pasar información a un procedimiento **Sub** en la forma de argumentos dentro del paréntesis que sigue al nombre del procedimiento.

```
Sub GetData (FileName As String)  'El parentesis es requerido
...
End Sub
```

Hay que proporcionar los valores cuando se llama al procedimiento.

```
FileName = "DATAFILE.DAT"
GetData FileName  ' Los parentesis no son requeridos en la llamada al Sub
```

Los procedimientos que emplean argumentos deben ser llamados desde otro procedimiento **Sub** o **Function**; no pueden ser invocados desde Microsoft Access.

No devuelven valores. Un procedimiento **Sub** no regresa un valor a la instrucción que lo ha llamado.

Un procedimiento Sub puede, sin embargo , pasar información de regreso al código invocante mediante la modificación de las variables que pueden accesar tanto el procedimiento Sub como el código que llama. Esto incluye cualquier argumento pasado al procedimiento Sub, en previsión de que se pasen los argumentos por referencia, el cual es el método predeterminado

Llamado a procedimientos Sub desde la ventana de depuración. Cuando se llamas procedimientos Sub desde la ventana de depuración, puede usar referencia total al objeto contenien do el procedimiento Sub. Por ejemplo, para llamar al procedimiento Test() del Módulo1n se puede referir a el como·

```
Test
Módulo1.Test
Test x ' conde x es una variable
Módulo1.Test x
```

Nota: Microsoft Access permite tener dos procedimientos **Sub** con el mismo nombre, siempre y cuando se encuentren en módulos separados. Cuando se invoca al procedimiento, debe cualificarse con el nombre del módulo

1.5 Cualidades de procedimientos Function.

Los procedimientos **Function** tienen cualidades similares a las de los procedimientos **Sub**, excepto que en lugar de desarrollar una tarea, los procedimientos **Function** devuelven un valor.

Encerrado por instrucciones Function y End Function. Cada procedimiento **Function** inicia con la instrucción **Function**, seguido del nombre de la función y el tipo de dato que se regresa, y termina con la instrucción **End Function**. El nombre de la función siempre esta seguido de paréntesis, aun si no hay argumentos.

```
Function IsFileOpen (MyFile As String) As Boolean
      ...
      IsFileOpen = Decision   ' argumento opcional de devolucion
      Valor
End Function
```

Pueden tomar argumentos. Pueden pasarse información al procedimiento **Function** en la forma de argumentos dentro del paréntesis que sigue al nombre del procedimiento.

```
' Los parentesis son requeridos en la declaracion de la Funcion
Function IsFileOpen (FileName$) As Boolean
      ...
End Function
```

Hay que proporcionar los argumentos cuando se invoque al procedimiento **Function**.

Tiene valor de devolución. El procedimiento **Function** siempre devuelve un valor. Si una función requiere devolver valores múltiples, puede hacerse modificando los valores de cualquier argumento pasado a esta o modificando las variables accesibles tanto por la función como por el procedimiento que invoca.

```
FileOpenStatus As Boolean
...
FileName$ = "DATAFILE.DAT"
FileOpenStatus = IsFileOpen(FileName$)
```

Llamando procedimientos Function desde la ventana de depuración

Cuando se invocan procedimientos **Function** desde la ventana de depuración, puede usar referencia completa al objeto conteniendo el procedimiento **Function**. Por ejemplo, para invocar la función Test() del Módulo!, se puede referir a ella como.

```
Test
?Test
?Módulo1.Test
Test(x)           'Donde x es una variable
?Test(x)     ' Donde x es una variable
?Módulo1.Test(x)
?Módulo1.Test()
```

Nota: Microsoft Access permite tener dos procedimientos **Function** con el mismo nombre, siempre y cuando se encuentren en módulos separados. Cuando se invoca al procedimiento, debe cualificarse con el nombre del módulo.

1.6 Alcance y visibilidad.

El alcance define la visibilidad de una variable, procedimiento u objeto.

Existen tres tipos de niveles de alcance en **Visual Basic**, como se describe en la siguiente tabla

Nivel de alcance	Visibilidad
Local	Una variable definida al nivel de procedimiento que es visible solo en el procedimiento en el cual esta declarada
Módulo	Una variable declarada al nivel de módulo que es visible a todos los procedimientos dentro del módulo en el cual esta declarada
Publica	Una variable declarada al nivel de módulo que es visible a todos los procedimientos dentro de todos los proyectos.

Declarar una variable al nivel del procedimiento o del módulo depende de que otros procedimientos dentro de la aplicación usaran esa variable.

Microsoft Access usa la palabra clave **Public** en vez de **Global**. De manera predeterminada, todos los procedimientos son públicos a menos que se use la palabra clave **Private**.

La siguiente tabla comprara el alcance en Microsoft Access con otras versiones.

Alcance	Sintaxis en versión 1.x y 2.0	Sintaxis en Microsoft Access
Procedimiento disponible a todos los módulos	**Function abc( )**	Public Function abc( ) O **Function abc( )**
Procedimiento restringido al módulo actual	**Private Function abc( )**	**Private Function abc( )**
Variables dentro de un procedimiento son estáticas	**Static Function abc( )**	**Public Static Function abc( )** O **Static Function abc( )**
Variable disponible solo al procedimiento	**Dim X** en el procedimiento	**Dim X** o **Private X** en el procedimiento
Variable disponible a todos los procedimientos en el módulo	**Dim X** en la sección de declaraciones	**Dim X** o **Private X** en la sección de declaraciones
Variables disponibles a todos los procedimientos en todos los módulos	**Global X**	**Public X**

Nota: por cuestiones de compatibilidad con versiones anteriores, puede seguir usando la palabra clave Global; din embargo, es preferible que use la palabra clave **Public**.

1.7 Preguntas de Repaso.

1. ¿Cuando una línea simple de código es demasiado larga para ajustar en un renglón, puede colocarse esta línea en varios renglones mediante el uso de que carácter?

2. Si se encuentra una llamada a un procedimiento, ¿Como podemos saltar a ese procedimiento?

3. ¿Cual de los siguientes es un nombre de variable valido?

   a. 1Amount

   b. $Amount

   c. Company.Name

   d. Company_name

4. ¿Cual es el propósito de la instrucción **Option Explicit**?

5 El siguiente código de ejemplo, ¿que tipo de procedimiento es CalcTax?

```
Dim Tax As Currency
Tax = CalcTax(99.99)
```

6 ¿Cuál de las siguientes instrucciones, cuando se colocan el la sección de declaraciones de y módulo, puede declarar una variable que pueda ser visible desde otros módulos?

   a. Private S As String

   b. Public S As String

   c. Static S As String

   d. Dim S As String

7. En el código de ejemplo siguiente, un tipo definido por el usuario es declarado. ¿Como podría referirse al elemento City?

```
Type Employee
   First Name As String
   LastName As String
   Address As String
   City As String
   State As String
   Zip As String
End type


Sub UDT_Example( )
   Dim MyEmployee As Employee
End Sub
```

8. ¿Cual de las siguientes instrucciones puede ejecutar un procedimiento Sub llamado Test desde el panel Immediate de la ventana de depuración?

   a. Test( )
   b. ? Test( )
   c. Test
   d. ? Test

9. ¿cual de las siguientes instrucciones podría ejecutar un procedimiento de Función llamado Test que requiere un argumento?

   a. Test(X)
   b. Test
   c. Test( )
   d. Test X

10. ¿cuál es el alcance de un procedimiento de forma predeterminada?

11. ¿Cual de los siguientes no es un tipo de datos valido en ***Visual Basic para Aplicaciones***?

    a. `Date`

    b. `Char`

    c. `Boolean`

    d. `Single`

12. ¿Que función es empleada para determinar si un argumento opcional fue proporcionado cuando se llamo un procedimiento?

    a. `IsNull()`

    b. `Exist()`

    c. `IsMissing()`

    d. `IsOptional()`

13. ¿Donde se encuentra el error de sintaxis en el código de ejemplo siguiente?

```
Sub ListPeople (ParamArray People( ))
  Dim p As String

  For Each p in People( )
        Debug.Print p
  Next p
End Sub
```

Capítulo 2: Programación orientada a objetos y acceso a datos externos.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Diferenciar entre objetos y colecciones.
- ☑ Diferenciar entre propiedades y métodos.
- ☑ Describir el uso del modelo de objetos de Microsoft Access.
- ☑ Describir y usar el modelo de objetos de acceso a datos (DAO)
- ☑ Crear nuevas propiedades para el modelo DAO.

2.1 Definiendo objetos y colecciones.

¿Qué es un objeto? En ***Visual Basic para Aplicaciones***, un objeto es algo que se puede controlar. El objeto se controla usando sus atributos y acciones, o sus *propiedades* y *métodos*.

¿Que es una colección? ***Visual Basic para Aplicaciones*** puede referirse a un grupo de objetos similares colectivamente. En Microsoft Access, una colección es un conjunto de objetos relacionados - por ejemplo, todas las formas abiertas.

objetos conteniendo ningún o varias colecciones de tipos variables Los objetos y colecciones están relacionados de la siguiente forma. Una colección contiene ningún o muchos objetos del mismo tipo. Un objeto contiene ningún o muchas colecciones de varios tipos.

2.2 Propiedades y métodos.

Esta sección presenta las propiedades y métodos y discute como se aplican a objetos y colecciones.

Definiendo propiedades y métodos.

Dentro de Microsoft Access, los objetos y colecciones proveen de un marco para usar código para crear y manipular los componentes de un sistema de base de datos. Dentro de este marco, se aplican propiedades y métodos a esos objetos y colecciones

¿Que son las propiedades? Todos los objetos tiene propiedades. Las propiedades describen el estado de un objeto. Las propiedades son similares a los campos de una estructura de datos: ellos contienen siempre un solo valor Las propiedades pueden ser de solo lectura (r/o), de lectura/escritura (r/w) o de solo escritura (r/o).

Las propiedades pueden tener diferentes tipos de datos: cadenas de caracteres (strings) y números son las mas comunes. No puede cambiarse el tipo de dato de una propiedad.

Escribir un valor en una propiedad altera el estado de un objeto. Por ejemplo, establecer la propiedad Visible de una forma ocasiona que la forma aparezca o desaparezca.

¿Qué son los métodos?. Los métodos realizan alguna acción sobare los objetos o colecciones. Cada objeto posee un conjunto de métodos que sabe como implementar. Estos métodos son la *única* forma de interactuar con el objeto.

Los objetos externos no pueden cambiar las propiedades de un objeto directamente. Solo pueden llamar métodos del objeto del que se requiere que cambie una propiedad.

Propiedades y métodos para colecciones.

Hay algunas propiedades y métodos que son generales para todas las colecciones.

Propiedades (se aplican a todos) Cada colección tiene exactamente una propiedad llamada **Count**. Esta propiedad especifica el numero de objetos contenidos en esa colección.

Métodos (puede no aplicar a todos) Cada colección puede tener los siguientes métodos:

- **Append** agrega un objeto a la colección.
- **Delete**: elimina un objeto de la colección.
- **Refresh**: actualiza la colección con cualquier cambio hecho a la membresía de la colección.

Nota: la propiedad **Count** aplica a todas las colecciones, pero los métodos **Append**, **Delete** y **Refresh** pueden o no aplicar a las colecciones

2.3 Modelo de objetos de Microsoft Access.

Esta sección define el modelo de objetos de Microsoft Access y da una explicación de su estructura. La sección se enfoca en el objeto **Application** y sus propiedades y métodos.

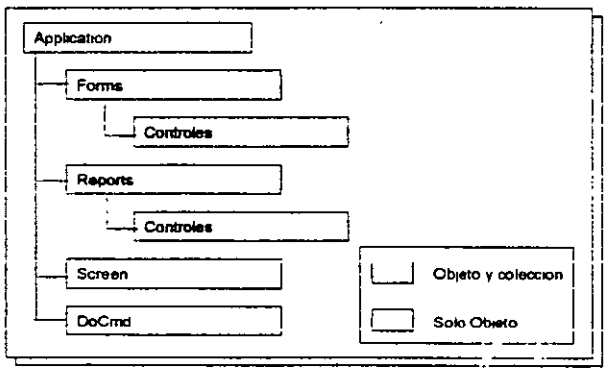
Definición de la estructura.

Este tópico introduce el concepto de una estructura de objetos y colecciones que esta jerárquicamente organizada. Nos referimos a esta estructura particular como el modelo de objetos de Microsoft Access. Es importante entender el concepto de jerarquía debido a que el modelo DAO (analizado posteriormente) también contiene una estructura jerárquica. Entendiendo las relaciones entre los diferentes objetos y colecciones dentro de la jerarquía es el primer paso para ser capaz de manipular los objetos en Microsoft Access y DAO en el código de Visual Basic.

Propiedades y métodos del objeto Application.

Ya se ha visto que las propiedades y métodos son la forma principal de trabajar con los objetos en Visual Basic. El objeto **Application** es el objeto de mas alto nivel en el modelo de objetos de Microsoft Access y se refiere a la aplicación completa de Microsoft Access. Este tópico cubre las propiedades y metodos mas comúnmente usadas del objeto **Application**.

Definición de la estructura.



Estructura jerarquica de objetos y colecciones de objetos
- Interface del usuario de Microsoft Access
- Captura de datos y elementos de salida (formularios, reportes)
- Objetos que puedan manipularse en VBA

Para poder trabajar con los objetos dentro del código, es necesario entender la estructura del modelo de objetos de Microsoft Access

estructura jerárquica de objetos y colecciones de objetos. El modelo de Microsoft Access incorpora objetos y colecciones de objetos dentro de una estructura jerárquica. Los objetos de Microsoft Access con representaciones codificadas de elementos de la aplicación, las cuales generalmente controla el usuario, tal como una interface de usuario.

Esta jerarquía de objetos determina cuales objetos pueden contener otros objetos. el objeto Application es el objeto de mas alto nivel en la jerarquía y contiene a todos los demás objetos y colecciones.

El objeto Application se refiere a la aplicación completa de Microsoft Access. Se emplea el objeto Application par aplicar métodos o establecer propiedades a la aplicación en general. Por ejemplo, puede emplearse la propiedad **MenuBar** para especificar una barra de menú personalizada para toda la aplicación. Puede usarse además el método **SetOption** para establecer parámetros en el cuadro de dialogo Opciones de la aplicación.

```
Application.,Setoption "Mostrar barra de estado", True
```

Propiedades y métodos del objeto Application

El siguiente ejemplo aplica un filtro a una forma. El filtro permite al usuario seleccionar registros especificos de una tabla llamada "Productos", basado en un criterio.

```
Sub SetFilter()
      Dim frm As Form, strMsg As String
      Dim strInput As String. StrFilter As String

      DoCmd.OpenForm "Productos"
      Set frm = Forms!Productos
      StrMsg = "Introduzca una o mas letras del nombre del producto" _
      & " segidas de un asterisco"
      strInput = InputBox(strMsg)
      strFilter = BuildCriteria("ProductName", dbText, strInput)
      frm.Filter = strFilter
      frm.FilterOn = True
End Sub
```

Propiedades. La siguiente tabla resume las propiedades del objeto **Application**

Nombre de la propiedad	Aplicación	Descripción
`CurrentObjectName`	Solo lectura	Devuelve el nombre del objeto de base de datos activo
`CurrentObjectType`	Solo lectura	Devuelve el tipo del objeto de base de datos activo
`MenuBar`	Lectura/Escritura	Especifica la macro de barra de menú que se ejecuta para desplegar una barra de menú personalizada para una base de datos, forma

		o reporte
`DBEngine`	Solo lectura	Representa el motor de base de datos Microsoft Jet

Métodos. La siguiente tabla resume los métodos de solo lectura del objeto **Application**

Método	**Descripción y uso**
`GetOption`	Devuelve el valor actual de una opción del cuadro de dialogo **Options**
`SetOption`	Establece el valor actual de una opción en el cuadro de dialogo **Options**
`BuildCriteria`	Este método permite la fácil construcción de criterios para filtros basados en entradas del usuario. Analiza el argumento de la expresión de la misma forma en que seria analizada si se hubiera introducido en la retícula del diseño de consultas o en el modo Filtrado de una forma
Método	**Descripción y uso (Continuación)**
`CloseCurrentDatabase`	Este método puede emplearse para cerrar la base de datos actual desde otra aplicación que haya abierto la base de datos mediante automatización OLE
`OpenCurrentDatabase`	Este método abre una base de datos existente y la deja como base de datos actual. Este método se emplea para abrir bases de datos desde otra aplicación que esta controlando a Microsoft Access mediante automatización OLE
`NewCurrentDatabase`	Crea una nueva base de datos en la ventana de Microsoft Access. Puede emplearse este método para crear una base de datos desde otra aplicación que esta controlando a Microsoft Access mediante automatización OLE
`DefaultWorkspaceClone`	Crea un nuevo objeto de espacio de trabajo (**Workspace**) sin necesidad que el usuario se vuelva a dar de alta
`Echo`	Especifica si Microsoft Access repinta o actualiza el monitor de la computadora. El método Echo no suprime la exhibición de cuadros de dialogo modales, tales como mensajes de error o formas pop-up, tales como hojas de propiedades
`Quit`	Se emplea para salir de Microsoft Access. Se pueden emplear una de varias opciones para guardar el objeto de base de datos antes de salir
`RefreshTitleBar`	Actualiza la barra de titulo de Microsoft Access después que han sido establecidas las propiedades **AppTitle** o **AppIcon** en Visual Basic
`Run`	Se emplea para ejecutar un procedimiento **Function** definido por el usuario o un procedimiento **Sub**. Este método es útil cuando se esta controlando a Microsoft Access desde otra aplicación mediante automatización OLE

2.4 Modelo de objetos de acceso a datos (DAO).

Adicionalmente a los objetos definidos por Microsoft Access, el motor de base de datos Microsoft Jet define objetos separados que controlan tareas de administración de datos de la aplicación. tales como tablas, consultas, relaciones e índices. Nos referimos a estos objetos como *objetos de acceso de datos*. Esta sección introduce el modelo DAO.

Vista conceptual del modelo DAO

Este tópico da una breve semblanza conceptual de DAO empleando la analogía del planeta tierra. Adicionalmente, el tópico lista algunos escenarios comunes para usar DAO y revisa parte de la terminología empleada en DAO

La jerarquía DAO.

DAO opera dentro de un marco jerárquico. La jerarquía consiste de objetos y colecciones. Las colecciones contienen objetos. esta sección además cubre la navegación entre los diferentes niveles de la jerarquía.

Vista conceptual del modelo DAO.

Los objetos definidos por el motor de base de datos Jet además existen dentro de la estructura de un modelo de objetos.

¿Que es DAO? Los objetos de acceso a datos (DAO) son un conjunto de objetos programables de alto nivel, que facilitan la programación y el uso de fuentes de datos. El modelo DAO consiste en una organización jerárquica de colecciones y objetos, con sus métodos y propiedades

Los usos comunes de DAO incluyen:

- Manipulación de datos.
- Procesamiento de transacciones
- Establecimiento de permisos.
- Creación de nuevos objetos de acceso de datos, por ejemplo **TableDef**, **QueryDef** y otros.

Cuando se trata con programación de bases de datos en Microsoft Access, es de gran ayuda tener un entendimiento conceptual claro de los objetos de acceso a datos y familiarizarse con la terminología asociada. He aquí algunos términos importantes con los cuales debe uno de familiarizarse:

- *Motor de base de datos*: es el motor de bases de datos Jet, usado en Microsoft Access y Visual Basic.
- *Sistema de base de datos*. Consiste de al menos un motor de base de datos, base de datos del sistema y base de datos del usuario.
- *Usuario*. Un usuario de DAO es un desarrollador que escribe código usando DAO. Existe también un objeto Usuario (**User**) especificado en el modulo, el cual se refiere a cualquiera registrado con la base de datos del sistema.

La jerarquía DAO.

Así como en el modelo de objetos de Microsoft Access, los objetos de acceso a datos pueden contener otros objetos, por ejemplo, una base de datos puede contener atablas. La contención emplea el concepto de colección de objetos contenidos que están asociados con el objeto padre

La aplicación en si misma. que en este ambiente es el motor de base de datos actual, es el objeto de acceso a datos mas exterior. Es el único objeto no contenido en nada mas. La aplicación a veces actúa como el lugar para almacenar todas las propiedades y objetos globales. La inicialización del motor de base de datos y las propiedades al nivel del sistema están asociados con el objeto de aplicación DBEngine.

La forma más común de navegar es desde un nivel de la jerarquía hacia el siguiente. Se inicia con el especificador de objeto para un contenedor (por ejemplo, una Base de Datos **Database**), y se necesita obtener un objeto especifico para uno de los contenidos (esto es, una definición de tablas o **TableDef**). En Visual Basic "." (el punto) es el operador de navegación. Así pues, si `db` es una base de datos, entonces `db.TableDefs("MiTabla")` es una tabla llamada `MiTabla` almacenada en la base de datos.

Cuando se emplea la sintaxis "." para almacenar miembros de una colección, Visual Basic siempre consulta la colección **Properties** cuando no hay una colección especifica. Por ejemplo, si se tiene una propiedad llamada "**Name**" y un campo llamado "**Name**", el siguiente código devolverá el valor de la propiedad "**Name**"

```
Algunvalor$ = MyRecordset.Name
```

En este ejemplo, Visual Basic regresa el valor de la propiedad "**Name**", debido a que el operador "." obliga al objeto a buscar en la colección de propiedades primero para el nombre del miembro dado.

2.5 DAO: objetos y colecciones de alto nivel.

En la jerarquía DAO, los objetos y colecciones de alto nivel incluyen **DBEngine**, **Workspaces** y **Databases**. Esta sección aborda los elementos de alto nivel del modelo DAO, mientras que las secciones siguientes se enfocan en las colecciones y objetos mas abajo en la jerarquía. La imagen superior muestra la jerarquía DAO de alto nivel.

Objeto DBEngine

El motor de base de datos, referido como **DBEngine**, es el objeto mas exterior del modelo DAO y contiene a todos los demás objetos. Es el motor de base de datos Microsoft Jet.

Colección Workspaces

La colección **Workspaces** contiene todos los objetos activos y no ocultos **Workspace** del objeto **DBEngine**. El objeto **Workspace** contiene **Databases**, **Users** y **Groups**.

Creación de bases de datos adicionales

Este tópico cubre el uso del objeto **Workspace** para crear bases de datos individuales. Mientras se logra esto a través de la interface del usuario, hay ocasiones cuando, como desarrollador, se desea crear bases de datos adicionales mediante código Esto debe incluir, por ejemplo, copiar un objeto dentro de una base de datos y distribuirla. Además, ahora se tiene acceso al motor de base de datos Jet a través de productos tales como Visual Basic y Visual C++, dando al desarrollador mas flexibilidad.

Colección Databases

La colección **Databases** contiene todas los objetos **Database** abiertos o creados en un objeto **Workspace** del motor de base de datos Microsoft Jet. El objeto **Database** contiene las colecciones **TableDefs**, **Recordsets**, **Containers** y **Relations**.

Objeto DBEngine

Objeto de alto nivel en la jerarquía DAO. El objeto **DBEngine** representa el motor de base de datos Microsoft Jet, y es el objeto de mas alto nivel en la jerarquía DAO Como objeto de alto nivel, contiene y controla todos los demás objetos en la jerarquía El objeto **DBEngine** no es elemento de ninguna colección.

El objeto **DBEngine** es un objeto predefinido, y no pueden ser creados objetos **DBEngine** adicionales.

Para referirse a una colección del objeto **DBEngine** o a un método, se emplea la siguiente sintaxis:

```
DBEngine.[colección | metodo | propiedad]
```

Esta sintaxis establece el objeto **DBEngine** para controlar el motor de base de datos, manipular sus propiedades e instalar el **Workspace** preestablecido.

DBEngine posee propiedades y métodos. Cada objeto DAO posee una colección de propiedades. Las propiedades pueden ser definidas por el usuario o definidas por objetos de acceso de datos individuales. Las propiedades DAO están siempre presentes como predeterminadas y no pueden ser cambiadas.

Debido a que **DBEngine** es in objeto de alto nivel y no tiene una representación en disco, no soporta propiedades definidas por el usuario a este nivel.

La siguiente tabla lista algunas de las propiedades asociadas con el objeto **DBEngine**.

Propiedad	Permiso	Descripción
Version	Solo lectura	Identifica el numero de versión del producto del motor de base de datos Microsoft Jet ejecutandose
SystemDB	Lectura/Escritura	Establece o devuelve la ruta de la posición actual del archivo de base de datos del sistema

La siguiente tabla lista algunos de los métodos del objeto **DBEngine**

Método	Descripción
Idle	Suspende el proceso de datos, habilitando al motor de base de datos para completar cualquier tarea pendiente. Se usa principalmente para bloqueos.
CompactDatabase	Compacta una base de datos de Microsoft Access.
RepairDatabase	Repara una base de datos de Microsoft Access.
RegisterDatabase	Introduce información de conectividad de una fuente de datos ODBC.
CreateWorkspace	Abre un nuevo objeto **Workspace**

DBEngine contiene otras colecciones. El objeto **DBEngine** además incluye las colecciones **Workspaces** y **Errors**. La colección predeterminada de **DBEngine** con las colecciones Workspaces. La siguiente sección de código permite accesar el espacio de trabajo predeterminado. Note que la colección **Workspaces** inicia en 0 y no en 1. así que apuntar al elemento 0 realmente refiere al primer objeto **Workspace**.

```
Dim MyWrkSpc As Workspace
Set MyWrkSpc = DBEngine.Workspaces(0)
```

Pueden ser creados nuevos objetos **Workspace** mediante el uso del método **CreateWorkspace** del objeto **DBEngine**. El siguiente código de ejemplo crea un nuevo espacio de trabajo llamado EspacioNuevo y establece su propiedad **UserName** a "invitado" sin contraseña

```
Dim WSp As Workspace
Set WSp = Dbengine.CreateWorkspace("EspacioNuevo", "invitado", "")
```

Colección Workspaces

Un espacio de trabajo define una sesión en Microsoft Access. Cada espacio de trabajo pueden tener un usuario y contraseña separados. Adicionalmente, las transacciones son independientes para cada espacio

La colección **Workspaces** contiene todos los objetos activos y no ocultos **Workspace**. La colección **Workspaces** posee los siguientes métodos y propiedades:

- Métodos – **Append**, **Refresh**
- Propiedades – **Count**

Espacio de trabajo predeterminado. El espacio de trabajo predeterminado es la sesión que es iniciada automáticamente por Microsoft Jet cuando los objetos de acceso a datos son referidos por primera vez en el lenguaje en tiempo de ejecución. Se puede establecer explícitamente el nombre de usuario y contraseña de una sesión estableciendo las propiedades **UserName** y **Password**. Si no se especifica un nombre de usuario, el motor de base de datos Jet inicializa la sesión con el usuario "Admin" y la contraseña = Null.

Revise los códigos de ejemplo de la pagina anterior donde se tenían dos espacios de trabajo El primero es el espacio de trabajo predeterminado que existía antes, y el segundo fue llamado EspacioNuevo. Podemos enumerar a través de la colección **Workspaces** y listar cada objeto **Workspace** usando el siguiente código:

```
For I = 0 to DBEngine.Workspaces.Count - 1
      Debug.Print DBEngine.Workspaces(I).Name
Next I
```

Contiene el objeto Workspace. El objeto **Workspace** soporta transacciones simultaneas, actúa como contenedor de bases de datos abiertas e identifica un contexto de seguridad para operaciones en la base de datos.

La siguiente tabla resume las propiedades asociadas con el objeto Workspace

Propiedad	Nuevo espacio	Workspace	Descripción
Name	Lectura/Escritura	Solo lectura	Expresión de texto que identifica de manera única al espacio de trabajo.
UserName	Lectura\Escritura	Solo lectura	Expresión de texto que identifica el nombre del usuario para el cual fue construido el espacio de trabajo
IsolateODBCTrans	Lectura\Escritura	Lectura\Escritura	Establece o devuelve un valor indicando cuando múltiples transacciones que involucran la misma base de datos ODBC están aisladas.

La siguiente tabla resume algunos de los métodos del objeto **Workspace**.

Método	Descripción
Close	Finaliza el espacio de trabajo
OpenDatabase	Abre una base de datos y la agrega a la colección **Databases**
CreateDatabase	Crea una nueva base de datos

Creación de bases de datos adicionales

El objeto Workspace permite crear objetos de base de datos adicionales. Pueden ser creados o abiertos objetos de base de datos adicionales desde el objeto **Workspace**. Cada base de datos adicional es accesible solo mediante código Estas bases de datos no se podrán ver en la interface del usuario de Microsoft Access. Esto proporciona gran flexibilidad al desarrollador.

El siguiente ejemplo muestra como se crea una base de datos.

```
Dim Ws As Workspace
Dim Db As Database
Set Ws = DBEngine.Workspaces(0)
Set Db = Ws.CreateDatabase("Nuevo.mdb", dbLangGeneral)
```

Nota: la constante **dbLangGeneral** es empleada para el argumento local y significa el orden de comparación para creación de bases de datos. Este argumento debe ser proporcionado o se generara un error.

Colección Databases

Contiene todos los objetos Database abiertos. La colección **Databases** contiene todos los objetos **Database** abiertos en una sesión del motor de base de datos Jet. En Microsoft Access, el miembro de la colección cero (0) es siempre la base actual. Cerrar una base de datos la remueve de la colección.

Nota: asegúrese de compactar la colección para llenar el espacio dejado por el miembro eliminado, ya que la referencia ordinal de este no será valida. Cualquier referencia al miembro devolverá el error "Objeto invalido". La colección predeterminada de **Workspace** es la colección **Databases**.

La colección **Databases** contiene los siguientes métodos y propiedades;

- Métodos – **Refresh**.
- Propiedades – **Count**.

El siguiente ejemplo lista los nombres de todas las bases de datos en la colección **Databases**

```
For I = 0 to DBEngine.Workspaces(0).Databases.Count - 1
      Debug.Print DBEngine.Workspaces(0).Databases(I).Name
Next I
```

Objetos Database. Un objeto **Database** representa una base de datos abierta. Use el método **OpenDatabase** de un **Workspace** para crear un objeto de base de datos. Puede manipular una base de datos abierta mediante el uso de sus métodos y propiedades. Puede examinar las colecciones en un objeto **Database** para encontrar información acerca de sus tablas, consultas, relaciones, formas y reportes. Puede además usar sus colecciones para modificar o crear tablas, consultas y relaciones.

Puede emplear el método **Close** para eliminar un objeto **Database** de la colección **Databases** sin borrarla del disco duro. El motor de bases de datos Jet usa **Databases(0)** para referirse la base de datos actual Se puede hacer referencia a cualquier otro objeto **Database** que sea creado y abierto mediante la propiedad **Name** del objeto, con la siguiente sintaxis:

```
Database("nombre")
```

Recuerde usar la sintaxis completa, por ejemplo:

```
Database(0)
   Database(1)
      ...
Database(x)
```

La siguiente tabla describe algunas propiedades del objeto **Database**

Propiedad	Base de datos	Descripción
Connect	Solo lectura	Provee información acerca del origen de una base de datos abierta, una base de datos empleada en una consulta de paso o una tabla adjunta.
Name	Solo lectura	El nombre de la base de datos
QueryTimeout	Lectura/Escritura	Especifica el numero de segundos que Microsoft Access espera antes de ocurrir un error de sobre-tiempo cuando se ejecuta una consulta en una base de datos ODBC
Updatable	Solo lectura	Indica cuando una base de datos puede ser modificada
AccessVersion	Solo lectura	Identifica la versión de Jet en el cual fue creada la base de datos

La siguiente tabla lista y describe los métodos del objeto **Database**

Método	Descripción
Close	Cierra una base de datos
Execute	Ejecuta una consulta de acción
OpenRecordset	Abre un conjunto de registros (**Recordset**)
CreateProperty	Crea un objeto **Property** nuevo
CreateRelation	Crea un objeto **Relation** nuevo
CreateTableDef	Crea un objeto **TableDef** nuevo
CreateQueryDef	Crea un objeto **QueryDef** nuevo

El siguiente ejemplo crea una nueva tabla por medio de código

```
Dim MiDb As Database
Dim MiTabla As TableDef

Set MiDb = DBEngine.Workspace(0).DataBases(0)
Set MiTabla = MiDB.CreateTableDef("TablaNueva")
```

2.6 Tablas, campos e índices.

Esta sección cubre los objetos y colecciones que generalmente se involucran al trabajar con tablas.

Colección TableDefs

La colección **TableDefs** contiene todos los objetos **TableDef** almacenados en una base de datos, así como las colecciones para los objetos **Field** (campo) e Index (Indice). **Database** es el objeto DAO de alto nivel que contiene las colecciones **TableDefs**.

Objeto TableDef

Un objeto **TableDef** simboliza la definición y estructura de cualquier tabla en la base de datos.

Ejemplos de usos del objeto TableDef

Este punto muestra varios ejemplos de cómo se emplea el objeto **TableDef**, incluyendo la adición y eliminación de una tabla de una base de datos y listar las tablas que no son del sistema.

Colección Fields.

La colección **Fields** contiene todos los objetos **Field** almacenados de un índice, consulta, conjunto de registros (recordset), relación o tabla. Con la excepción del objeto **Recordset**, los objetos contenidos dentro de la colección **Fields** abarca la estructura de los campos cuyos objetos representa. La colección **Fields** de in objeto **Recordset** designa un renglón de datos en una tabla.

Objeto Field.

Un objeto **Field** representa una columna de datos con un tipo común de datos y un conjunto común de propiedades.

Colección Indexes.

Una colección Indexes contiene almacenados objetos **Index** de un objeto **TableDef**.

Objeto Index.

Un objeto Index representa tanto un arreglo de valores y la unicidad de valores que dan acceso eficiente a los datos en renglones de una tabla

Colección TableDefs

La colección **TableDefs** enumera todas las tablas en una base de datos. La colección **TableDefs** contiene los siguientes métodos y propiedades:

- Propiedades – **Count**.
- Métodos – **Append**, **Delete**, **Refresh**.

Contando objetos TableDef en una colección TableDefs. Usando la propiedad **Count** de la colección **TableDefs**, puede encontrar el numero de tablas en una base de datos, tal como se muestra en el siguiente ejemplo:

```
Dim MiDb As Database
Set MiDb = DBEngine.Workspaces(0).Databases(0)
Debug.Print MiDb.TableDefs.Count
```

Objeto TableDef

El objeto **TableDef** define la estructura de una tabla.

Definición almacenada de una base o tabla adjunta. Un objeto **TableDef** representa la definición almacenada de una tabla base o adjunta. Se manipula la definición de la tabla mediante el uso del objeto **TableDef** y sus métodos y propiedades

Propiedad	TableDef nueva	TableDef existente	TableDef adjunta	Descripción
Connect	Lectura / Escritura	Solo lectura	Lectura / Escritura	Proporciona información acerca del origen de una tabla adjunta.
Name	Lectura / Escritura	Lectura / Escritura	Solo lectura	El nombre de la tabla o liga.
SourceTableName	Lectura / Escritura	Solo lectura	Lectura / Escritura	El nombre de la tabla fuente para tablas adjuntas
Updatable	TRUE	Solo lectura	FALSE	Indica cuando la definición de la tabla para un objeto **TableDef** puede ser modificado
ValidationRule	Lectura / Escritura	Lectura / Escritura	Solo lectura	Especifica una expresión que debe evaluar a VERDADERO para la terminacion exitosa del método **Update**.

La siguiente tabla lista y describe algunos de los métodos disponibles en el objeto **TableDef**

Metodo	Descripción
OpenRecordset	Ejecuta una consulta o abre una tabla base
CreateField	Crea un objeto **Field** vacío
CreateProperty	Crea un objeto **Property** vacío
CreateIndex	Crea un objeto **Index** vacío

Creación de índices. Para crear un índice en un campo de una tabla, primero es necesario accesar al objeto **TableDef**. Entonces pueden efectuarse los siguientes pasos:

1. Use el método **CreateIndex** para crear un índice.
2. Agregue un campo al índice, usando el método **CreateField** para crear un campo para el objeto **Index**.
3. Agregue el campo a la colección **Fields** del objeto **Index**.
4. Agregue el índice a la colección Indexes del objeto **TableDef**.

El siguiente ejemplo crea un índice en un campo de una tabla:

```
Sub NewIndex()
   Dim dbs As Database, tdf As TableDef, idx As Index
   Dim fld as Field

   ' Devuelve un objeto Database apuntando a la base de datos actual
   Set dbs = CurrentDb
   Set tdf = dbs.TableDefs!Employees
   Set idx = tdf.CreateIndex("LastNameIndex")
   Set fld = idx.CreateField("LastName")
   Idx.Fields.Append fld
   Tdf.Indexes.Append idx
End Sub
```

Creación de una propiedad definida por el usuario en un campo. El siguiente ejemplo crea una nueva propiedad definida por el usuario, establece su valor inicial y la agrega a la colección de **Properties** del objeto **TableDef**

```
Sub CreateNewProperty( )
      Dim dbs As Database, tdf As TableDef
      Dim prp As Property

      ' Devuelve un objeto Database apuntando a la base de datos actual
      Set dbs = CurrentDb
      Set tdf = dbs.TableDefs!Orders
```

```
    ' Crea la propiedad nueva, denota el tipo y establece el valor inicial
    Set prp = tdf.CreateProperty("LastSaved", dbText, "New")
    ' Agrega a la colección Properties del objeto TableDef
    tdf.Properties.Append prp
End Sub
```

Ejemplos de usos del objeto TableDef

Los ejemplos mostrados a continuación muestran diferentes formas de osar el objeto TableDef.

A veces, es necesario listar solo las tablas que no son del sistema. Puede llevarse a cabo esto usando una simple condición If, de acuerdo con el siguiente ejemplo:

```
Function NonSystemTables( )
    Dim MyDb As Database, I As Integer
    Set MyDb = DBEngine.Workspaces(0).Databases(0)

    For I = 0 to MyDb.TableDefs.Count - 1
        If Left(MyDb.TableDefs(I).Name, 4) <> "msys" Then
            Debug.Print MyDb.TableDefs(I).Name
        End If
    Next I
End Function
```

Se puede agregar una tabla a la base de datos por medio de código. En este caso, el siguiente ejemplo agrega Table1 con un campo llamado CustomerName de tipo Texto.

```
Function CreateTbl( )
    Dim MyDb As Database
    Dim MyTbl As TableDef
    Dim MyFld As Field
    Dim I As Integer
    Set MyDb = DBEngine.Workspaces(0).Databases(0)
    Set MyTbls = MyDb.CreateTableDef("Table1")
    Set MyFld = MyTbl.CreateField("CustomerName", dbText)
    MyTbl.Fields.Append MyFld
    MyDb.TableDefs.Append MyTbl
End Function
```

Puede además eliminar una tabla específica de una base de datos por medio de código. En este caso, el ejemplo elimina Table1.

```
Function DeleteTable( )
    Dim MyDb As Database, I As Integer
    Set MyDb = DBEngine.Workspaces(0).Databases(0)
    For I = MyDb.TableDefs.Count - 1 to 0 step -1
        If MyDb.TableDefs(I).Name = "Table1" Then
            MyDb.TableDefs.Delete MyDb.TableDefs(I).Name
            Exit Function
        End If
    Next I
End Function
```

Colección Fields.

Contiene objetos Field. Una colección Fields contiene todos los objetos Field almacenados para un índice, consulta, conjunto de registros, relación o tabla.

Contenido en otros objetos. la colección **Fields** de los objetos Index, **QueryDef**, **Relation** y **TableDef** contiene las especificaciones para los campos cuyos objetos representa. La colección **Fields** de un objeto **Recordset** representa los objetos **Field** en un renglón de datos.

Posee Métodos y Propiedades. La colección **Fields** posee los siguientes métodos y propiedades:

- Propiedades – **Count**.
- Métodos – **Append**, **Delete**, **Refresh**.

El siguiente ejemplo imprime la cuenta de campos en todas las tablas:

```
Dim MyDb as Database
Dim I As Integer, J As Integer

Set MyDb = CurrentDb( )
For I = 0 To MyDb.TableDefs.Count - 1
     Debug.Print "Tabla: " & MyDb.TableDefs(I).Name
     Debug.Print MyDb.TableDefs(I).Fields.Count
Next I
```

Objeto Field.

Aquí se muestran las propiedades mas usadas para el objeto **Field**:

- **Attributes**
- **Name**
- **DataUpdatable**
- **OrdinalPosition**

Y se muestra una lista de los métodos del objeto **Field**.

- **AppendChunk**
- **CreateProperty**
- **FieldSize**
- **GerChunk**

El siguiente ejemplo imprime los nombres de todos los campos en todas las tablas:

```
Dim MyDb As Database
Dim I As integer, J as Integer

Set MyDb = CurrentDb()
For I = 0 To MyDb.TabeDefs.Count - 1
      For J = 0 To MyDb.TableDefs(I).Fields.Count - 1
            Debug.Print MyDb.TableDefs(I).Fields(J).Name
      Next J
Next I
```

Colección Indexes.

Una colección **Indexes** contiene los objetos **Index** almacenados, para cada índice definido en la tabla.

No existe un orden predefinido para almacenar los miembros dentro de la colección. Se pueden agregar, eliminar o modificar los miembros si la propiedad **Updatable** de **TableDef** es **True**.

La colección Indexes contiene las siguientes propiedades y métodos:

- Propiedades – **Count**.
- Métodos – **Append**, **Delete**, **Refresh**.

listando los índices en una tabla. El siguiente ejemplo imprime una lista de todos los índices en una tabla.

```
Dim MyDb As Database, MyTbl As TableDef
Dim I As Integer

Set MyDb = CurrentDb()
Set MyTbls = MyDb.TableDefs("Categories")
For I = 0 To MyTbls.Indexes.Count - 1
      Debug.Print MyTbls.Indexes(I).Name
Next I
```

Objeto Index.

Los objetos Index especifican el orden en que los registros son accesados desde las tablas de la base de datos, sean estos duplicados o no los objetos Index proveen además un acceso eficiente a los datos.

El siguiente ejemplo crea un Index en una tabla existente, en este caso, Table1:

```
Function CreateIndx()
      Dim MyDb As Database, MyTbls As TableDef
      Dim MyFld As Field, MyIndx As Index

      Set MyDb = DBEngine.Workspaces(0).Databases(0)
      Set MyTbl = MyDb.TableDefs("Table1")
      SetMyIndx = MyTbls.CreateIndex("PrimaryKey")

      MyIndx.Primary = True
```

```
    MyIndx.Required = True

    Set MyFld = MyIndx.CreateField("CustomerName", dbText)

    MyIndx.Fields.Append MyFld
    MyTbls.Indexes.Append MyIndx
End Function
```

2.7 Colección Properties y Objeto Property.

Colección Properties. Cada objeto DAO posee una colección de propiedades. Las propiedades pueden ser definidas por el usuario o definidas por DAO. Puede usar la colección **Properties** para enumerar tanto las propiedades incorporadas como las definidas por el usuario. Las propiedades DAO están siempre presentes como predeterminadas y no pueden ser modificadas.

Objeto Property. El objeto **Property** representa un aspecto incorporado o definido por el usuario del objeto de acceso a datos. La siguiente tabla lista las propiedades incorporadas del objeto **Property** y los objetos de acceso a datos que soportan propiedades definidas por el usuario

Propiedades del objeto Property	Objetos de acceso a datos que soportan las propiedades definidas por el usuario
Inherited	**Database**
Name	**Index**
Type	**QueryDef**
Value	**TableDef**
	Objetos **Field** en la colección **Fields** de objetos **QueryDef** y **TableDef**

2.8 Consultas, contenedores y relaciones.

Esta sección cubre las colecciones y objetos que generalmente se involucran en el trabajo con Consultas, Contenedores y Relaciones. Al igual que en la sección anterior, cada tópico se enfoca primera en las colecciones y después en los objetos.

Colección QueryDefs y objeto QueryDef.

Una colección **QueryDefs** contiene todos los objetos **QueryDef** en un objeto **Database**, mientras que un objeto **QueryDef** define una consulta en la base de datos. Este tópico cubre propiedades y métodos para las colecciones **QueryDefs** y el objeto **QueryDef**. Adicionalmente, se incluyen ejemplos de cómo crear una nueva consulta por medio del código.

Colección Containers y objeto Container

La colección **Containers** se refiere a la colección de todos los tipos de objetos guardados definidos en la base de datos, mientras que el objeto **Container** recopila información acerca de la base de datos y de cada tipo de objeto que contiene

Colección Relations y Objeto Relation.

La colección **Relations** contiene los objetos **Relation** almacenados de un objeto **Database**, mientras que el objeto **Relation** representa una relación entre los campos de las tablas o consultas. Este tópico cubre el uso del objeto **Relation** para crear nuevas relaciones y examinar las relaciones existentes en la base de datos.

Colección QueryDefs y objeto QueryDef.

Colección QueryDefs. La colección **QueryDefs** incluye las colecciones **Fields**, **Parameters** y **properties**. La colección **Parameters** es la colección **QueryDefs** predeterminada. La siguiente sección describe estas colecciones con mayor detalle.

La colección **Fields** se refiere a una colección de objetos **Field**, un objeto para cada campo en una consulta. El ordenamiento de miembros dentro de la colección es de acuerdo al valor de la propiedad **OrdinalPosition** de los objetos **Fields** individuales

La colección **Parameters** se refiere a la colección de objetos **Parameter**, un miembro para cada parámetro en la consulta. Esta es una colección de solo lectura. **Parameters** es la colección **QueryDefs** predeterminada. La colección **Parameters** contiene estas propiedades y métodos:

- Propiedades – **Count**
- Métodos – **Refresh**.

La colección **Properties** se refiere a la colección de propiedades del objeto **QueryDef**. Los objetos **QueryDef** soportan propiedades definidas por el usuario.

Objeto QueryDef. El objeto **QueryDef** corresponde a una definición almacenada de una consulta dentro de una base de datos. Conceptualmente, una consulta almacenada es una instrucción SQL compilada. Una instrucción SQL para una nueva consulta no esta compilada hasta que no se guarda y ejecuta por lo menos una vez. Es posible leer y modificar la instrucción SQL de un objeto **QueryDef**, establecer los parámetros de la consulta en el **QueryDef**, y ejecutar la consulta.

La siguiente tabla resume las propiedades del objeto **QueryDef**:

Propiedad	QueryDef	QueryDef nuevo	Descripción
Type	Solo lectura	Solo lectura	El tipo de la consulta (selección, actualización, inserción, etc.)
SQL	Lectura/Escritura	Lectura/Escritura	Especifica la instrucción SQL que define la consulta
Updatable	Solo lectura	TRUE	Indica cuando la definición de la consulta puede ser modificada

La siguiente tabla resume los métodos del objeto **QueryDef**.

Método	Descripción
Close	Cierra un objeto **QueryDef**
OpenRecordset	Ejecuta la consulta de selección definida por el valor actual de la propiedad SQL.
Execute	Ejecuta la consulta de acción definida por el valor actual de la propiedad SQL.

El siguiente código de ejemplo crea una nueva consulta dada una instrucción SQL y un nombre

```
Sub NewQry (Qname As String, SQLSTR As String)
Dim db As Database, MyQuery As QueryDef

Set db = DBEngine(0)
Set MyQuery = db.CrateQueryDef(Qname, SQLSTR)
MyQuery.Close
SoCmd OpenQuery Qname
End Sub
```

Colección Containers y objeto Container

La colección **Containers** se refiere a la colección de todos los tipos de objetos guardados definidos en la base de datos. El motor de base de datos Jet soporta varios tipos de contenedores, tales como relaciones, formas y reportes Microsoft Access define sus propios contenedores, los cuales residen en la ventana de base de datos (tablas, formas y otros). Los contenedores incluyen objetos **Document**. La colección **Containers** contiene las siguientes propiedades y métodos:

- Propiedades – **Count**.

- Métodos – Ninguno.

La colección **Containers** incluye las colecciones **Documents** y **Properties**. La siguiente sección describe estas colecciones con mayor detalle.

La colección **Documents** se refiere a la colección de todos los documentos lógicamente agrupados dentro de su contenedor. Por ejemplo, el contenedor **Forms** agrupa todos los objetos **Form**. La colección **Documents** posee las siguientes propiedades y métodos.

- Propiedades – **Count**.

- Métodos – Ninguno.

La colección **Properties** se refiere a la colección de propiedades asociadas con el objeto **Container**. Las propiedades definidas por el usuario no están disponibles en el objeto **Container**.

La siguiente tabla resume las propiedades del objeto **Container**

Propiedad	Container	Descripción
Name	Solo lectura	Indica el nombre del contenedor
Owner	Lectura/Escritura	Indica el nombre del contenedor propietario. El propietario del contenedor puede ser modificado mediante la modificación de esta propiedad.
Username	Lectura/Escritura	Representa un usuario o grupo de usuarios cuando se manipula los permisos de acceso del objeto **Container**.
Permissions	Lectura/Escritura	Establece los permisos para el usuario o grupo de usuarios identificados por la propiedad **UserName** del objeto **Container** o **Document**.
Inherit	Lectura/Escritura	**Inherit** es un valor lógico el cual, si se establece, significa que ningún objeto nuevo creado en el contenedor heredará los permisos que se establecen para el nuevo documento.

Nota: No hay métodos que apliquen al objeto **Container**.

Un objeto **Container** recopila información acerca de la base de datos o de cada tipo de objetos, todas las formas, macros, módulos, relaciones, reportes o tablas (incluyendo consultas) almacenadas.

Cada objeto **Database** tiene una colección simple de *contenedores*. Los contenedores trabajan conjuntamente con los objetos **Document** para enumerar todos los objetos almacenados en la base de datos, incluyendo aquellos objetos definidos por aplicaciones cliente. Por ejemplo, Microsoft Access define varios contenedores: *Formas*, *Reportes*, *Módulos*. El motor de base de datos Jet define además sus propios contenedores, tales como *Tablas* y *Relaciones*.

El uso principal de los objetos **Container** y **Document** es el enumerar todos los objetos definidos en por la aplicación y por el motor almacenados en la base de datos y establece permisis de usuario y posesion. No es posible crear o eliminar Contenedores o Documentos.

El motor de base de datos Jet define varios objetos **Container**. Microsoft Access define varios objetos **Container**. La siguiente tabla lista el nombre de cada objeto **Container**, su originador y una breve descripción de su contenido.

Container	Originador	Contiene información acerca
Databases	Motor de base de datos Jet	Base de datos contenida
Forms	Microsoft Access	Formas guardadas
Modules	Microsoft Access	Módulos guardados
Relationships	Motor de base de datos Jet	Relaciones guardadas
Reports	Microsoft Access	Reportes guardados
Scripts	Microsoft Access	Macros guardadas
Tables	Motor de base de datos Jet	Tablas y consultas guardadas

Microsoft Access define y usa ademas otro objeto **Container**, **SysRel**, para definir el formato de la ventana de Relaciones del Sistema. Normalmente usted no usa o lo modifica.

Colección Relations y Objeto Relation.

La colección **Relations** contiene objetos **Relation** almacenados, incluyendo colecciones **Fields** y **Properties**. La siguiente seccion describe estas colecciones con mayor detalle.

La colección **Fields** contiene todos los objetos Field almacenados de un objeto **Index**, **QueryDef**, **Recordset**, **Relation** o **TableDef**. Es además la colección predeterminada.

La colección **Fields** contiene estas propiedades y métodos:

- Propiedades – **Count**.
- Métodos – **Append**, **Delete** y **Refresh**.

La colección **Properties** se refiere a la colección de todos los objetos **Property** de una instancia especifica de un objeto. La colección **Properties** tiene las mismas propiedades y métodos que la colección **Fields**.

Cada objeto **Database** posee una colección simple de objetos **Relation** El objeto **Relation** es empleado para informar al motor de base de datos Jet acerca de las relaciones que existen entre los campos de dos objetos **TableDef**.

Es posible tener que hacer que el motor de base de datos Jet refuerce ciertas condiciones **Update** y **Delete** en los datos asociados con los campos de las relaciones (Integridad Referencial). Estas son llamadas relaciones reforzadas, debido a que el motor de base de datos Jet refuerza las reglas

Puede definirse además una relación que involucre **QueryDefs** (o ligas a **TableDefs** en otras bases de datos) que no estén reforzadas. Las relaciones no reforzadas son útiles cuando mapean el esquema global de la base de datos. Agregar o eliminar desde la colección de relaciones en un objeto **Database** crea o elimina relaciones.

La siguiente tabla resume las propiedades del objeto **Relation**.

Propiedad	Nueva Relación	Relación	Descripción
Name	Lectura/Escritura	Solo lectura	El nombre de la referencia. El nombre de la relación puede ser modificado mediante la escritura en esta propiedad
Table	Lectura/Escritura	Solo lectura	El nombre de la tabla primaria (referenciadora).
ForeignTable	Lectura/Escritura	Solo lectura	El nombre de la tabla foránea (referenciada)
Attributes	Lectura/Escritura	Solo lectura	Contiene información acerca del tipo de relación

Todas las propiedades son de solo lectura cuando están almacenadas en una relación. Para modificar un objeto **Relation**, debe eliminar el objeto mismo de la colección **Relations** de la base de datos y volver a crearlo.

CreateField – crea un objeto **Field** vacío.

2.9 Preguntas de repaso

1. ¿Qué propiedad contiene toda colección?

2. ¿Qué colección contiene todo objeto?

3. ¿Qué hacen las propiedades?

4. ¿Qué hace la siguiente línea de código?

```
Applcation.MenuBar = "mcrMainMenu" ' donde "mcrMainMenu es el nombre de una
macro
```

5. ¿Qué hace el siguiente código de ejemplo?

```
Application.SetOption "Muestra barra de estados ", FALSE
```

6 ¿Cuál es el valor de tblName en el siguiente código de ejemplo. donde tblName es una variable de caracteres (string) y MyTableDef es in objeto **TableDef**?

```
Set MyTableDef = DBEngine(0)(0).TableDefs!Company
TblName = MyTableDef.Name
```

7. ¿**CompactDatabase** y **RepairDatabase** con métodos de que objeto?

   a. **Recordset**.
   b. **Database**
   c. **DBEngine**
   d. **Workspace**

8. De un ejemplo de cómo referirse a la base de datos actual abierta en Microsoft Access.

9. ¿Cuál es la sintaxis propia para abrir otra base de datos, adicionalmente a la base de datos actual?

10. ¿Qué hace el siguiente código de ejemplo?

```
Sub WhatDoesThisDo()
  Dim tdf As TableDef

  For Each tdf In DBEngine(0)(0).TableDefs
      Debug.Print tdf.Name
  Next
End Sub
```

11. El siguiente código de ejemplo crea una tabla con dos campos de texto. ¿Qué importante paso se esta omitiendo?

```
Sub WhatDoesThisDo()
  Dim tdf As TableDef
  Dim fld As Field
  Dim db AS Database

  Set db = DBEngine(0)(0)
  Set tdf = db.CreateTableDef("NewTable")
  Set fld = tdf.CreateField("Name", dbText)
  tdf.Fields.Append fld
  Set fld = tdf.CreateField("Phone", dbText)
  tdf.Fields.Append fld

End Sub
```

12. Adicionalmente a la colección **Properties**, ¿qué otra colección contiene un objeto Index?

13. Si se busca ejecutar una consulta de actualización, ¿qué método emplearía?

14. ¿Con que propósito se emplean los **Containers** y **Documents**?

Capítulo 3: Depuración y manejo de errores.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Establecer puntos de rompimiento para detener la ejecución de un programa.
- ☑ Ejecutar porciones seleccionadas de código.
- ☑ Rastrear la secuencia de ejecución de un programa.
- ☑ Monitorear los valores de las variables.
- ☑ Probar los datos y procedimientos mediante la ventana de depuración.
- ☑ Agregar y editar expresiones de revisión.
- ☑ Atrapar errores en tiempo de ejecución.
- ☑ Usar el objeto Err en Microsoft Access.
- ☑ Usar el objeto Error de DAO y la colección Errores.

3.1 Tipos de errores.

Los errores de programación se pueden agrupar en 3 categorías generales: compilación, tiempo de ejecución y lógicos.

Errores de compilación.

Los *Errores de compilación* resultan de la incorrecta construcción del código. Por ejemplo, una palabra clave escrita incorrectamente, puntuación necesaria omitida, o una instrucción **Next** sin su correspondiente instrucción **For** pueden todas ellas ocasionar errores del lenguaje. Visual Basic para Aplicaciones (VBA) detecta estos errores cuando se abandona la línea o inmediatamente antes de ejecutar el código.

Errores en tiempo de ejecución (Run – Time).

Los *Errores en tiempo de ejecución* ocurren y son detectados por (VBA) cuando una instrucción intenta una operación que es imposible de llevar a cabo. Un buen ejemplo de estero es una referencia a un objeto que no existe en el contexto actual

Errores lógicos.

Los *Errores Lógicos* ocurren cuando el código no ejecuta en la forma que se espera. El código puede ser sintácticamente valido y correr sin desarrollar cualquier operación invalida y aun así producir resultados incorrectos Solo mediante la prueba del código y análisis de resultados se puede verificar que el código desarrolla lo que se pretende. Los errores lógicos también generan errores en tiempo de ejecución.

3.2 La ventana de depuración.

En la ventana de depuración., se pueden observar los valores de expresiones y variables mientras se ejecutan paso a paso las instrucciones en el código. Puede emplearse también la ventana de depuración para cambiar los valores de las variables y propiedades en modo de interrupción para ver como diferentes valores afectan al código.

Agregando Inspecciones (Watches)

Un aspecto importante de la depuración es ser capaz de checar los calores de las variables o expresiones dentro de un procedimiento para asegurarse que el código esta operando correctamente. En Microsoft Access, puede emplearse el inspector de expresiones (**Watch expression**) para checar los valores de las variables y expresiones. Un expresión de inspección es una expresión definida por el usuario que se emplea para monitorear el comportamiento de la variable o expresión en el código.

Ventana de depuración: mostrando valores y expresiones.

La ventana de depuración en Microsoft Access 97 reemplaza a la ventana Inmediato (**Immediate**) en versiones anteriores de Microsoft Access. Cuando el programa suspende la ejecución del código para propósitos de depuración, las expresiones de inspección seleccionadas aparecen en el panel Inspección (**Watch**) de la ventana de depuración, donde se pueden observar sus valores.

Rastreando el flujo del programa con caja de dialogo de llamadas.

Este topico analiza el uso de la ventana de dialogo de llamadas como una horma de rastrear la operación de un programa mientras ejecuta una secuencia de procedimientos. El cuadro de dialogo de llamadas es especialmente útil cuando se necesita rastrear una serie de procedimientos anidados.

Agregando Inspecciones (Watches)

Puede agregar expresiones de inspección antes de ejecutar los procedimientos mientras se encuentre en modo de rompimiento El modo de rompimiento (**Break mode**) es cuando se suspende la ejecución del código para realizar funciones de depuración

Agregando expresiones de inspección. Las expresiones de inspección permiten observar el valor de una variable, propiedad o cualquier otra expresión.

▶ **Para agregar expresiones de inspección:**

1 Del menú Depurar, elegir Agregar Inspección.

2. En la caja de expresión, escriba la expresion que se desea evaluar

3 Para establecer el alcance de la expresión analizada, seleccione el nombre de procedimiento o modulo apropiado dentro de Contexto

4. Para determinar como se desea que el depurador responda a la expresión de inspección, seleccione una opción en Tipo de Inspección.

5. Una vez terminado, presione Aceptar.

Agregando una expresión de inspección instantánea. El cuadro de dialogo de inspección instantánea muestra el valor de la expresión seleccionada.

▶ **Para agregar una expresión de inspección desde el cuadro de dialogo de Inspección Instantánea:**

1. En el panel de código de la ventana de depuracion, seleccione la expresión que se desea inspeccionar.
2. Del menú Depurar elija Inspección rápida. Puede también dar click en el icono asociado en la barra de herramientas de Visual Basic.
3. Elija Agregar.

Ventana de depuración: mostrando valores y expresiones.

La ventana de depuración en Microsoft Access 97 incluye dos paneles: el panel *Inmediato* que trabaja de manera muy similar como en versiones anteriores de Microsoft Access, y el panel de Inspección. El panel de inspección aparece solo si se han establecido expresiones de inspección.

El panel de Inspección muestra las expresiones de inspecciones actuales – expresiones cuyos valores se han decidido monitorear mientras se ejecuta el código.

En el panel de inspección, la columna Contexto indica el procedimiento, modulo o módulos en los cuales es evaluada la expresión de inspección. El panel de inspección puede mostrar un valor para la expresión de inspección solo si la instrucción actual esta en el contexto especificado De otra forma, la columna Valor mostrara un mensaje indicando que la instrucción no esta en contexto.

El panel de inspección puede contener 3 tipos diferentes de inspección, cada uno identificado por in icono diferente en la parte izquierda del panel.

Expresión de inspección

Suspensión cuando la expresión es valida

Suspendido ciando la expresión cambie

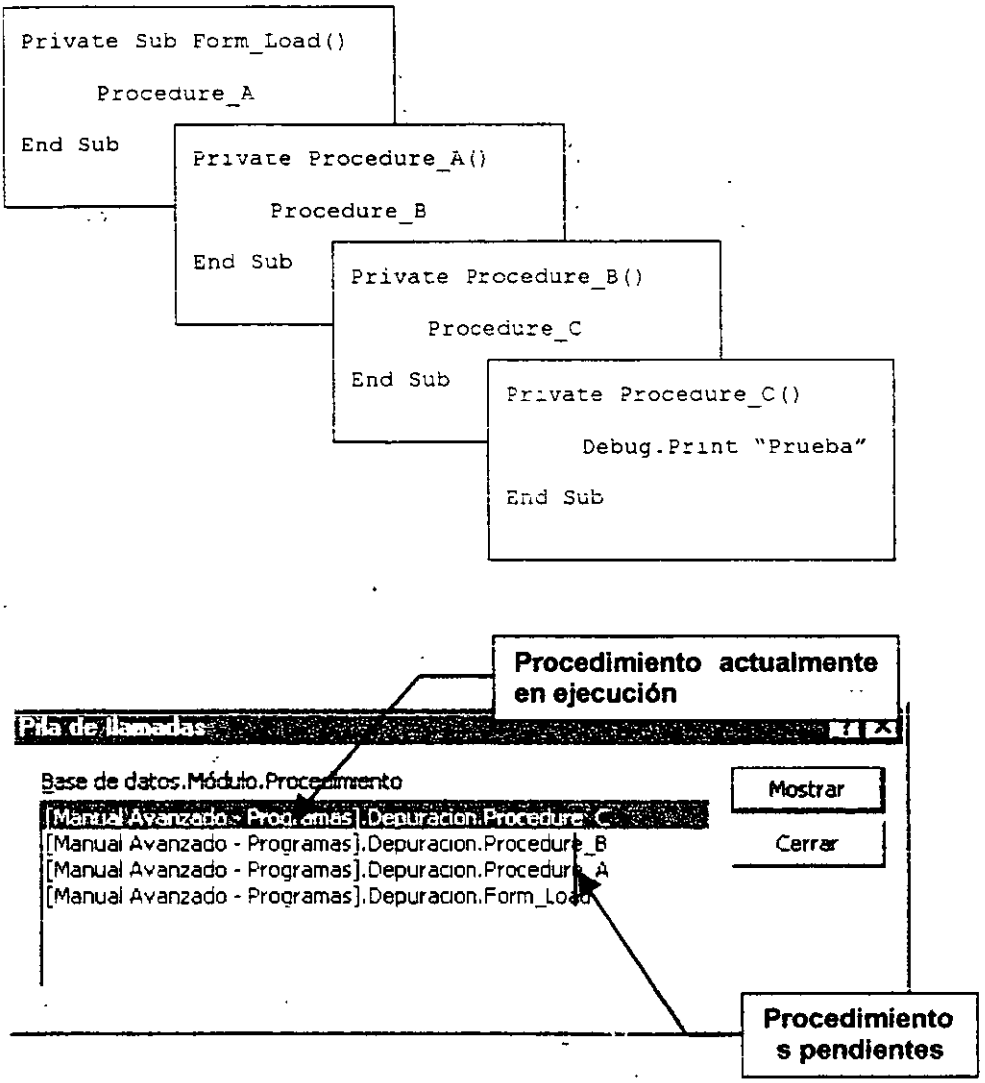
Una vez que se han agregado expresiones de inspección, puede llamar al procedimiento desde el panel Inmediato y el valor de la inspección será mostrada en el panel de inspección. Como se muestra en la imagen superior, si se inserta un punto de rompimiento en la función MyLoop y se agrega una inspección, puede entonces llamar a la función y examinar el valor de la expresión de inspección en el panel de Inspección.

Rastreando el flujo del programa con caja de dialogo de llamadas.

El cuadro de dialogo de Llamadas muestra una lista de todas las llamadas activas a procedimientos. Estas llamadas son los procedimientos en una aplicación que han iniciado pero no se han completado.

Supongamos que la siguiente secuencia de eventos ocurre de la siguiente manera:

1. Un procedimiento de evento llama al procedimiento A.
2. El procedimiento A llama al procedimiento B.
3. El procedimiento B llama al procedimiento C.



Mientras el procedimiento C se esta ejecutando, los otros procedimientos están pendientes, como se muestra en la lista de llamadas en el cuadro de dialogo de llamadas. Puede mostrar el cuadro de dialogo de llamadas solo cuando se esta en modo de suspensión (break mode)

3.3 Herramientas y técnicas de depuración.

VBA en Microsoft Access posee una variedad de herramientas de depuración integradas.

Herramientas de depuración en la barra de herramientas de Visual Basic.

Este tópico analiza los botones de la barra de herramientas de Visual Basic que son empleados para tareas de depuración comunes.

Uso del modo de suspensión (Break mode).

Para usar muchas de las herramientas de depuración de Microsoft Access, es necesario interrumpir la ejecución del código. En esta condición de suspensión, o *break mode*, el código aun se ejecuta pero espera entre las instrucciones. Esto permite revisar los valores de las variables y expresiones en la ventana de depuración. Si se encuentran problemas, pueden corregirse los errores en el código mientras se esta en este modo.

Uso de Debug.Print

Mientras se esta probando el código, pueden desplegarse los resultados de expresiones en el código mientras este se ejecuta. Se puede emplear la instrucción Debug.Print para mostrar los resultados en la ventana Inmediato. Esto es muy útil debido a que , a diferencia del modo de suspensión, no se requiere hacer una pausa en la ejecución para obtener información de cómo se esta desarrollando el código.

Herramientas de depuración en la barra de herramientas de Visual Basic.

Las herramientas de depuración mas comunes se encuentran localizadas en la barra de herramientas de Visual Basic según se describe en la siguiente tabla.

Herramienta	Función
Punto de interrupción	Crea o elimina un punto de interrupción Un punto de interrupción es una posición del código donde Visual Basic suspende la ejecución.
Inspección instantánea	Muestra el valor actual de una expresión y permite agregar la expresión al panel de inspección.
Paso a paso por instrucciones	Ejecuta solo la siguiente línea ejecutable de código. Si el código llama a otro procedimiento, la vista dentro del código se corre al procedimiento llamado hasta que termina.
Paso a paso por procedimientos	Ejecuta la siguiente línea de código ejecutable. Ejecutando procedimientos completos cuando son llamados sin analizarlos línea por línea.
Pila de llamadas	Muestra la lista de llamadas a procedimientos activas.
Ventana de depuración	Muestra la ventana de depuración para que pueda probarse y depurarse el código.

Restablecer	Termina la ejecución de los procedimientos de Visual Basic y limpia todas las variables publicas y privadas.
Continuar	Continua la ejecución del código después que ha sido suspendido
Terminar	Termina le ejecución de los procedimientos de Visual Basic y limpia todas las variables privadas, pero preserva las variables publicas.

Uso del modo de suspensión (Break mode).

El modo de suspensión o interrupción pausa la ejecución del código y da un cuadro de la condición en ese momento.

Accesible con la instrucción Stop, punto de interrupción o error en tiempo de ejecución.

Visual Basic entra en el modo de interrupción cuando cualquiera de las siguientes situaciones ocurran:

- La ejecución del código llega a una línea con un punto de interrupción.
- La ejecución llega a una instrucción Stop.
- Una expresión de interrupción definida en el cuadro de dialogo de agregar inspección cambia o se vuelve Verdadera, dependiendo de cómo fue definida.
- Una instrucción en una línea de código genera un error en tiempo de ejecución no atrapable.
- Se puede también interrumpir la ejecución desde el teclado presionando ESC o CTRL + BREAK.

Parar la ejecución. Cuando VBA encuentra una de las condiciones que lo llevan al modo de interrupción, la ejecución del código se para.

Muestra la ventana de depuración. La ventana de depuración aparece cuando se entra al modo de interrupción.

Mantiene las variables y propiedades establecidas. En el modo de interrupción, las variables y propiedades establecidas se mantienen, de tal forma que es posible:

- Inspeccionar los valores de variables, propiedades e instrucciones.
- Cambiar los valores de variables y propiedades.

Uso de Debug.Print

A veces es útil mostrar valores en el panel Inmediato para el código, mientras el código continua ejecutándose.

Uso del método Debug.Print en código para enviar la salida al panel Inmediato.

Dentro del código, puede usarse el método **Print** del objeto **Debug** para enviar la salida al panel Inmediato.

Por ejemplo, la siguiente instrucción imprime el valor de Salario en el panel Inmediato cada vez que la instrucción es ejecutada.

```
Debug.Print "Salario = " & Salario
```

Esta técnica trabaja mejor cuando existe un lugar en particular dentro del código en el cual la variable (en este caso Salario) se sabe que cambia. Por ejemplo. Se puede colocar la instrucción en un ciclo que repetidamente altera el Salario.

Sugerencia: un signo de interrogación (?) es la forma rápida para el método **Print**. El signo de interrogación significa precisamente lo mismo que **Print** y puede ser usado en cualquier contexto en el cual **Print** es usado.

3.4 Manejo de errores.

Manipular los errores comunes de manera elegante – esto es, sin necesidad de terminar el programa o mostrar un mensaje de error críptico – es parte del desarrollo de aplicaciones profesionales. Esta sección describe como manejar los errores en tiempo de ejecución dentro de la aplicación.

Atrapando errores de ejecución.

Este tópico define un error de ejecución y explica que sucede cuando uno de ellos ocurre. El tópico además provee una forma de crear rutinas de manipulación de errores racionales dentro del código.

Proceso de manipulación de errores.

Este tópico cubre el flujo general de la lógica empleada en muchas de las situaciones de manejo de errores.

Objeto Err (Microsoft Access).

Puede usarse el objeto **Err** para generar y limpiar errores de ejecución, y proveer a los usuarios con mensajes de error mas descriptivos.

Colección Errors (DAO).

Cuando un error se genera involucrando objetos de acceso a datos, el motor de base de datos Jet establece uno o mas objetos **Error** en la colección **Errors**.

Creación de un manejador de Errores.

Este tópico cubre los procedimientos generales que se aplican cuando se crean rutinas de manipulación de errores.

Otras formas de manejar errores.

A veces es deseable deshabilitar o restringir las capacidades de manipulación de errores en su aplicación, tal como durante las pruebas y prototipos. Este tópico cubre algunas alternativas para crear manejadores de errores.

¿Que rutina de manipulación de errores?

Este tópico cubre el orden de precedencia en el cual Visual Basic busca los manejadores de error en la aplicación.

Centralizando la manipulación de errores.

La manipulación de errores funciona mejor si el código contiene un procedimiento central que manipule todos los errores. Este tópico introduce lineamientos para la creación de una función central de manipulación de errores.

Atrapando errores de ejecución.

Las fuentes de errores de ejecución no siempre son obvias; a menudo es necesario probar para encontrarlos. Por ejemplo, aunque una aplicación pueda correr adecuadamente bajo circunstancias normales, puede no hacerlo quedando los usuarios accidentalmente introduzcan tipos de datos erróneos.

¿que pasa en los errores de ejecución? Un error de ejecución ocurre cuando VBA encuentra un comando que no puede llevar a cabo, tal como una división entre cero.

Cuando VBA encuentra un error de ejecución, la ejecución se detiene y un cuadro de dialogo se muestra con un mensaje describiendo el error.

En este punto uno puede:

- Detener la ejecución.
- Continuar la ejecución (si se resuelve el error en la ventana de Modulo).
- Ver la ventana de depuración, que resalta la línea que ha causado el error
- Ir directamente a la línea que causo el error
- Obtener ayuda sobre el error.

¿Por qué atrapar los errores de ejecución? Mediante la captura de los errores de ejecución se puede hacer la aplicación mas tolerante a error típicos que se puedan encontrar.

- Crear aplicaciones robustas – Las aplicaciones que atrapan errores de ejecución pueden manejar errores de usuario comunes sin detener la aplicación. Anticipar los errores comunes (tales como intentar abrir un archivo inexistente) y proteger contra ellos hacen que la aplicación sea menos susceptible de fallar.
- Permitir salidas elegantes – en esas ocasiones en que la rutina de manipulación de errores no puede resolver un error de ejecución, la rutina puede aun ejecutar acciones importantes, tales como cerrar cualquier archivo de datos abierto (y potencialmente guardar cantidades considerables de datos que puedan perderse de otra forma).

Proceso de manipulación de errores.

Muchos manejadores de error usan la misma lógica general.

Cuando VBA encuentra un error de ejecución, busca una instrucción On Error GoTo. Si se encuentra una, el error es manipulado y la ejecución se reanuda ya sea en la misma instrucción que ocasiono el error o en una diferente.

Si VBA no puede encontrar una instrucción On Error GoTo, la ejecución se detiene y Visual Basic muestra un mensaje de error de ejecución, el cual puede o no dejar al usuario confundido.

La siguiente tabla proporciona una breve descripción de las instrucciones y funciones estándar para manejar errores.

Instrucciones y Funciones	Descripción
On Error Go To	Habilita una rutina de manejo de errores y especifica la posición de la rutina dentro del procedimiento, puede ser usada también para deshabilitar una rutina de manejo de errores.
Resume	Reanuda la ejecución en la instrucción que ocasiono el error, después de que la rutina de manejo de error ha finalizado.
Resume Next	Reanuda la ejecución en la instrucción inmediatamente después de la que ocasiono error, después de que la rutina de manejo del error ha finalizado.
Resume línea	Reanuda la ejecución en un numero especifico de línea, después de que la rutina de manejo del error ha finalizado.
CVErr	Se emplean las funciones **CVErr** y **IsError** para crear errores definidos por el usuario.
IsError	Devuelve un valor lógico indicando si el valor de una expresión es un valor de error.

Objeto Err (Microsoft Access).

Provee información acerca de los errores de ejecución. Cuando se genera un error de ejecución, las propiedades del objeto Err contiene información que puede ayudar a identificar y manejar el error

Propiedades. Cuando se genera un error de ejecución, las propiedades del objeto **Err** se establecen de acuerdo a la forma en que se genero el error Estos errores se pueden originar a través de Visual Basic, un objeto OLE o el programador

La siguiente tabla lista y describe las propiedades del objeto **Err**.

Propiedad	Descripción
Number	Devuelve un numero valido de error. Es la propiedad predeterminada del objeto **Err**.
Description	El mensaje de error que corresponde a la propiedad **Number**.
Source	Una expresión de cadena nombrando el objeto o aplicación que originalmente genero el error El nombre del objeto actual de VBA que genero el error, proyecto o referencia (por ejemplo, la librería typelib de DAO).
HelpFile	La unidad, ruta y nombre del archivo de ayuda.
HelpContext	El identificador de contexto del archivo de ayuda VBA para el error correspondiente a la propiedad **Number**.

LastDLLError (solo en sistemas Microsoft Windows de 32 bits). Contiene el código de error de sistema para la ultima llamada a una librería de vínculos dinámicos. Esta propiedad es de solo lectura.

Métodos. La siguiente tabla lista y describe los métodos asociados al objeto **Err**.

Método	Descripción
Clear	Limpia todas las propiedades establecidas para el objeto **Err**. Limpiar es equivalente a hacer **Err**=0
Raise	Genera un error de ejecución basado en el objeto **Err** o en un error definido por el usuario. **Raise** puede ser usado (*La instrucción* **Error** *aun funciona por cuestiones de compatibilidad*)

Colección Errors (DAO).

Contiene todos los objetos Error almacenados. Cualquier operación usando objetos de acceso a datos puede genera uno o mas errores. Cuando se genera un error, el motor de base de datos Jet almacena objetos **Error** en la colección **Errors** del objeto **DBEngine**

La colección **Errors** contiene las siguientes propiedades y métodos:

- Propiedades – **Count**.
- Métodos – **Refresh**.

Propiedades del objeto Error. La siguiente tabla lista y describe las propiedades del objeto **Error**.

Propiedad	Descripción
Description	Devuelve una cadena descriptiva asociada con el error.
HelpContext	Devuelve un ID de contexto. una variable **Long**, para un tópico en un archivo de ayuda de Microsoft Windows.
HelpFile	Devuelve una ruta completamente cualificada para el archivo de ayuda.
Number	Devuelve un valor numérico especificando el error.
Source	Devuelve el nombre del objeto o aplicación que originalmente genero el error

Nota: el objeto **Error** no contiene métodos.

Creación de un manejador de Errores.

Hay tres pasos que se pueden aplicar a muchas de las rutinas de manipulación de errores:

- Establecer una trampa de errores.
- Escribir una rutina de manipulación de errores.
- Salir de la rutina.

Establecer una trampa de errores. Cada procedimiento **Sub** o **Function** que soporta trampas de errores debe incluir la instrucción **On Error** que indica a Visual Basic donde buscar las instrucciones de manejo de errores. Aunque **On Error** debe hacer referencia a una etiqueta o numero de línea dentro del mismo procedimiento, las instrucciones después de la etiqueta pueden llamar a otro procedimiento.

Escribir rutinas de manipulación de errores. Una rutina de manipulación de errores usualmente consiste de una instrucción **Select Case** (o alguna instrucción de toma de decisiones similar) que identifica diferentes números **Err** y como manipularlos.

Salir de la rutina. Use una de las instrucciones **Resume** para salir del manejador de error y continuar la ejecución del programa o, si el manejador del error no puede recuperarse del error, use **End** para salir de la aplicación.

Otras formas de manejar errores.

Introducción

Hay ocasiones en que se desea deshabilitar o limitar la manipulación de errores, tales como durante la creación del prototipo cuando se sabe que la manipulación del error no esta totalmente soportado.

Precaución Ambas técnicas eliminan todas las capacidades de manipulación de errores en el procedimiento; asegúrese de eliminarlas antes de liberar su aplicación

---

Deshabilitar la manipulación de errores dentro de un procedimiento Insertar la siguiente instrucción al inicio de un procedimiento deshabilita cualquier manejador de error en ese procedimiento.

```
On Error GoTo 0
```

Manipulación de error en línea Si un error se genera en cualquier línea del código, se puede instruir a Visual Basic que descarte esa línea y proceda con la siguiente línea mediante la inserción de la siguiente línea al inicio del modulo.

```
On Error Resume Next
```

Note que esta técnica puede ocasionar una que larga serie de errores sean ignorados. Por ejemplo, si un error se genera mientras se trata de abrir un archivo de datos, las lecturas subsecuentes y otras acciones que involucren al archivo pueden causar errores que son ignorados Sin alguna forma de notificación, los errores pueden no ser aparentes.

¿Que rutina de manipulación de errores?

Orden de precedencia Cuando Visual Basic encuentra un error de ejecución, busca un manejador de errores inactivo en este orden:

1. El procedimiento actual.
2. Los procedimientos listados en la lista de llamadas (iniciando con el procedimiento llamado mas recientemente).
3. El manejador de error dentro de Visual Basic, el cual detiene la ejecución del programa y muestra una caja de mensaje indicando el error de ejecución.

Centralizando la manipulación de errores.

Lineamientos para crear un manejador de errores centralizado Se puede centralizar la manipulación de errores de la aplicación mediante la creación de una función principal de manejo de errores que indica al procedimiento como procesar un error.

Deben tenerse en cuenta los siguientes lineamientos cuando se cree una función central de manejo de errores:

- Cada procedimiento debe tener su propia área del manejador de errores.
- Una función de manejo de errores simple centraliza el trabajo de manipulación de errores.
- Otros procedimientos llaman a la función principal de manejo de errores y toman acciones basadas en el valor de devolución.

Las instrucciones **Resume** dicen al procedimiento como recuperarse de un error. Sin embargo, debido a que las instrucciones **Resume** pueden aparecer solamente dentro del procedimiento que contiene la instrucción **On Error**, parte del código de manejo de errores debe permanecer dentro del procedimiento que requiere de las capacidades del manejo de errores

El siguiente ejemplo muestra como crear y usar una función de manejo de errores central.

```
Sub CodeWithErrorHandling()
    On Error GoTo ErrHnd
    '... codigo del procedimiento'
ErrHnd:
    iErrorType = Error_Handler(Err) 'Llama la funcion de manejo de error
    Select Case iErrorType
        Case 1
            ' Codigo para reanudar la misma linea que causo el error
            ...
            Resume
        Case 2
            ' Codigo para reanudar en linea siguiente a la del error
            ...
            Resume Next
        Case 3
```

```
            ' Codigo para reanudar en una linea especifica
            ...
            Resume Label ' Supone una etiqueta llamada "Label"
        Case 4
            ' Codigo para salir de la aplicacion elegantemente
            ...
            Resume
    End Select
End Sub

Function Error_Handler(iError_Num) As Integer
    Select Case iError_Num
        Case 11 ' Division entre 0
            ' Cualquier codigo necesario para resolver esta condicion
            ...
            iReturn = 1
        Case 53 ' Archivo no encontrado
            ' Cualquier codigo necesario para resolver esta condicion
            ...
            iReturn = 2
        Case 71 ' Disco no listo
            ' Cualquier codigo necesario para resolver esta condicion
            ...
            iReturn = 3
        Case Else ' Todos los demas errores
            ' Cualquier codigo necesario para resolver esta condicion
            ...
            iReturn = 4
    End Select
    Error_Handler = iReturn
End Function
```

3.5 Repaso.

1. ¿Cuál es la diferencia entre colección Errors y el objeto Er?

2. Cuándo la aplicación encuentra un error, ¿cuál es la forma recomendada para determinar que error se genero?

3. ¿Cómo probaría los datos y procedimientos en el panel Inmediato?

4. ¿Cuáles son las tres partes principales de los manejadores de error?

5. ¿Qué instrucción se usa para deshabilitar el manejo de errores en un procedimiento?

6 ¿Cuándo se usa la instrucción **Stop** en el código?

7. ¿cual de los siguientes argumentos se aplica cuando se entra en el modo interrupción?
   a. Puede usar el panel Inmediato para imprimir el valor de las variables.
   b. No puede abrir la ventana de depuracion.
   c. Puede agregar instrucciones de declaración para declarar nuevas variables y continuar ejecutando el código.
   d. No puede asignar nuevos valores a las variables.

8. En el código de ejemplo mostrado a continuación, cuando se genera un error en la línea 4, ¿cuál es la siguiente línea de código que será ejecutada?

```
1  On Error Resume Next
2  Dim liFile As Integer
3  liFile = FreeFile
4  Open "infile.txt" For Input As #liFile
5  Do Until EOF(liFile)
6      ' se procesa el archivo aqui
7  Loop
```

```
8  Exit Sub
9  ErrorHandler:
10     MsgBox "Ocurrio un error"
11     Resume Next
12 End Sub
```

Capítulo 4: Trabajando con conjunto de registros.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Identificara los diferentes tipos de conjuntos de registros.
- ☑ Diferenciara entre tablas, dynasets y snapshots.
- ☑ Manipulara datos empleando DAO.
- ☑ Creara conjuntos de registros.
- ☑ Manipulara conjunto de registros mediante la ejecución de las siguientes operaciones:
  - Ordenamiento.
  - Filtrado.
  - Moverse a través de los conjuntos de registros.
  - Encontrar conjunto de registros.
  - Agregar, editar y eliminar.
- ☑ Usara procesamiento de transacciones un conjunto de registros tipo dynaset.

4.1 Definiendo conjuntos de registros.

Esta sección introduce el concepto de usar objetos **Recordset** en el código para trabajar con datos.

Objetos Recordset dentro de la jerarquía DAO.

Este tópico muestra las posiciones relativas de la colección **Recordsets** y los objetos **Recordset** dentro de la jerarquía DAO. El tópico cubre además las propiedades y métodos para los objetos **Recordset**.

¿Qué es un objeto Recordset?

Este tópico define el objeto **Recordset**, el cual es esencialmente un conjunto de registros que ese comportan como un objeto, el cual puede ser manipulado por medio de código. Cuando se usa DAO, típicamente se trabaja extensivamente con conjuntos de registros, usándolos para manipular registros en las tablas de las bases de datos.

Tipos de Recordsets.

Este tópico identifica 3 tipos de objetos **Recordset** – **table**, **dynaset** y **snapshot** – y describe como difieren uno del otro, así como las situaciones en las cuales se usa cada tipo.

Objetos Recordset dentro de la jerarquía DAO.

La colección **Recordsets** contiene objetos **Recordset** y tiene las siguientes propiedades y métodos:

- Propiedad – **Count**.
- Método – **Refresh**.

DAO: Objetos Recordset. El objeto **Recordset** representa los registros en una tabla base o los registros resultado de ejecutar una consulta, el objeto **Recordset** contiene en si mismo la colección **Fields**.

Propiedades. El objeto **Recordset** contiene las siguientes propiedades:

Propiedad	Descripción
AbsolutePosition	Establece o devuelve el numero de registro relativo del registro actual de un objeto **Recordset**.
Sort	Establece o devuelve el sentido de ordenamiento para los registros de un objeto **Recordset**.

Métodos. El objeto **Recordset** contiene los siguientes métodos:

Propiedad	Descripción
GetRows	Recupera múltiples renglones de un **Recordset** y los vacía en un arreglo.
Requery	Actualiza los datos en un objeto **Recordset** mediante la reejecución de la consulta en la cual esta basada el objeto.

El siguiente ejemplo crea un **Recordset** llamado Table1 y lo ordena por el campo CategoryID.

```
Set db = CurrentDB()
Set rst = db.OpenRecordset("Table1", dbOpenDynaset)
Rst.Sort = "FirstName"
Set rstSorted = rst.OpenRecordset()
```

¿Qué es un objeto Recordset?

Se usa un objeto **Recordset**. en conjunto con sus propiedades y métodos para manipular registros en una base de datos.

Un conjunto de registros representado mediante código. El objeto **Recordset** representa los registros en una tabla base o los registros resultantes de la ejecución de una consulta. Un objeto **Recordset** puede referirse también a algunos o todos los campos o registros en otros objetos **Recordset**, o puede ser el resultado de una instrucción SQL.

Cuando se usa DAO, se trabaja con los datos casi completamente usando objetos **Recordset**. El motor de base de datos Jet construye todos los objetos **Recordset** usando registros (renglones) y campos (columnas).

La colección predeterminada de un objeto **Recordset** es la colección **Fields**, y la propiedad predeterminada del objeto **Field** es la propiedad **Value**. Puede simplificarse el código usando estos predeterminados. Por ejemplo, la siguientes línea de código establecen todas ellas el valor del campo **PubID** en el registro actual del objeto **Recordset**.

```
rstPublishers!PubId = 99
rstPublishers("PubID") = 99
rstPublishers.Fields("PubID").Value = 99
```

Tipos de Recordsets.

Se pueden usar tres tipos diferentes de objetos **Recordset** para manipular datos.

Table

Un objeto **Recordset** del tipo **Table** es una representación en código de una tabla base que puede usarse para agregar modificar o eliminar registros de una tabla simple de una base de datos.

Dynaset

Un objeto **Recordset** del tipo **Dynaset** es el resultado de una consulta que puede tener registros actualizables. Un objeto **Recordset** de este tipo es un conjunto dinámico de registros que pueden usarse para agregar, modificar o eliminar registros desde una tabla o tablas de la base de datos El objeto **Recordset** del este tipo puede contener campos de una o mas tablas en la base de datos.

Snapshot

Un objeto **Recordset** del tipo **Snapshot** es una copia estática de un conjunto de registros que puede ser usado para encontrar datos o generar reportes. Un objeto **Recordset** de este tipo puede contener campos de una o mas tablas en la base de datos, pero no puede ser actualizado.

Trabaja con **snapshots** es en ocasiones mas rápido que trabajar con **table** o **dynaset**, debido a que los **snapshots** contienen solo una copia fija del conjunto de registros como se encontraban estos al momento de crear el **snapshot**. Las operaciones de la base de datos, tales como ordenar e indexar requieren mucho mas procesamiento que la visualización del conjunto fijo de registros.

Nota: los objetos **Recordset** del tipo **Snapshot** contienen el dato completo (todos los campos), mientras que los objetos **Recordset** tipo **Dynaset** crean solo un conjunto de llaves. Así pues, un **Recordset** que contiene un gran número de renglones puede incrementar el nivel máximo de procesamiento de una maquina.

El tipo de Recordset empleado dependerá de lo que se desea hacer y de si se requiere modificar o simplemente visualizar los datos.

Si se desea	**Use este tipo de Recordset**
Ordenar datos	**Table**
Trabajar con índices	**Table**
Trabajar con tablas de Microsoft Access locales no adjuntas	**Table**
Actualizar registros basados en una consulta de selección	**Dynaset**
Trabajar con tablas adjuntas	**Dynaset**
Solo ver o buscar un conjunto de registros (esto trabaja con tablas locales y tablas vinculadas)	**Snapshot**
Crear reportes	**Snapshot**

4.2 Creación de Recordsets.

Esta sección describe diferentes formas de crear objetos **Recordset**

Método OpenRecordset

Este tópico cubre la sintaxis empleada cuando se emplea el método **OpenRecordset** para crear un objeto **Recordset**.

Ejemplos de diferentes Recordsets

Puede crear objetos **Recordset** desde tablas, consultas o formas. El tipo de **Recordset** creado dependerá en como se desean manipular los datos. Este tópico muestra dos ejemplos de diferentes formas para crear **Recordset** Un ejemplo usa una consulta almacenada, mientras que el otro emplea un **Recordset** existente sobre el cual se basará el nuevo **Recordset**.

Creación de Recordset usando SQL

Este tópico cubre la creación de un **Recordset** usando cadenas SQL. Situaciones donde puede requerirse el uso de cadenas SQL para crear **Recordset** incluyen la consulta "fuera de línea" de grandes bases de datos o la creación de consultas temporales.

Método OpenRecordset

Para usar el método **OpenRecordset**, primero debe crearse la variable de objeto Recordset.

Creación de una variable de objeto Recordset. Se usa el método **OpenRecordset** con el objeto Recordset para crear una variable de objeto Recordset. La aplicación puede contener tantas variables de objeto Recordset como sea necesario

El método **OpenRecordset** además trabaja con los siguientes objetos:

- **Database**.
- **TableDef**.
- **QueryDef**.

▶ **Para crear una variable de objeto Recordset.**

1 Declare una variable de tipo **Recordset**.

2. Establezca la variable al objeto devuelto por el método **OpenRecordset**

En el siguiente ejemplo se muestra la sintaxis del método **OpenRacordset** para objetos **Database**.

```
Dim MyDb As Database, rstOrder As Recordset

Set MyDb = CurrentDB()
Set rstOrder = MyDb.OpenRecordset ("Order")
```

La sintaxis del método **OpenRecordset** para todos los tipos de objetos es:

. **Set** *variable* = *objeto*.**OpenRecordset**([*tipo*[,*opciones*]])

El argumento *tipo* del método es una constante intrínseca que permite designar el tipo de objeto **Recordset**. Si no se especifica el tipo, el motor de base de datos Jet trata de crear uno en el siguiente orden de preferencia:

1. **Table**.
2. **Dynaset**.
3. **Snapshot**.

El método **Recordset** tiene los siguientes tipos disponibles de constantes:

- **dbOpenTable**
- **dbOpenDynaset**
- **dbOpenSnapahot**

El argumento *opciones* permite controlar el comportamiento del acceso multiusuario del **Recordset**. Puede estar vacío, o contener una de las siguientes constantes:

Argumento	Descripción
dbAppendOnly	Agrega solo nuevos registros (solo en el tipo **dynaset**)
dbForwardOnly	El **Recordset** es un **snapshot** de corrimiento hacia delante solamente. No puede clonar **Recordsets** creados con esta opción. Además, este **Recordset** soporta solo el método **MoveNext**
dbSQLPassTrough	Usado con paso a través de SQL. Causa que la instrucción SQL sea pasada a una base de datos ODBC para procesarse.
dbSeeChanges	Genera un error de ejecución si otro usuario esta modificando datos que se estén editando actualmente
dbDenyWrite	Otros usuarios no pueden modificar o agregar registros.
dbDenyRead	Otros usuarios no pueden ver registros (solo para **Recordset** tipo **table**)
dbREadOnly	Solo pueden visualizarse los registros, otros usuarios no pueden modificarlos
dbInconsistent	Permite solo actualizaciones consistentes (solo para tipos **dynaset**)
dbConsistent	Permite actualizaciones consistentes (solo tipos **dynaset**)

El motor de base de datos Jet almacena los objetos **Recordset** tipo dynaset y **snapshot** en memoria local. Si no hay suficiente espacio en la memoria local para almacenar los datos, el motor de la base de

datos guarda los datos adicionales en el espacio TEMP del disco. Si este espacio se agota, se genera un error atrapable.

Nota: si se usan variables para representar el objeto **Recordset** y el objeto **Database** que contiene al **Recordset**, asegúrese que las variables tengan el mismo alcance, o tiempo de vida por ejemplo, si se declara una variable publica para representar una base de datos conteniendo el objeto **Recordset**, debe declarar otra variable publica para representar la base de datos que contiene al **Recordset**. Alternativamente, declare la variable en el procedimiento **Sub** o **Function** usando la palabra clave **Static**.

Ejemplos de diferentes Recordsets

Los siguientes ejemplos muestran algunas formas diferentes de crear objetos Recordset.

Recordset basado en una consulta EL siguiente ejemplo crea un Recordset que devuelve renglones basados en una consulta almacenada.

```
Dim dbSales As Database
Dim rstSales As Recordset

Set dbSales  = CurrentDb()
Set rstSales = dbSales.OpenRecordset("qrySomeCust")
..
```

Recordset basado en un Recordset existente. En ocasiones es apropiado crear un Recordset basado en otro objeto Recordset existente. Por ejemplo, en una situación donde los usuarios tienen diferentes privilegios de acceso, puede necesitar que ciertos usuarios solo visualicen la información pero no puedan modificarla. En este caso, puede tomarse un dynaset existente y crear un **snapshot** para esos usuarios.

Debido a que ya ha sido especificada la fuente de datos, no es necesario especificarla nuevamente cuando se crea el Recordset basado en un objeto existente, como se muestra en el siguiente ejemplo:

```
Set MyDb = CurrentDb()
Set rst1 = MyDb.openrecordset("Customers", dbOpenDynaset)

Rst1.Sort = "Country"
Set rst2 = rst1.openRecordset(dbOpenSnapshot)
```

Puede usar la propiedad **Sort** con objetos Recordset tipo **dynaset** o **snapshot** El ordenamiento ocurre cuando un objeto Recordset subsecuente es creado. En el código de ejemplo previo, rst2 es el Recordset que esta ordenado La propiedad **Sort** se establece para rst1, y rst2 es el objeto **Recordset** creado desde rst1.

Sugerencia: a veces es mas rápido crear un nuevo Recordset basado en una instrucción SQL con la cláusula **Order** en lugar de ordenar uno existente usando la propiedad **Set** (por ejemplo, `SQL = "SELECT * FROM Employees ORDER BY City"`)

Creación de Recordset usando SQL

Puede usar las instrucciones de SQL (**Structured Query Language**) así como las consultas de selección para crear objetos Recordset Esto es útil cuando se necesita crear una consulta temporal. Además, si los criterios cambian basados en la selección de un Formulario, puede dinámicamente construir la instrucción SQL.

El siguiente ejemplo crea un Recordset basado en una instrucción SQL:

```
Dim MyDb As Database, rstVendors As Recordset
Dim SQLStr As String

Set MyDb = CurrentDB()
SQLStr = "SELECT * FROM Vendors ORDER BY VendorID;"
Set rstVendors = MyDb.OpenRecordset (SQLStr)
```

Sugerencia: si no se esta familiarizado con SQL, abra cualquier consulta de selección desde el menú Ver y elija SQL. Esto mostrara la ventana SQL la cual contiene todas las instrucciones SQL para la consulta.

4.3 Manipulación de Recordsets.

Una vez creado un Recordset, puede usarse en una variedad de formas para navegar y manipular los registros de la base de datos.

Ordenando y filtrando Recordsets

Este tópico abarca las operaciones de ordenamiento y filtrado de **Recordsets** usando Visual Basic. Puede solamente ordenar y filtrar operaciones sobre **Recordsets** del tipo dynaset o **snapshot**.

Determinación de los limites del Recordset

Cada Recordset soporta dos propiedades, **BOF** y **EOF**, las cuales indican cuando el registro actual esta actualmente al principio del Recordset (**BOF**) o al final (**EOF**). Determinar los limites del Recordset es importante debido a que se recibe un error de ejecución si se trata de mover mas allá del inicio o final de Recordset.

Moviendo a través de Recordsets

Una vez creado el **Recordset**, el motor de base de datos Jet provee una variedad de métodos para navegar a través de los renglones. Estos incluyen los métodos **MoveNext**, **MovePrevious**, **MoveFirst** y **MoveLast**

Encontrando registros.

La tarea de encontrar datos especificos en un Recordset se maneja de forma distinta dependiendo del tipo de Recordset. Los objetos Recordset de tipo **Table** pueden usar una búsqueda indexada para localizar datos, pero los objetos tipo dynaset o **snapshot** no pueden. Din embargo, el motor de base de datos Jet permite una gran facilidad en las búsquedas hechas en **dynasets** o **snapshots**. Los cuatro métodos diferentes de encontrar (**FindFirst**, **FindNExt**, **FindPrevious** y **FindLast**) permite optimizar la búsqueda par ver a través de un numero pequeño de registros para encontrar los datos necesitados.

Uso del método Seek

Si se ha creado un objeto Recordset del tipo **Table**, puede usar el método **Seek** para localizar renglones especificos. Este método **Seek** permite buscar a través de índices, lo cual es generalmente mas rápido, pero solo puede usarse en objetos **Recordset** de tipo **table**

Ordenando y filtrando Recordsets

Es necesario especificar un sentido de ordenamiento o establecer un filtro en el objeto **Recordset** para asegurar que los registros aparezcan en un orden especifico o que solo aparezca un subconjunto de registros deseado.

Ordenamiento. Las formas mas eficientes de ordenar y filtrar registros en un objeto **Recordset** son:

1 Abrir el **Recordset** desde un objeto **QueryDef** o una consulta ordenada.

– o –

2. Usar una nueva instrucción SQL como fuente del Recordset. Incluya las cláusulas SQL WHERE y ORDER BY en su consulta.

El siguiente ejemplo usa una instrucción SQL para obtener y ordenar registros:

```
Dim dbs As Database, rstHourly AS Recordset

Set dbs = CurrentDB()
Set rstHourly = dbs.OpenRecordset("SELECT FirstName, LastName FROM " _
"Employees WERE Title ='Manager' ORDER BY LastName")
```

Se usa la propiedad **Sort** para establecer el sentido de ordenamiento para registros en un **Recordset**. Esta propiedad solo aplica a objetos **Recordset** del tipo **dynaset** y **snapshot**. El sentido de ordenamiento predeterminado es ascendente.

Nota: establecer la propiedad **Sort** sola no ordena los registros en un **Recordset**. Es la subsecuente creación de otro Recordset la que causa el ordenamiento.

Filtrado. Filtrar un Recordset es similar a ordenar un **Recordset**. Filtrar involucra criterios específicos que limiten los registros que seran parte del **Recordset**.

El siguiente ejemplo muestra como usar la propiedad **filter** para obtener solo los registros para los clientes que viven en una zona geográfica en particular, en este caso, el Northeast.

```
MyRecordset.Filter = "[Regio] = 'Northeast' "
Set FilteredRecordset = MyRecordset.OpenRecordset
```

Determinación de los limites del Recordset

En un **Recordset**, leer después de la marca de fin de archivo (**EOF**) o antes de la de inicio de archivo (**BOF**) provoca un error de ejecución. Mediante el uso de las propiedades **BOX** y **EOF**, puede atrapar el error con rutinas de manejo de errores

Identificando el inicio o fin de un Recordset Para iterar a través de todos los registros en un **Recordset**, puede usar las propiedades **BOF** y **EOF** para revisar programáticamente el inicio y final del objeto **Recordset** Cuando crea un objeto Recordset, el registro actual esta posicionado en el primer registro si hay algún registro. Si no hay registros, la propiedad **RecordCount** se establece a 0 y las propiedades **BOF** y **EOF** se establecen a **TRUE**

El siguiente ejemplo lista todos los campos **CategoryId** en la tabla **Categories**.

```
Function RecPrint()
Dim recset As Recordset
Dim mydb As Database

Set mydb = CurrentDb()
Set recset = mydb.OpenRecordset("Categories")
Do While Not recset.EOF
      Debug.Print recset![CategoryID]
```

```
      Recset.MoveNext
Loop
Recset.Close
End Function
```

Moviéndose a través de Recordsets

Pueden emplearse los métodos **MoveNext**, **MovePrevious**, **MoveFirst** y **MoveLAst** para desplazarse a través de un **Recordset** y reposicionar el apuntador de registros.

```
Sub RecordsetMovement()
      Dim db As Database
      Dim rst As Recordset

      Set db = CurrentDb()
      Set rst = db.OpenRecordset("qrySomeCust")

      If rst.RecordCount > 0 Then
            Rst.MoveLast
            Do While Not rst.BOF
                  Debug.Print rst![Name]
                  Rst.MovePrevious
            Loop
      End If
      Rst.Close
End Sub
```

Propiedad RecordCount LA propiedad **RecordCount** de un **Recordset** no devuelve el núemro real de renglones en un **Recordset**. Devuelve el numemro de renglones accesados hasta el momento en el **Recordset**, si este no es del tipo **Table**.

▶ **Para encontrar el número real de renglones en un Recordset (que no es de tipo Table)**

- Use el método **MoveLast** antes de revisar el valor de la propiedad **RecordCount**.

  Si desea no moverse hasta el ultimo renglón, la propiedad **RecordCount** devuelve 0 si no hay renglones, o devuelve el numero de registros accesados hasta el momento. Por ejemplo, si hay 200 registros en un **Recordset**, y solo han sido accesados 5, la propiedad **RecordCount** devolverá 5. Así pues, para obtener el numero correcto de registros, use el método **MoveLast** antes revisar la propiedad **RecordCount**.

  Los **Recordsets** de tipo tabla mantienen la propiedad **RecordCount** sin necesidad de moverse hasta el ultimo renglón. Desde la creación de un **Recordset** tipo tabla, la propiedad **RecordCount** contiene el numero total de registros en esa tabla

Revisando si el Recordset contiene algún renglón Es posible crear un **Recordset** que no devuelva ningún renglón, lo cual puede afectar lo que se desea hacer con el **Recordset**. Use una expresión similar a la siguiente para revisar si el **Recordset** contiene algún renglón

```
If rst.RecordCount = 0
```

Encontrando registros.

Para objetos Recordsets de tipo dynaset y snapshot, use los métodos FindFirst, FindLast, FindNext y FindPrevious para localizar un registro especifico basado en cierto criterio. Si el motor de la base de datos do encuentra el registro, establece la propiedad NoMatch a TRUE. Para objetos Recordset tipo table, puede buscar registros con el método Seek.

Los métodos Find permiten optimizar la búsqueda de tal forma que el motor de la base de datos busque a través de un pequeño numero de renglones para encontrar los datos que necesita. Debido a que puede usar FindNext en una búsqueda, no necesita regresar el apuntador de registros par encontrar coincidencias subsecuentes. Esto permite ciclar a través de los registros, debido a que puede reiniciar la búsqueda sin tener que regresar al primer registro.

Se usa la misma sintaxis para los métodos Find:

Recordset.{FindFirst | FindPrevious | FindNext | FindLAst} *criterio*

Donde Recordset es una variable abierta de Recordset tipo dynaset o snapshot, y criteria es una cláusula WHERE formateada como en una expresión SQL, sin la palabra WHERE. Por ejemplo, el siguiente ejemplo busca el ultimo apellido que sea Smith:

Rts.FindFirst " [LastNAme] = 'Smith' "

En el siguiente ejemplo, el código abre un objeto **Recordset**, localiza cada registro cuyo campo Title satisface el criterio de búsqueda, y lo copia en el buffer de copia. El código entonces prepara el registro para una subsecuente edición, modifica el nombre del puesto y guarda el cambio mediante el método **Update**.

```
Dim dbsNeptuno as Database, rstEmployees As Recordset
Dim strCriteria As String, strNewTitle As String
' Establece el criterio de búsqueda
strCriteria = "Title = 'Sales Representative' "
' Establece el nombre del Puesto
strNewTitle = "Account Executive"

set dbsNeptuno = DBEngine.Workspaces(0).OpenDatabase("Neptuno.mdb")
' Crea Dynaset
Set rstEmployee = dbsNeptuno.OpenRecordset("Empleados", dbOpenDynaset)
rstEmployees.FindFirst strCriteria ' Encuentra la primera ocurrencia
 ' Repite nasta encontrar registros no soincidentes
Do Until rstEmployees.NoMatch
      With rstEmployees}
            .Edit ' Habilita la edicion
            !Title = strNewTitle ' Cambia el puesto
            .Update ' Guarda los cambios
            .FindNext strCriteria ' Encuentra la sitguiente ocurrencia
      End With
Loop ' Fin del ciclo
```

Empleando una consulta de actualizacion para cambiar los puestos puede ser mas eficiente. Por ejemplo, considere emplear el siguiente código para archivar los mismos resultados

```
StrSQL = "UPDATE Employees Set Title = 'Account Executive' " &
      "WHERE Title = 'Sales Representative' "
dbsNeptuno.Execute strSQL
```

Uso del método Seek

Empleado con Recordset tipo tabla para buscar registros especificos Si se ha creado un objeto **Recordset** tipo tabla, puede usar el rápido método **Seek** para localizar renglones específicos. Intentar usar el método **Seek** con cualquier **Recordset** diferente al tipo tabla ocasionara el error de ejecución (3219), "Operación invalida".

▶ **Para usar el método Seek para buscar datos**

1. Establezca la propiedad Index del **Recordset**.

   Esto indica a Microsoft Access a través de que índice se desea buscar. Si se desea usar la llave primaria para la búsqueda, debe de conocer el nombre de esta llave.

2. Proporcione el operador de búsqueda y uno o mas valores para los cuales buscar.

   El operador de búsqueda debe ser uno de los siguientes, indicando como desea que Microsoft Access busque·

   < <= = >= >

   Si el operador es =, >= o >, el motor de la base de datos busca desde el inicio del **_Recordset**. De otra forma, se inicia la búsqueda al final y se realiza hacia atrás. Debe indicar a Microsoft Access el criterio para el cual requiere la búsqueda. Esto se logra proporcionando uno o mas valores, correspondientes a sus llaves en el índice seleccionado.

3. Use la propiedad **NoMatch** del **Recordset** para revisar que realmente se ha encontrado un renglón.

El siguiente código de ejemplo emplea el método **Seek** par localizar la primera compañía cuyo nombre comienza con la letra B.

```
Rst.Index = "Company name"
Rst.Seek ">=", "B"

If rst.NoMatch Then
MsgBox "No es posible encontrar una coincidencia"
Else
MsgBox "El nombre de la compañía es " & rst![Company name]
End If
```

4.4 Actualización de Recordsets.

Las aplicaciones de base de datos necesitan ser capaces de agregar, actualizar y eliminar datos. El motor de la base de datos Jet proporciona métodos para llevar a cabo cada una de estas tareas. Los siguientes tópicos cubren varios métodos de manipulación de datos que soporta Microsoft Access

Adición de registros.

Este tópico cubre el proceso de agregar registros a un objeto **Recordset** tipo **dynaset** o tabla. No puede agregar, editar o eliminar registros en un objeto **Recordset** tipo **snapshot**.

Edición de registros.

El proceso para modificar registros es similar al de agregarlos a un objeto **Recordset** tipo tabla o **dynaset**. Se desplaza hacia el renglón deseado, o se agrega, se editan los datos o agregan valores a los campos del registro, y se usa el método **Update** para guardar los cambios.

Eliminación de registros.

Eliminar registros es una operación permanente, sin advertencias o comprobaciones – a menos que se haga dentro del contexto de procesamiento de transacciones. Es este caso, puede deshacer la transacción para recuperar el renglón eliminado

Adicion de registros.

Ser capaz de agregar registros a una tabla a través del código ofrece gran flexibilidad en el desarrollo de aplicaciones.

Proceso Agregar registros a un **Recordset** incluye los siguientes pasos:

1. Use el método **AddNew** para agregar un registro Todos los campos se establecen a **Null**
2. Asigne valores a los campos
3. Use el método **Update** para guardar el nuevo registro

   Si no se llama al método **Update** antes de dejar el registro actual, el motor de la base de datos deshecha cualquier cambio que haya sido hecho y no agregara ningun registro.

Cuando se emplee el método **AddNew**, el registro actual es aquel que era el actual antes de agregar el nuevo registro. Si se desea que el nuevo registro sea el registro actual, use el método **Move**, especificando como argumento la marca devuelta por la propiedad **LastMidified** del **Recordset**

El siguiente ejemplo crea un nuevo registro en la tabla Employee, introduce valores y solicita al usuario guardar los cambios Si el usuario elige no guardar los cambios, el método **Update** es cancelado.

```
Sub NewRecord()
```

```
    Dim dbs AS Database, rst AS Recordset

    ' Devuelve variables de base de datos apuntando a la base de datos
    acutual
    Set dbs = CurrentDb()
    Set rst = dbs.OpenRecordset("Employees")
    With rst
        'Agrega un nuevo registro al final del objeto Recordset
        .AddNew
        ![LastName] = "Cristopher"
        ![FirstName] = "Kevin"
    End with
    ' Solicita al usuario guardar los cambios
    If MsgBox("¿Desea guardar los cambios?", vbYesNo) = vbNo Then
        ' Si el usuario elige No, se cancelan los cambios
        rst.CancelUpdate
    Else
        ' Si el usuario elije Si, se almacenan los cambios
        rst.Update
    End If
    Dbs.Close
End Sub
```

Los objetos **Recordset** tipo **dynaset** tratan los registros nuevos de manera diferente a como los trata un tipo table. La siguiente tabla resume las diferencias entre estos dos tipos de **Recordset**

Recordset tipo tabla	Recordset tipo **dynaset**
El motor de la base de datos coloca el nuevo registro en su posición correcta en el índice (suponiendo que se ha establecido la propiedad Index)	El motor de la base de datos siempre coloca el nuevo registro al final del **Recordset**
Los usuario no pueden ver los nuevos registros hasta que hayan actualizado los registros	El motor de la base de datos agrega el nuevo registro en la tabla asociada.

Edición de registros.

Proceso para modificar registros Siga los siguientes pasos para modificar **Recordsets**; de otra forma, puede encontrar resultados no deseados. El paso mas importante de este proceso es probablemente el 4. Si se hacen cambios a un registro, pero no se usa el método **Update** para confirmar los cambios, el motor de base de datos trata al registro como si nunca se hubieran realizado estos.

▶ **Para cambiar los datos en un** Recordset **actualizable**

1. Coloquese en el registro que se desea modificar.
2. Use el método **Edit** para colocar el registro actual en modo Edición.
3. Realice los cambios necesarios.
4. Use el método **Update** para guardar los cambios.

El siguiente ejemplo reemplaza todas las ocurrencias de "London" con "Rome" en la columna City de la tabla Customers.

```
Function SeekChange()
      Dim db As Database
      Dim rst As Recordset

      Set db = CurrentDb()
      Set rst = db.OpenRecordset("Customers")
      rst.Index = "City"
      rst.Seek "=", "London"
      Do While Not rst.NoMatch
            rst.Edit
            rst!City = "Rome"
            rst.Update
      Loop
      rst.MoveFirst

      Do While Not rst.EOF
            Debug.Print rst![Contact Name], rst!City
            Rst.MoveNext
      Loop
End Function
```

Eliminación de registros.

Proceso Puede eliminar un registro existente en un objeto **Recordset** tipo tabla o **dynaset** usando el método **Delete**. No puede eliminar registros en un objeto **Recordset** tipo **snapshot**.

Para eliminar registros de un **Recordset**

1. Colóquese en el registro deseado

2. Use el método **Delete** para eliminarlo

   A diferencia e otros métodos de modificación de registros, no es necesario usar el método Update cuando se elimina un registro. Una vez que se ha eliminado el registro, la eliminación es permanente, a menos que se incluya la operación de eliminación dentro de una transacción. En este caso, puede deshacer la transacción y recuperar el registro eliminado.

Sugerencia Despues de eliminar un registro, este es aun el registro actual. El registro previo es aun el registro previo, y el siguiente igual Use el método **Move Next** para desplazarse al siguiente registro, si esa es la posición que se desea sea el nuevo registro activo.

El siguiente código de ejemplo elimina todos los registros de los empleados por horas en la tabla Employees.

```
Function DeleteHourlyEmployees()
      Dim dbs As Database, rst As Recordset
      Set dbs = CurrentDb()
      Set rst = dbs,OpenRecordset("Employees")
      Rst.MoveFirst
      Do Until rst.EOF
            If rst!Title = "Hourly" Then
                  rst.Delete
            End If
            rst.MoveNext
      Loop

      rst.Close
      dbs.Close
End Function
```

4.5 Procesamiento de transacciones.

Las transacciones son una manera efectiva de mejorar el desempeño de las actualizaciones de un **Recordset**. Habilitan al motor de la base de datos acumular múltiples transacciones y escribirlas como un simple **batch**.

Refuerza la integridad de los datos

A veces, para reforzar la integridad de los datos, debe considerar un conjunto de operaciones como una unidad simple. Por ejemplo, la transferencia de fondos de una cuenta de banco a otra consiste de dos operaciones: introducir el débito en una cuenta y el crédito coincidente en la otra. En la practica, ambas operaciones deben ser exitosas o la aplicación no procesará ninguna operación.

Permite actualizaciones, adiciones o eliminaciones rápidas por lotes

Cualquier operación del **Recordset** que modifique los datos puede beneficiarse grandemente utilizando transacciones. Se usan las transacciones primeramente para permitir del deshacer y el confirmar los cambios masivos de datos, pero debido a que el buffer de transacciones lee y escribe, puede usarlos para agilizar las operaciones de edición, actualización y eliminación.

Nota Las transacciones necesitan ser parte del mismo procedimiento. No puede dividirlas.

DAO soporta tres métodos de transacciones

Inicie una transacción después de abrir un **Recordset** pero antes de realizar cualquier cambio. Confirme la transacción después de haber realizado el ultimo cambio antes de cerrar el **Recordset**. Los tres métodos que soporta DAO son todos métodos del objeto **Workspace**. Estos son:

- **BeginTrans**: inicia una nueva trancasen.
- **CommitTran**: confirma todos los cambios hechos a los datos desde el método **BeginTRans** mas reciente.
- **Rollback**. deshace. o cancela todos los cambios realizados a los datos desde el método **BeginTRans** mas reciente

El siguiente código de ejemplo usa transacciones para agilizar la eliminación de renglones. Comienza la transacción antes de que el primer renglón cambia y la confirma después del cambio final.

```
Sub DeleteTable (strTable As String, fUseTransactions As Integer)
    ' Elimina todos los registros de una tabla especifica, con o sin transacciones.
    Dim rst As Recordset, db As Database, wrk As Workspace

    Set wrk = DBEngine.Workspaces(0)
    Set db = wrk.Databses(0)
    Set rst = db.OpenRecordset(strTable)

    ' Si no se especifica una transaccion, entonces se inicia la transaccion
    ahora
    If fUseTransactions Then wrkBeginTrans
    rst.MoveFirst
    Do While Not rst.EOF
        rst.Delete
```

```
            rst.MoveNext
        Loop
        If fUseTransactions Then wrk.CommitTrans
        rst.Close
End Sub
```

4.6 Preguntas de repaso

1. ¿Qué colección contiene el objeto **Recordset**?

2. ¿Cuándo se usa DAO, se trabaja con datos (registros en una base de datos) casi completamente usando que objeto?

3. ¿Cuál de los siguientes tipos de **Recordsets** es una copia estática de un conjunto de registros y no puede ser actualizada?

   - **Table**
   - **Dynaset**
   - **Snapshot**
   - **Dynaset** inconsistente

4. ¿Qué método se emplea para crear un **Recordset**?

5. ¿Qué método de un **Recordset** se emplea para navegar al siguiente registro?

6. ¿Qué método de un **Recordset** usaría para encontrar un registro en un **Recordset** tipo tabla?

7. Cuando no hay registros en un **Recordset**. (la propiedad **RecordCount** es igual a 0), ¿cuáles son los valores de **BOF** y **EOF**?

8. ¿Qué hace la siguiente instrucción?

```
rst.FindFirst "[LastName] Like 'B' "
```

9. ¿Qué propiedad de un **Recordset** necesita esta establecida antes de que se pueda implementar el método **Seek**?

10. ¿Qué método de un **Recordset** se usa para crear un nuevo registro?

11. ¿Qué método es usado para editar un registro en un **Recordset** tipo **snapshot**?

12. ¿**BeginTRans**, **CommitTrans** y **RollBack** son todos métodos de que objeto?

13. ¿Qué método es empleado para finalizar una transacción y cancelar todas las ediciones, inserciones y eliminaciones?

Capítulo 5: Módulos de clase.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Definir y crear módulos de clase en Microsoft Access.
- ☑ Usar Property Set, Property Get y Property Let para implementar propiedades de forma.
- ☑ Indentificar el alcance de un modulo de clase.
- ☑ Abrir y referenciar múltiples instancias de una forma.
- ☑ Usar el examinador de objetos para ver procedimientos, métodos y propiedades.
- ☑ Usar el cuadro de diálogo Referencias para agregar o eliminar referencias a librerías.

5.1 ¿Qué es una clase?.

El concepto de clase es importante para trabajar con módulos e clase. En Microsoft Access, los objetos Form y Report son los únicos objetos que permiten la creación de módulos de clase.

Definición de una clase.

Una clase es la definición formal de un objeto. La clase actúa como una plantilla desde la cual se crea una instancia de objeto durante la ejecución. La clase define las propiedades del objeto y los métodos usados para controlar el comportamiento del mismo

Usando clases para crear objetos.

Un objeto es una instancia de una clase, la cual tiene métodos y propiedades. Por ejemplo, considere el diagrama esquemático (diagrama de circuito) de un teléfono como una clase y el teléfono en si mismo como un objeto.

5.2 Módulos de clase en Microsoft Access.

Este tópico introduce los módulos de clase y diferentes aspectos relacionados al trabajo con ellos.

¿Que es un modulo de clase?.

Cada modulo de clase actúa como un negativo de un objeto. En otras palabras, cada modulo de clase define un tipo de objeto. Pueden tenerse varios módulos de clase en una aplicación. Durante la ejecución, se crean objetos mediante la creación de una instancia de determinada clase.

Nombres de clase.

Microsoft Access emplea identificadores predeclarados, tales como Form_ o Report_, para permitir hacer referencia directa al objeto Form o Report.

Alcance de los módulos de clase.

Ciertas reglas y lineamientos gobiernan el alcance de los módulos de clase. Este tópico trata con los lineamientos de alcance para los módulos de clase y cubre variables, procedimientos de evento, las palabras Private versus Public y las llamadas a procedimientos

Creación de procedimientos Property.

Microsoft Access proporciona los procedimientos Property que habilitan la creación de propiedades personalizadas durante la ejecución. Existen tres nuevos procedimientos Property en Microsoft Access: Property Get, Property Set y Property Let. Este tópico examina y da ejemplos de cada uno.

Creación de métodos.

Este tópico trata de como crear métodos personalizados para los módulos de clase haciendo globalmente disponibles los procedimientos.

¿Que es un modulo de clase?.

definición de módulos de clase Un modulo de clase es un modulo de forma o reporte que contiene cierta definición para nuevos objetos personales. Dentro del modulo de clase, los procedimientos creados por el usuario pueden convertirse en métodos y propiedades para el objeto personal. Puede entonces, crear una nueva instancia de este objeto y manipularlo por medio de las propiedades y métodos definidos en los procedimientos dentro del modulo de clase.

Pueden visualizarse todos los módulos de clase con el Examinador de Objetos. En Microsoft Access, las únicas clases que pueden crearse son formularios y reportes.

Nombres de clase.

Identificando objetos Form y Report. El nombre de clase es el nombre usado para referirse a un modulo de clase, el cual es un modulo de formulario o reporte. Para distinguir entre los módulos de formulario y reporte, el nombre de la clase es prefijado con el tipo de modulo. Por ejemplo, el nombre de

clase para un modulo asociado con un formulario llamado OrderForm puede ser Form_OrderForm, mientras que uno asociado con un reporte llamado OrderReprot podria llamarse Report_OrderReport.

El siguiente ejemplo muestra el uso de Form_ como un identificador predeclarado para un nombre de formulario:

```
Function PredeclaredIdentifiers()
      Dim frm As Form

      Set frm = Form_Categories
      Frm.Caption = "Es un nuevo titulo"
End Function
```

creación de instancias múltiples de un formulario usando arreglos. Puede declarar un arreglo de formularios con las palabras clave Private, Dim, ReDim, Static o Public de la misma forma en que se declara un arreglo de cualquier otro tipo. Cuando se declara un arreglo con la palabra clave New, VBA automáticamente crea una nueva instancia del formulario para cada elemento en el arreglo

El siguiente ejemplo crea cinco instancias de un formulario y establece cada titulo con el nombre del formulario y el numero de instancia.

```
' En la sección de declaraciones
Dim MyFormArray() As Form_Form1
' puede ser  también Dim MyFormArray() As Form

Function CreateFiveForms()
      Dim x As Integer
      ReDim MyFormArray(5) As New Form_Form1

      For x = 0 To 4
            MyFormArray(x).caption = "Form1:" & x
            MyFormArray(x).Visible = True
      Next
End Sub
```

Dentro de la colección Forms, no se puede hacer referencia a una instancia no predeterminada de una forma mediante nombre Solo puede referirse a estas instancias mediante sus respectivos números de índice. Cuando se crean múltiples instancias no predeterminadas de un formulario, cada instancia puede tener el mismo nombre. Así pues, es posible tener mas de un formulario con el mismo nombre en la colección Forms, sin otra forma de distinguirlos entre ellos mas que con el numero de índice.

El siguiente ejemplo muestra como cerrar una instancia particular de un formulario desde el conjunto de instancias múltiples de ese formulario. Se emplea el método DoCmd.Close para referirse a una instancia simple del formulario

```
DoCmd.Close acForm, Forms(1).Name
```

Nota: este ejemplo solo trabaja cuando se trata con una clase de formularios.

Alcance de los módulos de clase.

El *alcance* define la visibilidad de un procedimiento, variable, objeto o modulo de clase.

Esta lista describe las reglas y lineamientos que gobiernan el alcance de los módulos de clase.

- Los procedimientos creados en un modulo de clase son Public de manera predeterminada.

Los procedimientos Public están disponibles para otros módulos de clase y módulos estándar. Esto permite tener acceso a ellos desde el exterior de un formulario o reporte particular. Puede emplearse la palabra clave Public, pero no es necesario.

Las variables creadas en la sección de declaraciones son Publicas de manera predeterminada. Puede usar la palabra clave Public, pero no es necesario.

- Cuando Microsoft Access inserta un procedimiento de evento, agrega la palabra reservada Private de manera predeterminada.

Todos los procedimientos de evento que se declaren como Public son métodos de la clase (formulario o reporte).

- cuando se declaran procedimientos en los módulos de clase como Public, se convierten en métodos de ese formulario.

El procedimiento puede ser invocado desde cualquier sitio en la aplicación especificando el nombre del formulario y el nombre del procedimiento.

- solo los procedimientos adjuntos a un modulo de clase de un formulario particular puede llamar a procedimientos Private.

Para declarar funciones como Private, es necesario especificar la palabra clave Private.

- cuando se declaran variables como Public dentro de un modulo de clase, se convierten en propiedades de la clase (formulario o reporte).
- Existen tres tipos de procedimientos Property:
  - Property Get: devuelve el valor de una propiedad.
  - Property Set. establece una referencia a un objeto.
  - Property Let: asigna un valor a la propiedad.

Nota una variable o procedimiento a nivel de clase no puede tener el mismo nombre de una propiedad del formulario (por ejemplo, BackColor) o un control del mismo. Si se define tal miembro, el programa genera un error de compilación Esto se aplica tanto a miembros Public como Private.

Los procedimientos de otros formularios, reportes y módulos estándar pueden llamar y hacer referencias a procedimientos y variables publicos de una forma o reporte

Los procedimientos y variables Public son considerados para ser métodos y propiedades del modulo de clase en el cual han sido declarados.

Sugerencia use el Examinador de Objetos para ver los metodos de un formulario (funciones publicas).

Creación de procedimientos Property.

Los procedimientos Property permiten crear propiedades de ejecución personalizadas que pueden devolver valores y desarrollar acciones cuando se establecen esos valores. Esto proporciona una considerable flexibilidad en la personalización de los módulos de clase y aplicaciones.

Existen tres nuevos procedimientos Property en Microsoft Access.

Property Get

Se puede usar la instrucción Property Get para definir un procedimiento Property que devuelve el valor de la propiedad para un objeto definido por el usuario.

Property Let

Se puede usar la instrucción Property Let para definir un procedimiento Property que asigna un valor de una propiedad para una instrucción definida por el usuario. Usualmente se pueden usar las instrucciones Property Let y Property Get juntas para crear una propiedad para un objeto.

Property Set

Se puede usar la instrucción Property Set para crear una propiedad personalizada de ejecución que establece una referencia a un objeto.

Cada instrucción Property Set debe definir al menos un argumento para el procedimiento que define. Ese argumento (o el ultimo argumento s hay mas de uno) contiene la referencia actual del objeto para la propiedad cuando el programa invoca al procedimiento definido por la instrucción Property Set.

Debe declarar el ultimo argumento en un procedimiento Property Set con un objeto para que trabaje adecuadamente.

▶ **Para crear una propiedad personalizada de ejecución que devuelve un valor.**

- Crear dos procedimientos con el mismo nombre.

a. Un procedimiento usa Property Ser para establecer la propiedad.

b. El otro procedimiento usa Property Get para devolver el valor.

En el siguiente ejemplo puede crear una propiedad que devuelve el color de un ato y entonces cambia el color del mismo.

```
Public Property Get CarColor As string

        [instrucciones del código]

End Property

Public Property Let CarColor z As String

        [instrucciones del código]

End Property
```

Cuando se usa el procedimiento Property Get con el procedimiento Property Let, el nombre y tipo de datos de cada argumento en el procedimiento Property Get debe coincidir con sus respectivos en el procedimiento Property Let Sin embargo, el procedimiento Property Let debe tener argumentos adicionales los cuales son pasados como el valor de la propiedad. El ejemplo siguiente muestra como funciona.

Public Property Get CustomProp() As String

Public Property Let CustomProp(x As String, y As Integer, z As String)

La variable x es el valor que realmente se asigna a la propiedad personalizada en ejecución. Puede usar las variables x e y como argumentos adicionales.

Creación de métodos.

Así como se crean propiedades personales para los módulos de clase, pueden crearse además métodos para esos módulos. Esto se logra creando procedimientos públicos Sub en los módulos de clase.

El ejemplo siguiente muestra un método personal para un modulo de clase. El código modifica la propiedad Visible a True y establece la propiedad Caption de la forma a un valor String.

```
Public Sub Change Frm(strCaption As String)
      Me.Visible = True
      Me.Caption = strCaption
End Sub
```

5.3 Herramientas para trabajar con objetos.

El ambiente de desarrollo de VBA incluye dos características visuales el Examinador de Objetos y el cuadro de dialogo de Referencias, que son útiles cuando se trabaja con objetos.

Cuadro de dialogo de Referencias

El cuadro de dialogo de Referencias contiene una lista de las librerías de objetos (*.DLL, *.OLB, *.TLB) y librerías de bases de datos (*.MDA) disponibles para la base de datos actual.

Examinador de objetos.

El Examinador de objetos permite examinar todos los procedimientos, métodos y propiedades disponibles para la base de datos actual.

Cuadro de dialogo de Referencias

El cuadro de dialogo de Referencias contiene una lista de las librerías de objetos (*.DLL, * OLB, *.TLB) y librerías de bases de datos (*.MDA) disponibles para la base de datos actual.

- un ejemplo de una librería de objetos es la librería de objetos Microsoft DAO 3.0, la cual permite referenciar el modelo de objetos DAO.
- Un ejemplo de una librería de base de datos es el archivo WZLIB.MDA.

Nota De manera predeterminada, Microsoft Access carga las siguientes librerías de tipos como referencias: Visual Basic para Aplicaciones, Microsoft Access y la Librería de Objetos Microsoft DAO. No pueden eliminarse estas referencias, aun cuando se deseleccionen estas opciones

▶ **Para mostrar el cuadro de dialogo de referencias.**

- Seleccione la opción Referencias en el menú Herramientas cuando este abierto un modulo de Visual Basic.

La siguiente tabla describe la funcionalidad de los elementos individuales del cuadro de referencias

Este elemento	Hace esto
Referencias disponibles	Junto a cada uno de los elementos en la lista de referencias disponibles se encuentra una caja de selección que indica si la base de datos actual (o proyecto) esta o no referenciando este elemento. Pueden agregarse o eliminarse referencias activando o desactivando las cajas de selección y dando Aceptar.
Resultado	Cuando se selecciona una referencia de la lisa, Microsoft Access muestra la ruta completa y el lenguaje de la librería seleccionada o proyecto este o no habilitada en la base de datos actual (o proyecto). Las referencias que no están físicamente disponibles aparecen con el prefijo "Perdida"
Botones de prioridad	Las flechas arriba y debajo de prioridad permiten cambiar el orden en el cual Visual Basic trata de ligar a la lista de objetos mostrada. No puede usar estas

flechas para cambiar el orden de asociación de las librerías predeterminadas (las librerías de objetos VBA y Microsoft Access).

Botón Examinar — Este botón permite seleccionar archivos no listados aun en la lista de Referencias Disponibles Para referencias ubicadas en una red compartida, debe usarse la ruta con la convención universal de nombres (UNC) cuando se registre.

Examinador de objetos

Microsoft Access soporta el mismo Examinador de Objetos que otras aplicaciones de Microsoft Office.

El Examinador de Objetos es una caja de dialogo modal que permiten examinar todos los procedimientos, métodos y propiedades disponibles para la base de datos o proyecto actual.

El Examinador de Objetos esta disponible en el menú Ver de un modulo estando en la vista de diseño en la barra de herramientas de Visual Basic.

Use el Examinador de Objetos para:

- aprender sobre los diferentes objetos disponibles en Microsoft Access, Visual Basic y otras librerías.
- Ver los procedimientos, propiedades y métodos disponibles en objetos o librerías.
- Seleccionar procedimientos métodos y propiedades, y pegar la sintaxis directamente en el modulo de código.

5.4 Preguntas de repaso

1. ¿que objetos permiten crear módulos de clase?

2. El procedimiento Property siguiente esta ubicado en el modulo de una forma llamada frmSignOn. ¿Qué instrucción causaría que el siguiente código sea ejecutado desde la ventana de depuración?. Nota: la variable mUSer esta declarada como privada.

```
Prperty Let UserName (sNAme As String)

mUser = sName

MsgBox "Bienvenido " & mUser

End Property
```

3. El siguiente procedimiento Property esta ubicado en el modulo de un formulario llamado frmSignOn ¿Qué instrucción causaría que el siguiente código sea ejecutado desde la ventana de depuración?. Nota: la variable mUSer esta declarada como privada.

```
Prperty Get UserName (sNAme As String)

mUser = sName

End Property
```

4 El siguiente procedimiento Property esta ubicado en el modulo de un formulario llamado frmSignOn. ¿Qué instrucción causaría que el siguiente código sea ejecutado?.

```
Prperty Let CompanyList(List As Recordset)

Set mList = List.Clone

mList.MoveLAst

MsgBox "Numero de compañias : " & mList.RecordCount

End Property
```

5. El siguiente procedimiento esta declarado en el modulo del formulario frmCompany. ¿Qué instrucción, cuando se ejecuta desde frmOrders, puede exitosamente llamar al código de ejemplo mostrado?

```
Private Sub InitCOmpany()

' ... realiza algun proceso

End Sub
```

6. ¿Qué instrucción puede crear otra instancia del formulario frmCompany?

Capítulo 6: Integrando aplicaciones de Microsoft Office.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Describir como funciona la automatización OLE con Microsoft Access.
- ☑ Compartir datos mediante el con control de objetos ligados e incrustados.
- ☑ Controlar otras aplicaciones de Microsoft Office a través del uso de código de automatización OLE.
- ☑ Controlar Microsoft Access desde otras aplicaciones a través del uso de código de automatización OLE.

6.1 Aplicaciones centradas en documentos.

Usando OLE, existen varias formas de integrar funcionalidad desde diversas aplicaciones.

OLE es una tecnología que permite integrar información y funcionalidad desde diferentes aplicaciones.

Aplicación centralizada En el pasado, cuando un desarrollador creaba una solución integrada usando OLE, se enfocaba a la programación de la aplicación en vez de a los datos de la aplicación.

Documento centralizado Una aplicación centrada en documentos se enfoca en los datos y su presentación desde el punto de vista del usuario.

Por ejemplo, mediante el uso de automatización OLE, puede integrar un formulario de Microsoft Access con una hoja de calculo de Microsoft Excel, una gráfica de Microsoft Excel y un documento de Microsoft Word. La integración de estos diferentes tipos de información crea una solución que se enfoca en los datos en lugar de en las aplicaciones con las cuales fueron creados.

Aplicaciones centradas en documentos usando Microsoft Office La habilidad para crear aplicaciones centradas en documentos es a veces llamada programabilidad inter-aplicaciones. Este modulo se centra en la programabilidad inter-aplicaciones usando aplicaciones de Microsoft Office.

6.2 Referenciando un objeto usando una librería de objetos.

Una librería de objetos contiene la descripción de todos los objetos que proporciona una aplicación, incluyendo definiciones de todas las propiedades y métodos disponibles. Si una aplicación proporciona una librería de objetos, debe añadirse una referencia a la librería de objetos en la aplicación de Microsoft Access antes de usar los objetos de esa librería.

Nota debe estar en un modulo para tener el comando Referencias disponible en el menú Herramientas.

Para agregar una referencia a una librería de objetos de una aplicación

1. En el menú Herramientas, seleccione Referencias. Microsoft Access mostrara el cuadro de referencias.

2. En la caja de Referencias disponibles, seleccione el nombre de la referencia conteniendo los objetos que se desea usar en la aplicación.

   Use el botón Examinar para buscar la librería de objetos conteniendo el objeto deseado. Las librerías de objetos usualmente tienen la terminación .TLB o .OLB. Las aplicaciones (.EXE) y las librerías de vínculos dinámicos (.DLL) también pueden proporcionar librerías de objetos.

3. En el menú Ver, seleccione Examinador de Objetos para ver la librería de objetos referenciada. Seleccione la librería de objetos apropiada en el cuadro Librerías/Bases de datos.

   Puede usar todos los objetos, propiedades y métodos listados en el Examinador de Objetos en la aplicación.

6.3 Accesando objetos ligados e incrustados.

La habilidad para ligar e incrustar objetos es una de las características de OLE. Esta sección abarca la vinculación e incrustación desde una perspectiva del usuario y la interface del usuario.

Cuando usar vinculación sobre incrustación.

Las aplicaciones pueden compartir objeto con otras aplicaciones usando vinculación e incrustación. Cuando los usuarios vinculan o incrustan estos objetos en sus archivos, esto les permite crear documentos compuestos cuyos datos se originan en diferentes aplicaciones. La diferencia entre vincular e incrustar objetos tiene que ver con el lugar donde residen los datos. Este tópico se centra en la vinculación e incrustación desde la perspectiva de cual enfoque es el mas apropiado, dada una tarea particular.

Objetos ligados e incrustados

Las aplicaciones que soportan vinculación (ligado) e incrustación pueden insertar objetos dentro de una aplicación de Microsoft Access a través de la interface del usuario. Los objetos ligados e incrustados aparecen como parte de la base de datos. Puede mover. eliminar o editarlos de la misma manera que cualquier otra parte de la base de datos.

Trabajando con las propiedades de los objetos OLE.

Este tópico lista y da una breve descripción de cada una de las propiedades usadas cuando se trabaja con objetos OLE.

Cuando usar vinculación sobre incrustación.

Vincular a un objeto en otra aplicación e incrustar el objeto directamente en la aplicación proporcionan diferente funcionalidad. Elegir el método apropiado depende en lo que el usuario requiera hacer con los datos.

Objetos vinculados Un objeto vinculado tiene las siguientes características.

- El objeto es creado en una aplicación separada de la que esta trabajando actualmente el usuario.
- Se inserta un nicho para el objeto vinculado en la aplicación.
- La aplicación que crea el objeto vinculado almacena y administra sus datos

Cuando se actualiza el objeto vinculado. por ejemplo un archivo de hoja de calculo de Microsoft Excel dentro de una aplicación de Microsoft Access, las modificaciones aparecen en la aplicación nativa (Microsoft Excel).

Objetos incrustados Cuando se incrustan objetos en un archivo. se incluyen la presentación de datos del objeto y todos los datos necesarios par editarlos *dentro* del documento compuesto que se esta creando. Esto hace que los archivos que contienen documentos incrustados sean significativamente mas grandes que aquellos que contienen objetos vinculados.

Incrustar un objeto en la aplicación es una buena elección sobre la vinculación a un objeto cuando la información o es estática o cambia muy esporádicamente.

Por ejemplo, una aplicación puede contener un formulario que tiene un bitmap con el logotipo de la compañía.

Objetos ligados e incrustados

Microsoft Access puede incrustar tanto objetos que se crean mientras se trabaja con Microsoft Access como objetos basados en archivos existentes de manera externa (tales como documentos de Word). Microsoft Access puede vincular solamente un archivo que ya exista.

diferencia entre vincular e incrustar objetos

la diferencia entre vincular e incrustar objetos radica en el lugar donde residen los datos.

- Cuando los datos son parte de un objeto *vinculado*, la aplicación original que creo los datos almacena los datos en el documento original.
- Cuando los datos son parte de un objeto *incrustado*, el archivo conteniendo el objeto incrustado almacena los datos.

Estos ejemplos de varios escenarios en Microsoft Access delinean los procedimientos para incrustar y vincular objetos.

▶ **Para incrustar un nuevo objeto en una tabla de Microsoft Access.**

1. Seleccione el campo de objeto OLE en la tabla dentro de la cual se desea insertar el objeto incrustado.
2. Del menú Insertar, seleccione Objeto.
3. Seleccione la opción Crear nuevo y seleccione el tipo de objeto deseado de la lista de Tipos de objetos.

   Esta opción se emplea para crear e incrustar un nuevo objeto.
4. De Aceptar.

▶ **Para incrustar un objeto de archivo existente en un formulario de Microsoft Access.**

1. Del Menú Insertar, seleccione Objeto.
2. Seleccione la opción Crear desde archivo El cuadro de dialogo de Insertar Objeto cambiara adecuadamente.
3. Escriba el nombre en el cuadro de Archivo, o con el botón Examinar localice el archivo deseado.
4. En la ventana de Examinar seleccione Aceptar Esto cerrará la ventana de Examinar, permaneciendo abierta el cuadro de Insertar.
5. Seleccione Aceptar.

▶ **Par vincular un objeto de archivo existente en un formulario de Microsoft Access.**

5. Del menú Insertar, seleccione Objeto.
6. Seleccione la opción Crear desde archivo. El cuadro de dialogo de Insertar Objeto cambiara adecuadamente.
7. Siga los pasos 3 y 4 del procedimiento anterior.
8. Seleccione la opción Vincular en el cuadro de dialogo Insertar Objeto.
9. Seleccione la opción Mostrar como Icono si se desea.
10. De click en Aceptar.

Microsoft Access inserta una imagen del contenido del archivo en el formulario. La imagen esta vinculada con el archivo tal que el formulario reflejara cualquier modificación hecha al archivo.

Puede querer tratar de modificar el archivo fuente que acaba de vincular al formulario. Notara que los datos cambian cuando el archivo fuente es actualizado.

Nota debe asegurarse de instalar adecuadamente las aplicaciones que use para crear objetos incrustados; de otra forma, no aparecerán en la lista de Tipos de Objeto en el cuadro de dialogo Insertar Objeto.

Trabajando con las propiedades de los objetos OLE.

Existen varias propiedades que son expuestas por los objetos OLE. Un objeto OLE es cualquier objeto en una aplicación que soporta el protocolo OLE.

La siguiente tabla lista y describe brevemente cada una de las propiedades de los objetos OLE.

Propiedad	Descripción
Action	Especifica la operación a ejecutar sobre un objeto OLE.
AutoActivate	Especifica como el usuario puede activar el objeto OLE.
Class	Especifica o determina el nombre de clase del objeto OLE incrustado.
Enabled	Especifica si el control puede tener el enfoque en una vista de Formulario.
Locked	Especifica si el usuario puede editar los datos en un control en la vista de Formulario.
ObjectVerbs	Determina la lista de verbos que soporta el objeto OLE. Un verbo, tal como **edit** o **play**, especifica una operación que el usuario puede ejecutar sobre el objeto OLE.
ObjectVerbsCount	Determina el número de verbos soportados por un objeto OLE.
OLEType	Determina si un control contiene un objeto OLE, y si así es, si el objeto es vinculado o incrustado.

OLETypeAllowed	Especifica el tipo de objeto OLE que puede contener un control.
SourceDoc	Especifica el archivo que será vinculado o incrustado cuando se cree un objeto vinculado o incrustado usando la propiedad **Action**.
SourceItem	Especifica los datos a ser vinculados cuando se cree un objeto OLE vinculado.
UpdateOptions	Especifica como son actualizados los objetos vinculados.
Verb	Especifica la operación que se ejecuta cuando un objeto OLE es activado

6.4 Automatización OLE.

Esta sección cubre la implementación programática de la automatización OLE.

Integrando aplicaciones usando automatización OLE.

Puede usar automatización OLE para iniciar aplicaciones y crear objetos. La automatización OLE es especialmente útil si se necesita usar las características de una aplicación, tal como Microsoft Word, dentro de otra aplicación, tal como Microsoft Access. Con automatización OLE, puede integrar la funcionalidad de ambas aplicaciones en un simple procedimiento.

Funciones de Automatización OLE.

Generalmente, se emplean o la función GetObject o la función CreateObject para abrir o tener acceso a objetos de Automatización OLE de otras aplicaciones. La función que se use dependerá de si la aplicación involucrada tiene o no una librería de objetos.

Integrando aplicaciones usando automatización OLE.

La programabilidad inter-aplicaciones incluye la idea de "objetos programables", los cuales se pueden controlas a través de automatización OLE. Esto incrementa la usabilidad y reusabilidad haciendo posible el construir aplicaciones que controles objetos de otras aplicaciones

La automatización OLE permite controlar objetos y compartir información en otras aplicaciones.

La automatización OLE es una característica de los objetos ligados e incrustados (OLE) que permite controlar objetos en otras aplicaciones fuera de la aplicación. Las aplicaciones que soportan automatización OLE proporcionan objetos que se usan de la misma forma que los objetos de Microsoft Access.

Puede usar el código de Visual Basic con aplicaciones que soportan automatización OLE para ejecutar las siguientes tareas:

- Controlar objetos en otras aplicaciones.
- Obtener información de otras aplicaciones.
- Enviar información a otras aplicaciones.

Objetos de automatización Ole.

Puede usar automatización OLE para crear y manipular objetos de Microsoft Access que estén disponibles a través de la interface del usuario (tales como formularios y reportes). Puede además usar la automatización OLE para manipular los objetos que no están automáticamente visibles al usuario, tales como objetos Workspace

Estos objetos son llamados *objetos de automatización OLE*. Debido a que el usuario no ve automáticamente los objetos de automatización OLE, puede usar estos objetos para ejecutar tareas que

no involucren la interacción del usuario. Sin embargo, a diferencia de esos objetos que pueden ser accesados a través de la interface del usuario, solo puede accesar los objetos de automatización OLE a través de un lenguaje de macros o un lenguaje de programación, tal como Visual Basic.

La siguiente tabla muestra ejemplos de objetos de automatización OLE en otras aplicaciones Microsoft que puede usar en Microsoft Access

Aplicación	Objetos de automatización OLE
Microsoft Word	**Application**
	Document
	WordBasic
Microsoft Excel	**Application**
	Chart
	Worksheet
Microsoft Project (versión 4.0)	**Application**
Microsoft Access	**Application**
	Module
	Forms
	Reports

Nota Visual Basic 4.0 permite crear sus propios servidores de automatización OLE.

Funciones de Automatización OLE.

La instrucción Set y las funciones GetObject y CreateObject son las principales palabras clave para implementar la automatización OLE. Después de hacer referencia a un objeto, este se manipula con sus propios métodos y propiedades intrínsecos

Función CreateObject LA función CreateObject crea un objeto de automatización OLE. Si una aplicación no esta registrada con Microsoft Access, puede emplearse la función CreateObject para crear un nuevo objeto en la aplicación.

Use la siguiente sintaxis para la función CreateObject

```
Set VariableDeObjeto = CreateObject(programID)
```

La función CreateObject no crea una instancia de una aplicación. Crea una instancia de un objeto que la aplicación expone. Dependiendo de los objetos que la aplicación expone, puede o no obtener una nueva instancia de la aplicación cuando usa la función CreateObject.

```
Set WordObj = CreateObject("Word.Basic")
```

Función GetObject La función GetObject obtiene un objeto de automatización OLE desde un archivo. Use esta función para obtener un objeto existente desde una aplicación. El objeto puede ser un archivo en disco o algún otro objeto desde la aplicación que esta ejecutándose.

Sugerencia si se desea trabajar con objetos de automatización OLE, primero llame a GetObject para ver si el objeto ya existe. Si lo es, use el objeto. Si no, llamea CreateObject para crear un nuevo objeto de automatización OLE para esa aplicación.

Use la siguiente sintaxis para la función GetObject

```
Set VariableDeObjeto = GetObject(RutaDelARchivo, clase)
```

El siguiente ejemplo muestra como usar la función GetObject para obtener un objeto de Word desde un archivo.

```
Set WordObj = GetObject("C:\Q1.DOC")
```

Instrucción Set La instrucción Set asigna una referencia de objeto a una variable o propiedad. Si se desea usar el mismo objeto repetidamente, use la instrucción Set para crear una variable que se refiera a ese objeto. Después de crear la variable, puede usar el nombre corto de la variable cuando ejecute acciones sobre el objeto.

Use la siguiente sintaxis para la instrucción Ser:

```
Set VariableDeObjeto = [CreareObject] o [GetObject]
```

El siguiente ejemplo muestra como usar la instrucción Set para asignar la referencia de un objeto a una variable llamada WordObj

```
Dim WordObj as Object
Set WordObj = CreateObject("Word.Basic")
```

Use las instrucciones estándar de Visual Basic para ejecutar operaciones OLE.

Para hacer esto ...	**Use esto ...**
Crear nuevos objetos u obtener existentes	Set, CreateObject o GetObject. (Use Set su la aplicación soporta librerías de objetos).
Accesar los métodos y propiedades de un objeto	Coloque después del nombre del objeto el operador punto(.) y el nombre del método o propiedad (igual que accesar un objeto en Microsoft Access)
Cerrar un objeto	Método Close para el objeto.

Ejemplo de automatización OLE.

Este ejemplo muestra como Microsoft Access puede incrustar un objeto en un documento de Word usando automatización OLE.

▶ **Para ejecutar el ejemplo**

1. inicie Microsoft Access y abra la base de datos Neptuno.

2. Cree un nuevo modulo y escriba el siguiente procedimiento.

```
Sub OLEAuto()
      ' La variable estática existirá aun después de finalizado el
      procedimiento.
      Static MSWord AS Object
      Set MSWord = CreateObject("word.basic")   ' Crea el objeto Word
      With MSWord
            .FileNewDefault   'Crea un nuevo documento
            .AppMaximize      'Maximiza Word
            .FontSize 28      ' cambia el tamaño de la letra a 28
            .centerpara ' Centra el texto
            .Bold ' Establece el estilo negritas
            .INSERT "Este es un texto de prueba"      ' Inserta este texto
      End With
End Sub
```

3. Guarde el modulo como Automatización OLE y ejecute el procedimiento.

6.5 Microsoft Access como servidor de automatización OLE.

Microsoft Access proporciona sus propios objetos para ser usados por otras aplicaciones. Desde cualquier aplicación que actúe como un controlador de automatización OLE, puede iniciar Microsoft Access y manipular sus objetos. Adicionalmente, puede usar Microsoft Access como un controlador de automatización OLE para manipular otras instancias de Microsoft Access que estén actuando como servidores de automatizaron OLE.

Si una aplicación soporta automatizaron OLE, puede usar código VBA para manipular objetos que la aplicación expone a otras aplicaciones.

Un servidor de automatización OLE es una aplicación que proporciona sus objetos para que otras aplicaciones las utilicen.

Un controlador de automatización OLE es una aplicación que puede accesar y manipular los objetos de automatización OLE.

- puede escribir código en Visual Basic para abrir tablas de Microsoft Access en una base de datos de Microsoft Access. Microsoft Access será el servidor de automatización y Visual Basic el controlador de automatización

El siguiente ejemplo muestra el uso de Microsoft Access como servidor de automatización OLE para abrir un formulario de la base de datos Neptuno.

```
Sub AccessForm
      Static acc AS Object
      Set acc = CreateObject("Access.Application")
      acc.visible = True
      acc.OpenCurrentDatabase
      filepath:="C:\MSOffice\Access\Samples\Neptuno.mdb"
      add.DoCmd.OpenForm FormName:="Categorias"
      acc.DoCmd.Maximize
End Sub
```

6.6 *Preguntas de repaso.*

1. ¿Cuál es la diferencia entre objetos vinculados e incrustados?

2. ¿Cuándo debe elegir entre usar un objeto incrustado en vez de un objeto vinculado?

3. ¿Qué propiedad de un objeto OLE determina la lista de todas las acciones que un objeto puede ejecutar, tales como edit, play, record y otras?

4. ¿Cuál es la diferencia entre documento centrado y aplicación centrada?

5. ¿Dónde debe ir para encontrar información acerca de un objeto OLE y sus propiedades y métodos?

6. ¿Qué propiedad de un objeto OLE especifica que accion, tal como incrustar, vincular o actualizar, para ejecutar sobre un objeto?

7 ¿Qué hace la propiedad SourceDoc sobre un objeto OLE?

8. ¿Qué instrucción debe usar su desea crear un documento nuevo de Word? (WordObj es declarada como tipo Objeto).

9. ¿Cuál es la diferencia entre servidor de automatización OLE y un controlador de automatización OLE?

10. ¿Qué instrucción inicia Microsoft Excel y carga la hoja de calculo "Q1.XLS"? (xlsObj es declarada como tipo Object).

11 ¿Qué se necesita para tener visible una librería de objetos en el Examinador de Objetos?

12. ¿Puede abrir la base de datos Neptuno programáticamente desde Excel?, ¿Por qué?

13. Esta ejecutando Microsoft Excel y ejecuta el siguiente código:

```
Set xlObj = CreateObject("excel.application")
```

14. ¿Cuantas instancias de Microsoft Excel están corriendo ahora en su equipo?

Capítulo 7: Controles OLE.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Identificar los tipos de controles OLE.
- ☑ Agregar controles a un formulario.
- ☑ Manipular controles OLE tanto manualmente como programáticamente.
- ☑ Establecer propiedades para controles personales.

7.1 Introducción a los controles OLE.

El Toolkit del Desarrollador de Microsoft Access (ADT) proporciona formas posteriores se extender las aplicaciones a través del uso de controles OLE.

¿Qué es un control OLE?

Los controles OLE son componentes reusables que se colocan en un formulario para mejorar la interacción del usuario. Proporcionan funcionalidad especifica que facilita tareas tales como usar cuadros de dialogo comunes que están disponibles en Windows 95. En Microsoft Access, los formularios son los contenedores para los controles OLE.

Los controles OLE pueden o no tener una interface de usuario, pero cada control tiene propiedades, métodos y eventos que pueden ser usados para afectar los datos y la funcionalidad de los controles.

Ventajas de los controles OLE.

Los controles OLE permiten añadir rápidamente poderosas características nuevas a una aplicación. Pueden usarse estos controles como componentes reusables en las aplicaciones, con el consiguiente ahorro de tiempo de desarrollo. Proporcionan también el concepto "mira y siente" de Windows 95.

7.2 Tipos de controles OLE.

La funcionalidad individual de un control OLE determina aquello que es visible y que no en la ejecución.

Visible en tiempo de diseño y tiempo de ejecución.

Algunos controles OLE tienen funcionalidad que involucra interacción directa con el usuario. Estos controles son visibles tanto durante el diseño como en la ejecución. Ejemplos de controles que son visibles tanto durante el diseño como durante la ejecución incluyen los controles **ListView**, **Slider**, **Data Outline** y **Toolbar**.

Visible solo durante el diseño.

Los controles OLE que no requieren interacción directa del usuario aparece solo durante el diseño de un formulario. Por ejemplo, solo se puede ver el control para un cuadro de dialogo común cuando se esta trabajando con el formulario en la vista Diseño

Visible en tiempo de diseño y tiempo de ejecución.

Los siguientes párrafos describen varios controles que vienen con el Kit de desarrolladores de Microsoft Access para Windows 95 y son visibles tanto durante el diseño como durante la ejecución

ListView El control **ListView** permite organizar las entradas de una lista, llamados objetos **ListItems**, dentro de una de cuatro diferentes vistas:

- Iconos grandes (estándar).
- Iconos pequeños
- Lista.
- Reporte.

Este control funciona de manera muy similar a la ventana de Mi PC en Windows 95.

TreeView El control **TreeView** muestra una lista jerárquica de objetos Nodo, cada uno de los cuales consiste de una etiqueta y un bitmap opcional. El control **TreeView** funciona de la misma forma que el panel izquierdo del Explorador de Windows 95.

DataOutline El control **DataOutline** muestra una vista jerárquica de los datos en un formulario de Microsoft Access El control divide cada jerarquía en niveles y puede mostrar hasta 16 niveles.

ProgressBar El control **ProgressBar** muestra el progreso de una operación larga mediante el llenado de un rectángulo con bloques de izquierda a derecha mientras la operación es completada.

StatusBar El control **StatusBar** proporciona una ventana, generalmente en la parte inferior de un formulario, a través del cual una aplicación puede mostrar diferentes tipos de datos de estado. El control puede ser dividido en un máximo de 16 objetos **Panel** que están contenidos en la colección **Panels**

TabStrip Un **TabStrip** es similar a los separadores de un cuaderno o las etiquetas en un grupo de folders de archivo. Mediante el uso del control **TabStrip**, puede definir múltiples paginas para la misma área de una ventana o cuadro de dialogo en una aplicación.

ToolBar El control **ToolBar** contiene una colección de objetos **Button** empleados para crear una barra de herramientas que esta asociada con una aplicación.

RichText El control **RichText** permite al usuario introducir y editar texto mientras proporciona además características avanzadas de formato que el control convencional **TextBox**. El control **RichText** ofrece la capacidad de modificar formatos para cada carácter sin afectar el formato del contenido completo del cuadro de texto.

Slider El control **Slider** es una ventana conteniendo un deslizador y pequeñas marcas opcionales. Puede mover el deslizador mediante arrastre, dando click con el ratón en cualquier lado del deslizador o usando el teclado.

Calendar El control **Calendar** muestra un calendario mensual que puede ser insertado en un formulario de Microsoft Access.

SpinButton El **spinButton** es un control de tipo spinner que puede usar con otro control para incrementar o decrementar números. Puede usarlo también para recorrer hacia delante o hacia atrás a través de un rango de valores o lista de elementos.

Visible solo durante el diseño.

Debido a que el control **CommonDialog** e **ImageList** no requieren de interacción por parte del usuario, estos no aparecen en un formulario en la vista Formulario; en vez de ello, solo aparecen en la vista Diseño.

El siguiente párrafo describe los controles **CommonDialog** e **ImageList**.

- **CommonDialog**

El control **CommonDialog** proporciona un conjunto estándar de cuadros de dialogo para operaciones tales como abrir, guardar e imprimir archivos o seleccionar colores y tipos de letra. Estos cuadros de dialogo comunes son visibles solo durante la ejecución.

- **ImageList**

Un control **ImageList** contiene una colección de objetos **ImageList**, cada uno de los cuales puede ser referido por su índice o llave. Se usa este control como un almacén central para proporcionar convenientemente otros controles con imágenes.

7.3 Registrando y agregando controles OLE.

Registrando controles OLE

Los controles deben ser registrados antes de que puedan ser usados con los formularios. Los controles OLE que vienen con el ADT son registrados predeterminadamente durante la instalación. El comando Controles Personalizados del menú Herramientas permite registrar cualquier nuevo control OLE.

El comando Controles Personalizados en el menú Herramientas además permite quitar el registro a cualquier control OLE que no se vaya a usar mas.

Agregando controles OLE a un formulario.

El comando Controles Personalizados en el menú Insertar permite agregar un control a un formulario.

7.4 Trabajando con controles OLE.

Esta sección cubre aspectos programáticos de los controles OLE, incluyendo el establecimiento de propiedades, uso de métodos y repuesta a eventos. Un archivo simple identificado con la extensión .OCX contiene todas las propiedades, métodos y eventos para un control OLE.

Estableciendo propiedades.

Este tópico contrasta el establecimiento de propiedades a través de una interface de usuario con el establecimiento mediante código. Usar el control **Calendar** es un ejemplo, ya que puede cambiar el valor de propiedades tales como **FirstDay** ya sea a través de las propiedades del control o mediante las instrucciones apropiadas de Visual Basic. Este tópico diferencia además entre propiedades estándar y personalizadas.

Usando métodos.

Se emplean los métodos asociados con un control OLE para manipular ese control, por ejemplo, puede usar el método **Refresh** del control **Calendar** para "repintar" el calendario. Este tópico muestra el uso del método **NextDay** para modificar el día actual al siguiente día en el control **Calendar**.

Respuesta a eventos.

Este tópico da un ejemplo simple de cómo un control OLE responde a eventos, usando los principios de manejado por eventos (*event-driven*). El ejemplo usa el evento **Update** del control **Calendar** para mostrar como dando click en una fecha nueva dispara un código d evento

Uso del control CommonDialog.

Este tópico cubre el uso del control **CommonDialog** para proporcionar la funcionalidad de algunos de los cuadros de dialogo mas comunes de Windows 95 dentro de las aplicaciones

Estableciendo propiedades.

Pueden manipularse las propiedades personalizadas de un control OLE a través de la interface de usuario o mediante la escritura de código.

Nota Es importante entender que los controles OLE tienen propiedades estándar y propiedades personalizadas. Microsoft Access usa las propiedades estándar para identificación, características físicas, ubicación, etc. Las propiedades personalizadas son únicas para control OLE. Pueden accesarse a través de la hoja de propiedades del control

---

modificando propiedades a través de la interface de usuario Mediante la selección del control y viendo sus propiedades, pueden modificarse las opciones disponibles. Por ejemplo, en el control **Calendar**, las propiedades personalizadas están divididas en tres categorías: General, Colores y Fuentes.

▶ **Para cambiar el valor de la propiedad FirstDay desde la interface de usuario**

1. seleccione el control y de click con el botón derecho del ratón.
2. Desde el menú Sub, elija Objeto Control **Calendar** y elija propiedades. Esto mostrara la hoja de propiedades del control **Calendar**.
3. Seleccione el separador Propiedades General y elija un valor de la lista desplegable **FirstDay**.
4. De click en Aceptar.

Cambiar propiedades programáticamente Para manipular las propiedades programáticamnte, emplee la siguiente sintaxis:

NombreDelControl.Propiedad

Por ejemplo, para establecer la fecha del control **Calendar**, use la propiedad **Value** del control

OleCalendar.Value = "1/1/96"

Usando métodos.

Los controles OLE exponen su funcionalidad a través de sus métodos. Estos métodos invocan acciones sobre ese control.

Por ejemplo, para establecer el valor del control **Calendar** a la fecha de hoy, use el método **NextDay**. Este método incrementa el día, reinicia el mes(si el día actual es el ultimo día del mes), y repinta el control en una simple operación.

OleCalendar.NextDay

Respuesta a eventos.

Los controles OLE pueden responder a cualquier número de eventos. El control recibe mensajes, los cuales inician eventos.

La programación *manipulada por eventos* es la forma de dirigir a los objetos en la aplicación a responder a acciones del usuario

En el ejemplo mostrado, cuando se da click en una fecha particular en el control **Calendar**, esa fecha es mostrada en el texto de la esquina superior izquierda Esto ocurre debido a que el control responde al evento **Update**, el cual ocasiona que se ejecute el código

Este ejemplo, nuevamente usando el control **Calendar**, actualiza el cuadro de texto cuando el valor del Calendario cambia

```
Private Sub oleCalendar_Updated(code As Integer)
      Me![TodaysDate] = oleCalendar.value
EndSub
```

Uso del control CommonDialog.

El control **CommonDialog** permite al usuario trabajar con los cuadros de diálogos Abrir, Imprimir, Fuente y Color. Este control implementa funcionalidad múltiple en un simple control OLE.

El control **CommonDialog** proporciona un conjunto estándar de cuadros de dialogo para operaciones tales como abrir, guardar e imprimir archivos o seleccionar colores y fuentes.

El siguiente ejemplo abre el cuadro de dialogo Abrir de Windows 95, permite al usuario seleccionar un archivo y mostrar el contenido de ese archivo en un control Ole **RichText**.

```
Sub cmdFile_Click()
      ' Invoca al cuadro de dialogo Archivo
      oleCommonDialog.ShowOpen
      ' Ahora abre el archivo dentro del control RichText
      oleRichText.LoadFile oleCommonDialog.Filename
End Sub
```

7.5 Preguntas de repaso.

1. Todas las siguientes instrucciones sobre controles OLE son ciertas excepto:
   a. Los controles OLE son reusables.
   b. Los controles OLE siempre tiene una interface de usuario.
   c. Los controles ole son colocados en formulario.
   d. Cada control OLE tiene propiedades, métodos y eventos.

2. Si. en una aplicación se desea crear una lista de objetos jerárquicos dónde el usuario pueda seleccionar (parecida a la lista de contenido del CD Mastering Access), ¿qué control OLE usaría?

3. ¿Cuáles son algunas de las formas diferentes de obtener controles OLE?

4. ¿Cómo registra un control OLE?

5. En el siguiente código de ejemplo. ¿qué es "**Value**"? (**oleCalendar** es el nombre del control OLE).

```
OleCalendar.Value = Date
```

6. ¿que hará el siguiente código de ejemplo? (**oleCalendar** es el nombre del control OLE).

```
Private Sub oleCalendar_Updated()
MsgBox oleCalendar.Value
End Sub
```

7. ¿Qué control proporciona la habilidad de modificar fuentes durante la ejecución?

Capítulo 8: Librerías de Vínculos Dinámicos (DLL).

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Describir el objetivo y ventajas de las DLL's.
- ☑ Mapear tipos de datos en C hacia tipos de datos de Visual Basic para Aplicaciones.
- ☑ Describir el propósito del tipo de datos **Any**.
- ☑ Pasar un apuntador **Null** a un procedimiento DLL.
- ☑ Describir la diferencia entre las palabras reservadas **ByVal** y **ByRef**.
- ☑ Crear aplicaciones VBA que llamen procedimientos DLL de Windows para:
  - ☑ Ocasionar que la barra de titulo de una ventana parpadee.
  - ☑ Revisar si la forma esta minimizada.

8.1 Introducción a los DLL's.

Esta sección introduce las librerías de vínculos dinámicos (DLL's), dando razones para usarlas y lista algunas fuentes de ayuda para trabajar con ellas.

¿Qué son las DLL's?

puede usar DLL's para complementar las tareas en un programa que Microsoft Access no puede ejecutar, o ejecutar ciertas funciones complejas de programación que no pueden realizarse fácilmente en VBA.

Librerías externas vinculadas durante la ejecución Una DLL es una librería de procedimientos externos a una aplicación que puede ser llamada desde la aplicación.

Un DLL esta vinculada durante la ejecución y no esta ligada al archivo .EXE.

Puede ser compartida por muchos programas Las DLL's son compartibles y reentrantes. El código en una DLL puede ejecutarse mas rápido que el código de Visual Basic.

El sistema operativo Windows esta compuesto en parte por un conjunto de DLL's de sistema, incluidos User32, GDI32 y Kernel32 Estas DLL proporcionan la interface del usuario de Windows, interface gráfica, servicios de sistema tales como administración de memoria y otras capacidades, tales como grabación y reproducción multimedia. Puede llamar procedimientos en esas y otras DLL's para extender las capacidades de una aplicación.

¿Por qué usar DLL's?

aunque puede escribir aplicaciones enteras en VBA, hay situaciones cuando se puede desear llamar una u otra DLL.

Completar tareas no disponibles en VBA las DLL's pueden ejecutar tareas que no son posibles desde VBA directamente. Por ejemplo, puede invocar un procedimiento DLL que ocasione que la aplicación permanezca como la ventana superior

Desempeño mas rápido Si se tienen ciertos procedimientos que requieran de un desempeño mas allá del que puede obtenerse actualmente con Microsoft Access, puede escribir estos como DLL y entonces invocarlos desde VBA

Actualización independientemente de la aplicación Puede actualizar DLL's sin tener que modificar o recompilar una aplicación que invoca la DLL. Esto proporciona mayor modularidad y facilidad en el mantenimiento de aplicaciones

¿Que procedimientos DLL pueden ser llamados?

DLL's de Windows El sistema operativo Windows incluye 3 DLL's que contienen procedimientos que puede ser llamados:

- user32

USER32.DLL maneja tareas relacionadas con la administración de ventanas, menús, controles y cuadros de dialogo. **FlashWindow** es un ejemplo de un procedimiento de USER32.DLL que puede llamar para hacer parpadear una ventana.

- GDI32

GDI32.DLL es responsable de la salida de gráficas.

- Kernel32

KERNEL32.DLL maneja tareas del sistema operativo. **GetWindowsDirectory** es un ejemplo de un procedimiento que puede ser llamado para dar la ruta actual del directorio de Windows.

DLL's de terceros Algunas DLL's de terceros proporcionan las funciones de manipulación de archivos y otra funcionalidad que no esta disponible o expuesta en las DLL's de Windows o aplicaciones.

DLL's escritas por el usuario Finalmente, se pueden crear DLL's propias. Típicamente se emplea el lenguaje C o C++ para escribir una DLL. Puede además usar Microsoft Visual Basic para crear OLE DLL.

Información sobre procedimientos DLL de Windows.

Herramientas importantes de información acerca de las DLL's de Windows incluyen los archivos de ayuda WIN32API y el visor de API Win32.

Archivos de ayuda WIN32API Varios archivos de ayuda para el Win32API están disponibles a través de Microsoft Visual C++ y MSDN. Estos archivos de ayuda dan información acerca de las DLL's de Windows, tal como el propósito real de cada función y los parámetros y valores de devolución empleados.

Visor Win32 API El visor Win32 API es una aplicación que viene el Kit del Desarrollador de Microsoft Access para Windows 95 y Microsoft Visual Basic 4.0. Este archivo permite examinar a través de declaraciones, constantes y tipos incluidos en un archivo de texto API o una base de datos JET Una vez que se conoce la función que se desea llamar, puede copiar su declaración desde el visor

▶ **Para cargar el archivo API.**

1. De click sobre el icono del visor API Win32 para iniciar la aplicación.
2. Del menú Archivo, escoja Carga archivo de texto o Carga Base de datos.

Cuando carga un archivo de texto API, puede seleccionar elementos, copiar esos elementos al portapapeles, y subsecuentemente, copiarlos en el código de VBA.

▶ **Para copiar un elemento al código VBA**

1. De la lista Elementos Disponibles, seleccione una o mas entradas y seleccione Agregar Esto agrega los elementos seleccionados a la lista Elementos Seleccionados al fondo de la forma.
2. Seleccione Copiar. Todos los elementos en la lista se copiaran al portapapeles.
3. Vaya al modulo de Visual Basic en el cual desea copiar los elementos.
4. Del menú Edición, escoja Pegar.

8.2 Uso de DLL's con VBA.

Pasos para usar DLL's en VBA

Debe declarar el procedimiento de una DLL antes de llamarlo.

Declaración del procedimiento Copie y Pegue la declaración desde el archivo WIN32API MDB o WIN32API.TXT dentro de la sección de declaraciones de un formulario, modulo estándar o de clase.

El siguiente ejemplo declara el Windows API **FlashWindow** en un modulo de Microsoft Access.

```
Private Declare Function FlashWindow Lib "user32" (ByVal hwnd As Long, ByVal
bInvert As Long) As Long
```

Sugerencia puede copiar la declaración **FlashWindow** del archivo WIN32API.MDB usando el Visor Win API.

Si se declara un procedimiento DLL en un modulo estándar, es publico de manera predeterminada. Esto significa que el código en cualquier parte de la aplicación puede llamar al procedimiento.

Para declarar un procedimiento DLL en la sección declaraciones de un formulario o modulo de clase, debe incluir la palabra clave **Private** en la declaración.

Llamando al procedimiento una vez declarado el procedimiento, se invoca de la misma forma que se invocaría una instrucción o procedimiento de VBA.

El siguiente ejemplo muestra como llamar a un procedimiento DLL

```
Sub cmdFlash_Click()
      Rc = FlashWindow (Forms!form1.hWnd, 1)
End Sub
```

Guardando el trabajo VBA no puede verificar que se están enviando los valores correctos a un procedimiento DLL. Si se envían valores incorrectos, el procedimiento puede fallar, lo que puede ocasionar que la aplicación falle

Precaución Tenga cuidado cuando experimente con procedimientos DLL, y guarde el trabajo periódicamente

Detalles de la instrucción Declare

Debido a que los procedimientos que se llamando desde un DLL son externos a VBA, es necesario declarar esos procedimientos. Esto da a la aplicación acceso a dichos procedimientos durante la ejecución.

La instrucción Declare involucra varias partes Use la instrucción **Declare** para declarar un procedimiento. note que hay varias partes en esta instrucción:

1. La palabra clave **Private**.

2. El nombre del procedimiento.

3. El nombre de la librería en la que se encuentra el procedimiento.

   Si se tienen escritas DLL propias, puede sustituir el nombre del archivo, por ejemplo:

   ```
   Private Declare Function AddNum Lib "c:\add.dll" (ByVal a As Integer,
   ByVal b As Integer) As Integer
   ```

   Si no se incluye la ruta, el programa buscará en las rutas predeterminadas del sistema, comenzando por \\Windows y finalmente \\Window\System.

4. Alias de la función, no necesario.

   Ocasionalmente, un procedimiento DLL tiene un nombre que no es un identificador legal para VBA. Cuando sea este el caso, use la palabra clave Alias par renombrar el procedimiento. Por ejemplo:

   ```
   Private Declare Function lopen Lib "kernel32" Alias "_lopen" (ByVal
   lpPathName As String, ByVal iReadWrite As Long) AS Long
   ```

5. Los argumentos que requiere el procedimiento.

   El procedimiento de Windows **FindWindow** requiere como argumentos dos **Longs** Cada tipo de datos de cada argumento debe coincidir con el tipo de datos especificado en la declaración, y los argumentos debes ser pasados correctamente – ya sea por valor o por referencia.

Nota Los procedimientos **Function** devuelven valores, los **Sub** no.

Use un alias para declarar un procedimiento varias veces Para ayudar a prevenir errores con los tipos de datos, es una buena idea especificar correctamente el tipo de datos en la instrucción **Declare** y evitar usar el tipo de datos **Any**. Si se indican tipos de datos específicos en la declaración, Visual Basic revisa el tipo de datos y puede fallar durante la compilación en lugar de durante la ejecución.

Un ejemplo simple.

En el ejemplo mostrado de la ventana parpadeante, un procedimiento Sub invoca a **FlashWindow** de Windows API Esto es útil si se desea llamar la atención del usuario hacia una ventana en particular. En las siguientes paginas, se discutirán como funcionan todos los pasos.

1. Cree un formulario llamado Forma1.

2. Copie la declaración **FlashWindow** del archivo WIN32API.MDB usando el visor Win API y péguela en la sección de declaraciones de Forma1

```
Private Declare Function FlashWindow Lib "user32" (ByVal hwnd AS Long, ByVal
bInvert AS Long) AS Long
```

3. Agregue un botón de comando.

4. Agregue el siguiente código al evento click del botón de comando

```
Sub Button0_Click()
Dim rc AS Long
```

```
Rc = FlashWindow(Forms!Formal.hWnd, 1)
End Sub
```

5. Ejecute el programa. De click al botón para hacer que la ventana parpadee una vez.

8.3 Información general acerca del uso de DLL's.

By Val v.s. By Reference

Con los procedimientos DLL, puede pasarse un argumento ya sea como su valor real o mediante una referencia a una dirección de memoria.

Por valor Cuando se pasa un argumento por valor, una copia del contenido del argumento se envía al procedimiento. Si el procedimiento cambia el valor del argumento, esto no afecta a la variable original. Para pasar un argumento por valor. Coloque la palabra clave **ByVal** frente a la declaración del argumento en la instrucción Declare. El siguiente ejemplo muestra el usa de **ByVal** dentro de la instrucción Declare.

```
Private Declare Function FlashWindow Lib "user32" (ByVal hwnd As Long, By Val
bInvert AS Long) As Long
```

Por referencia -- predeterminado cuando se pasa un argumento por referencia, la dirección de la variable original se envía al procedimiento. Si el procedimiento modifica el valor de la variable, la variable original es modificada.

El siguiente ejemplo muestra el paso de argumentos por referencia. Aunque no se especifica la palabra clave **ByRef**, el ultimo argumento es sin **ByVal**. En otras palabras, **ByRef** es el predeterminado, a menos que se especifique otro.

```
Private Declare Function waveOutGetPlaybackRate Lib "winm.dll" (ByVal hWaveOut
As Long, lpdwRate As Long) As Long
```

Visual Basic pasa todos los argumentos por referencia de manera predeterminada. Aunque no es necesario especificar la palabra clave **ByRef**, puede usarse como documentación del código.

Nota Cuando se busca en la documentación de procedimientos DLL que emplean una sintaxis de lenguaje C, recuerde que C pasa todos los argumentos por valor, excepto los arreglos.

Convirtiendo declaraciones en C a tipos de datos de Visual Basic.

Debe estar pendiente de algunos elementos relacionados con la conversión cuando se declaren procedimientos DLL.

DLL's son típicamente escritos en C Los DLL's típicamente están escritos en lenguaje C o C++ Por lo tanto, se definen los argumentos requeridos usando los tipos de datos de C. la siguiente tabla lista las declaraciones comunes del lenguaje C y su equivalente en Microsoft Access.

Declaración en C	Equivalente Visual Basic	Llamada con
Lógicos	**ByVal S As Boolean**	Cualquier variable **Integer** o **Variant**
Apuntador a una cadena (LPSTR)	**ByVal S AS String**	Cualquier variable **String** o **Variant**
Apuntador a un entero (LPINT)	**I As Integer**	Cualquier variable **Integer** o **Variant**

Apuntador a un entero largo (LPDWORD)	**L As Long**	Cualquier variable **Long** o **Variant**
Apuntador a una estructura	**S As Rect**	Cualquier variable de tipo definido por el usuario

Declaración de DLL en Visual Basic Cuando se declaran procedimientos en Visual Basic, debe especificar el tipo de datos coincidente de Visual Basic.

Visual Basic no tiene un tipo de datos de carácter. Si se esta llamando a un procedimiento que espera un tipo de datos carácter, pase el valor ASCII del carácter en un entero. Esto funciona debido a que, como mínimo, siempre son enviados a la pila 2 bytes. Los tipos de datos Char e int son intercambiables cuando se pasan directamente como parámetros.

Nota EL lenguaje C no soporta el tipo de dato Currency. Si se necesita enviar un valor Currency desde Visual Basic hacia procedimientos DLL en C, debe convertir la variable a un tipo de datos Double. Entonces, debe convertirse de Double a Currency cuando se reciba información de la DLL.

Típicamente puede llamar cualquier procedimiento desde Visual Basic excepto aquellos que toman apuntadores de funciones como argumentos, o aquellos en los cuales las estructuras contienen apuntadores.

Declarando argumentos As Any.

En algunos casos, los procedimientos DLL están definidos para aceptar mas de un tipo de datos. Un ejemplo de esto es el procedimiento FindWindow.

```
Private Declare Function DinfWindow Lib "user32" Alias "FindWindowA" (ByVal
lpClassName As Any, By Val lpWindowName As Any) As Long
```

Este procedimiento devuelve el manejador de la ventana si la encuentra y 0 en caso contrario. Se proporciona ya sea el nombre de la clase o el título de la ventana. , con lo cual puede ser una cadena para alguno de los argumentos y un **Null** para el otro. Así pues, cada argumento puede aceptar tanto una cadena como un **Null**.

Usando el tipo de datos "Any" Para declarar un argumento que puede aceptar mas de un tipo de datos. se usa el tipo de datos Any. Este tipo de datos indica que Visual Basic no realizará verificación de tipo de datos. Es responsabilidad del programador estar en la certeza que el procedimiento DLL puede aceptar el parámetro como es pasado. Pasando un tipo incorrecto de datos puede resultar en un error de la aplicación.

Pasando apuntadores Null a DLL's

Declarar argumentos ByVal como Long o Any Cuando un procedimiento DLL acepta **Null**, espera el valor 0 en lugar de una cadena vacía Para pasar un valor **Null**, se usa el tipo de datos Long y se pasa el valor usando la palabra clave **ByVal** Si se declara un procedimiento de la siguiente forma:

```
Private Declare Function FindWindow Lib "User32" Alias "FindWindowA" (ByVal
lpClassName As Any, lpCaption As Any) As Long
```

Entonces se invoca de la siguiente forma:

```
Rc = FindWindow(vbNullString, "Calculator")
If rc = 0 then
      MsgBox "No se encontro la calculadora"
Else
MsgBox "Se encontro la calculadora"
End If
```

Pasando Null la constante vbNullString pasa el valor **Null** a la función

Pasando cadenas de caracteres.

Los procedimientos en varios DLL's esperan cadenas estándar de C que terminan en un carácter **Null**.

Paso de cadenas ByVal Si un procedimiento DLL espera un tipo de datos cadena de C como argumento, declare el argumento como una cadena con la palabra clave **ByVal**.

Cuando se usa con un argumento de cadena, **ByVal** especifica que VBA debe pasar una cadena como las cadenas de C que terminan con el carácter **Null**. VBA pasa la dirección de la cadena terminada en **Null** (esto es, pasa la cadena terminada en **Null** por referencia).

Cadenas variables o uso de cadenas de longitud fijas Debido a que las cadenas pasadas a los procedimientos DLL son pasados por referencia, un procedimiento DLL puede modificar la variable de cadena de VBA que recibe como argumento.

Sin embargo, el DLL no puede incrementar el tipo de datos cadena de VBA, y puede escribir mas allá del final de la cadena si este no es lo suficientemente largo. Esto corrompe otras áreas de memoria. Para evitar un error, haga el argumento de cadena suficientemente largo para que el procedimiento DLL nunca escriba pasando el final de este.

Pare asegurar que un argumento de cadena es siempre suficientemente largo para almacenar una cadena de C, use uno de los dos siguientes métodos:

- Llenar la cadena de caracteres con al menos 255 caracteres de longitud con caracteres **Null**

```
Dim FilePath As String
FilePath = String$(255,0)
```

-- o --

- defina las cadenas como cadenas de longitud fija con al menos 255 caracteres de longitud.

```
Dim FilePath As String * 255
```

El primer método es preferible si de planea usar la cadena posteriormente, ya sea reduciéndola o aumentándola. No puede modificar la longitud de cadenas de longitud fija

Algunos procedimientos aceptan además un entero que indica la longitud de la cadena proporcionada.

Devolución de cadenas conteniendo un Null Las cadenas devueltas por un procedimiento DLL típicamente contiene el carácter **Null** al final de esta. Generalmente, los procedimientos además regresan un entero indicando la longitud de la cadena devuelta. Puede remover el carácter **Null** mediante la función Left$ o mediante la búsqueda del **Null** explícitamente. Es necesario excluir el **Null** si se están combinando varias cadenas.

Ejemplo de paso de cadenas.

El procedimiento GetWindowsDirectoryA devuelve el directorio de Windows. Se proporciona una cadena vacía que es modificada para reflejar la ruta del directorio de Windows.

Debe procesarse la cadena para que sea lo suficientemente largo para aceptar el nombre de directorio mas largo posible.

Además se pasa al procedimiento un argumento entero que indica la longitud de la variable de cadena cuando es pasada.

El valor devuelto por el procedimiento es un entero que indica el numero de caracteres en la cadena modificada. Esta cadena modificada contiene un como ultimo caracter un **Null**. Se puede remover este carácter mediante búsqueda con las funciones Mid$ o Left$.

```
Private Declare Function GetWIndowsDirectory Lib "kernel32" Alias _
"GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize AS Long) As long

Sub Command1_Click()
      Dim WinDir As String, sLen As Integer

      WinDir = String$(255,0)
      SLen = GetWindowsDirectory(WinDir, Len(WinDir))
      WinDir = Left(WinDir, sLen)
      WinDir = WinDir + "\"
      MsgBox "Windows esta en " - WinDir
End Sub
```

8.4 Ejemplo de llamadas a DLL.

Determinar cuando una forma esta maximizada La función IsIconic puede ser empleada para revisar el estado de una ventana. Esta función se encuentra en el DLL user32, parte de WIN32API. Puede pegar la declaración de la función dentro de su código usando el Visor de API Winn32.

Este ejemplo muestra el llamado a la función IsIconic para determinar si Form1 esta en un estado minimizado o maximizado.

```
Private Declare Functin IsIconic Lib"user32" (ByVal hWnd as Long) As Long

Sub Button0_Click()
      If IsIconic(forms!form.hwnd) <> 0 then
            MsgBox "La forma esta minimizada"
      Else
            MsgBox "La forma no esta minimizada"
      End If
End Sub
```

8.5 Preguntas de repaso.

1. ¿Cuál de los siguientes argumentos acerca de DLL's es cierto?

   a. Son procedimientos que residen en módulos en la base de datos.

   b. Contienen procedimientos que residen en archivos fuera de la base de datos y están estáticamente vinculados con la base de datos.

   c. Contienen procedimientos que residen en archivos fuera de la base de datos y están dinámicamente vinculados con la base de datos.

   d. Los procedimientos en DLL's corren mas lento que los procedimientos escritos en VBA.

2. Todas las siguientes son buenas razones para usar DLL's en lugar de escribir cualquier cosa en VBA, excepto:

   a. Usar DLL completa tareas no disponibles en VBA.

   b. Los DLL's simplifican la tarea de escribir código.

   c. Usar DLL's mejora el desempeño.

   d. Usar DLL's permite actualizar el código independientemente de la aplicación de Microsoft Access.

3. Puede usar todos los DLL's siguientes excepto.

   a. DLL's de terceros.

   b. DLL's de Windows (Windows API).

   c. DLL's creados con lenguaje C.

   d DLL's creados con VBA en Microsoft Access.

4. ¿Que esta mal en la instrucción de declaración mostrada?

```
Private Declre Function _lopen Lib "kernel32" (ByVal lpPathName AS
String,ByVal iREadWrite As Long) As Long
```

5. La función FindWindow del DLL de Windows es usada para encontrar el manejador de una aplicación ejecutándose Para encontrar el manejador de la Calculadora, puede pasar un **Null** como primer argumento y "Calculator" como segundo. ¿Qué código llamará correctamente a la función FindWindow?

```
Private Declre Function FindWindow "user32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpCaption As String) As Long
```

6. Cuando pasa argumentos de cadena a un DLL que espera cadenas tipo C, ¿como puede declarar el argumento de cadena? (sArg es el argumento)

7. Cual de los siguientes códigos de ejemplo crea una cadena que es suficientemente larga para almacenar 255 caracteres devueltos por un DLL.

   a FilePath = String$(255,0).

   b. Dim FilePath As String * 255

   c. Dim File Path AS String

   d. Tanto A como B.

8. En el siguiente código de ejemplo. ¿cuál será el valor de la variable sLen después de que la función GetWindowsDirectory es ejecutada?

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias
"GetwindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long

Dim WinDir As String * 255
Dim sLen As Integer
SLen = GetWindowsDirectoryA(WnDir, Len(WinDir))
```

9. Cuando se pasa una cadena por valor, ¿qué es lo que realmente se esta pasando?

10. ¿Cuál es la diferencia entre pasar un argumento **ByVal** y **ByRef**?

Capítulo 9: Replicación.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Definir el concepto de replicación de bases de datos y comparar situaciones en las cuales su uso es apropiado e inapropiado.
- ☑ Explicar como el Maletín de Windows 95 se usa para replicación.
- ☑ Usar el administrador de replicación para:
  - ☑ Convertir una base de datos a Diseño Maestro.
  - ☑ Crear replicas adicionales.
- ☑ Identificar los cambios que el motor de base de datos Jet realiza cuando una base de datos es convertida a una base de datos replicable.
- ☑ Crear y sincronizar replicas por medio de código.

9.1 Introducción a la replicación de bases de datos.

¿Qué es la replicación de bases de datos?

Con Microsoft Access se tiene la capacidad de sincronizar múltiples copias de una base de datos a través de la *replicación de bases de datos*.

La replicación de bases de datos es una tecnología que permite hacer copias de una base de datos asegurado que los cambios en el diseño y los datos con propagados a través de las copias.

Con la replicación de bases de datos, la base de datos original es convertida a un *Diseño Maestro*. En este punto, el conjunto de replicas contiene un solo miembro, el Diseño Maestro. Desde este Diseño Maestro pueden crearse copias adicionales llamadas *replicas* El conjunto de *Diseño Maestro* y *Replicas* conforman el conjunto de *Replica*.

El Diseño Maestro es una base de datos a la cual han sido agregadas tablas y campos de sistema, así como propiedades de replicación. El diseño maestro tiene los siguientes atributos:

- Pueden realizarse cambios a la estructura de la base de datos solo con el Diseño Maestro.
- Un Diseño Maestro es el primer elemento en un conjunto de replicas.
- Se usa para crear replicas adicionales al conjunto.

Una replica es creada desde el Diseño Maestro y es un miembro del conjunto de replicas. Al igual que el Diseño Maestro, contiene tablas y campos de sistema. así como propiedades de replica. Los cambios en el diseño realizados a los objetos replicables e el Diseño Maestro con replicados desde este hacia todas las replicas en el conjunto Los cambios en los datos son replicados a través de todas la replicas y el Diseño Maestro.

Tanto el Diseño Maestro como todas las replicas comparten el mismo diseño y un identificador único del conjunto de replicas.

¿Cuándo usar la replicación de base de datos?

La replicación de bases de datos trabaja bien para ciertas aplicaciones de negocios Sin embargo, existen situaciones en las cuales la replicación no es la solución ideal.

La replicación de bases de datos se recomienda para los siguientes escenarios:

Compartir datos entre oficinas Se usa la replicación para crear copias de una base de datos corporativa que serán enviadas a cada oficina regional.

Compartir datos entre usuarios dispersos Un escenario típico ocurre cuando información nueva es introducida a la base de datos en la oficina corporativa. Los empleados en sitios remotos pueden sincronizar la base de datos mediante una conexión electrónica con la red corporativa.

Proporcionar un fácil acceso a los datos del servidor Si la aplicación no requiere tener actualizaciones inmediatas de datos, se puede usar la replicación para reducir la carga de la red en el servidor primario. Introducir un segundo servidor con su propia copia de la base de datos mejora el

tiempo de respuesta. Puede determinarse la programación de la sincronización de las replicas y ajustar esa programación para cumplir las necesidades cambiantes de los usuarios.

Distribuir actualizaciones de la aplicación Cuando se replica una aplicación, automáticamente se replican tanto los datos en las tablas, como los objetos de la aplicación. Si se realizan cambios al diseño de la base de datos, los cambios son transmitidos durante la siguiente sincronización; no es necesario distribuir una copia completa del las nuevas versiones del software.

Respaldo de datos La replicación de bases de datos puede verse como una simple copia. Sin embargo, la replicación hace una copia completa de la base de datos, y simplemente sincroniza los objetos de la replica en intervalos regulares Los usuarios pueden usar esta copia pare recuperar datos en el caso de que se pierdan los datos de la base original.

La replicación no es una solución ideal para las siguientes situaciones:

Actualizaciones masivas en múltiples replicas En aplicaciones que requieran de actualizaciones frecuentes de registros existentes a través de diferentes replicas, se pueden encontrar mas conflictos de registros que los que se presentarían en aplicaciones donde se insertan nuevos registros. La resolución de conflictos es un proceso manual de una labor intensiva.

La consistencia de datos es critica Las aplicaciones que dependen del procesamiento de transacciones, tales como transferencia de fondos, necesitan ser corregidos muchas veces. Por lo tanto, la sincronización de datos a diferentes momentos no garantiza la consistencia del dato en tiempo real y su precisión.

Las bases de datos replicables usan el mismo modelo que las bases de datos ordinarias – el modelo de seguridad Microsoft Jet. Cuando se implementa la seguridad para una base de datos replicable, debe tenerse cuidado en lo siguiente:

- Los usuarios deben tener permisos de Administrador en la base de datos.
- Las bases de datos con seguridad a nivel de compartición no son replicables.

Terminología de replicación.

Existen algunos términos con los que se debe estar familiarizado para tener un entendimiento básico de la replicación de bases de datos implementada por Microsoft Access.

Diseño Maestro El Diseño Maestro es una base de datos a la cual se han agregado tablas del sistema, campos del sistema y propiedades de replicación. Cuando trabaja con un Diseño Maestro, debe adherirse a los siguientes principios

- El Diseño Maestro es el primer miembro del conjunto de replicas.
- Se usa el Diseño Maestro para crear la primera replica en el conjunto de replicas.
- Solo se pueden hacer cambios a la estructura de la base de datos a través del Diseño Maestro.
- Solo puede tener un Diseño Maestro a la vez en un conjunto de replicas.

Sin embargo, las replicas en un conjunto de replicas pueden ser Diseño Maestro.

Replica una replica es una copia de la base de datos, incluyendo tablas, consultas, formas, reportes, macros y módulos. Una replicas es un miembro del conjunto de replicas. Puede sincronizar una replica con otras replicas del conjunto Cuando se modifican los datos en una tabla replicada, esas modificaciones son propagadas a otras replicas y al Diseño Maestro.

Conjunto de replicas Un conjunto de replicas comprende al Diseño Maestro y todas las replicas que comparten el mismo diseño de la base de datos y el identificador único del conjunto de replicas.

Objeto Global Un objeto global es cualquier objeto en el Diseño Maestro cuya propiedad replicable esta establecida a TRUE, y es replicado a través de conjunto de replicas.

GUID Un identificador global único (GUID) es un campo de 16 bits usado para establecer un identificador único para la replicación. Los GUID's identifican replicas, conjuntos de replicas, tablas, registros y otros objetos. Microsoft Access se refiere al GUID como ID de replicación.

Objeto Local Un objeto local se refiere a una tabla, consulta, formulario, reporte, macro o modulo que permanece en la replica o Diseño Maestro donde fue creado.

Ni el objeto ni los cambios realizados al objeto son copiados a otros miembros del conjunto de replicas.

Replica ID Un ID de replica es un identificador único para replicas en un conjunto de replicas.

9.2 Replicación a través de la interface del usuario.

Replicación del Maletín bajo Windows 95.

Microsoft Windows 95 permite tomar ventaja de la replicación de bases de datos a través del maletín.

Por ejemplo, un usuario puede arrastrar un archivo de base de datos Microsoft Jet desde el servidor de una red hasta Mi Maletín en una computadora portátil. El usuario trabaja sobre su computadora portátil y se reconecta al servidor de la red. En este punto, el usuario puede sincronizar las dos copias de la base de datos empleando las opciones disponibles del Maletín.

▶ **Para usar la replicación del maletín bajo Windows 95.**

1. Abra el maletín de Windows 95
2. Cree en Microsoft Access una base de datos nueva.
3. Importe dos o tres tablas de la base de datos Neptuno a la base creada.
4. Arrastre la base de datos hacia el Maletín. Un mensaje informara acerca de los cambios de incremento en el tamaño de la base de datos
5. Diga que si en la ventana mostrada. En este punto, se pide elegir entre la base de datos original o la copia en el maletín para crear el Diseño Maestro.
6. Elija la base de datos original. Se vera el Diseño Maestro y la replica en la ventana del Maletín.

▶ **Para ver como trabaja la repiicación del Maletín.**

1. abra el Diseño Maestro y agregue algunos registros a una tabla.
2. Del menú Maletín, elija Actualizar todo. Se abrirá la ventana de actualización del Maletín.
3. Elija Actualizar
4. Abra la base de datos y abra la tabla para visualizar los cambios.

9.3 Conflictos y errores de la replicación.

Errores de diseño.

Los errores de diseño en una base de datos se presentan cuando se modifica un diseño en el Diseño Maestro y entra en conflictos con un cambio de diseño en una replica.

Proceso de grabar cambios de diseño

A medida que se van generando cambios en el diseño de una base de datos, Microsoft Jet graba cada cambio en la tabla de sistema MsysSchChange. Cada registro contiene información acerca del cambio realizado. Cuando Microsoft Jet sincroniza los dos miembros del conjunto de replica, compara la tabla MsysSchChange del miembro que inicia con la correspondiente del miembro objetivo.

Cuando Microsoft Jet aplica todos los cambios de diseño de un miembro del conjunto de replicas a otro miembro. estos se aplican en el orden en que fueron hechos en el Diseño Maestro, y, consecuentemente, el orden en el cual los cambios fueron grabados en la tabla MsysSchChange.

Por ejemplo, para agregar una tabla a la base de datos, y entonces hacer esta tabla replicable, Microsoft Jet toma las siguientes acciones.

- Registra la creación de la tabla como un cambio en el diseño.
- Hace la tabla replicable, el segundo cambio en el diseño.
- Si se elimina la tabla de replicación, Microsoft Jet registra esto como un tercer cambio en el diseño.

En la siguiente sincronización Jet aplica cada cambio en ese orden. aun cuando la tabla no exista cuando la sincronización se complete.

Como ocurren los errores.

Sin embargo, los cambios realizados a una replica pueden causar errores El siguiente ejemplo simple explica como puede ocurrir un error de diseño.

1. Un usuario crea una tabla local en una replica, y le da el nombre predeterminado Tabla1.
2. Otro usuario crea una tabla replicable en el Diseño Maestro y también le da el nombre predeterminado Tabla1.

Microsoft Jet fallará cuando intente sincronizar la replica con el Diseño Maestro. La replica ya tiene una tabla con ese nombre.

La tabla MsysSchemaProb registra los errores que ocurren cuando la sincronización falla entre una replica y el Diseño Maestro.

Conflictos de sincronización.

Los conflictos de sincronización ocurren cuando Microsoft Jet detecta que el mismo registro ha cambiado en ambos miembros de un conjunto de replicas.

Aun si los cambios hechos a un miembro del conjunto de replicas sea en diferentes campos que los hechos al otro miembro, Microsoft Jet los trata como conflicto de sincronización.

Papel de Microsoft Jet en conflictos de sincronización.

Cuando se presenta un conflicto de sincronización, Microsoft Jet no intenta resolver dicho conflicto basado en el contexto de los registros o los cambios hechos a los datos.

Microsoft Jet usa un algoritmo simple para seleccionar una de las versiones de registros y tomarlo como el cambio oficial, y escribe los datos desde la otra versión del registro dentro de la tabla conflicto. Los nombres de las tablas en conflicto toman la forma *table*_Conflict Es entonces la responsabilidad del usuario el revisar cada conflicto para determinar que la información correcta sea aplicada a la base de datos.

Proceso de solución.

Desde la apertura de una tabla que contiene una tabla de conflicto, se le muestra al usuario la ventana de Solución de conflictos de Replicación. El usuario puede ver el registro que esta causando el conflicto y toma las acciones apropiadas.

Errores de sincronización

Existen 4 fuentes potenciales de errores de sincronización, lo cuales deben ser corregidos tan pronto como sea posible para asegurar que los datos en replicas diferentes no se dañen.

La validación a nivel de tablas (TLV) permite restringir el valor o tipo de datos introducidos a una tabla Sin embargo, si intenta sincronizar una replica existente conteniendo registros que no satisfacen la TLV en el Diseño Maestro, recibirá un error Si el valor falla, la actualización falla y un error es escrito en la tabla MsysErrors en el miembro receptor

Las llaves duplicadas pueden ocurrir cuando:

- Dos usuarios de diferentes replicas insertan simultáneamente un registro nuevo y usan la misma llave primaria para sus respectivos registros.
- Dos usuarios en diferentes replicas cambian la llave en dos diferentes registros cuando ambos tratan de usar el mismo valor

Cuando la sincronización de replicas ocurre, la sincronización tiene éxito o Microsoft Jet escribe un error de llave duplicada a una tabla MsysErrors para cada uno de los registros que fallaron al insertarse o actualizarse

Cuando se refuerza la integridad referencial, se previene a los usuarios de agregar o eliminar un registro de una tabla si no existe el correspondiente registro en la tabla primaria La integridad referencial puede además ser una fuente de errores de sincronización.

El siguiente ejemplo describe una situación en la cual la integridad referencial puede originar errores de sincronización.

1. El usuario A elimina un registro con una llave primaria desde el Diseño Maestro.

2. El usuario B inserta un registro dentro de una replica que referencia a la llave primaria en el Diseño Maestro. La siguiente sincronización de los dos miembros generará errores.

La sincronización entre miembros del conjunto de replicas puede ocasionalmente fallar debido a conflictos regulares de bloqueos multiusuarios. Si un registro permanece bloqueado después de intentos repetidos, la sincronización falla y Microsoft Jet registra un error en la tabla de sistema MsysErrors.

Puede ignorar los errores causados por el bloqueo de registros debido a que Microsoft Jet reintentara actualizar los registros durante la siguiente sincronización. Es extremadamente inusual que los mismos registros tengan un bloqueo durante la siguiente sincronización.

Nota Asegúrese que la base de datos no este abierta de modo exclusivo, debido a que esto causara errores de sincronización.

9.4 Replicación usando DAO.

Haciendo replicable una base de datos.

Para convertir una base de datos a una base de datos replicable mediante código, debe crearse y agregarse la propiedad Replicable al objeto Database.

El proceso de crear la propiedad Replicable es similar a crear propiedades definidas por el usuario mediante DAO

Después de haber creado la propiedad Replicable y agregarla al objeto Database, es necesario establecer la propiedad DesignMasterID igual a la propiedad ReplicaID. En este punto, , la base de datos es convertida en un Diseño Maestro, como se muestra en el ejemplo

Importante Asegúrese de hacer una copia de respaldo antes de ejecutar el siguiente código

```
Sub ConvertToReplica()

     Dim dbs As Database

     Set dbs = DBEngine.Worksopaces(0).OpenDatabase_
     ("c:\replication\trade2.mdb", True)
     Dbs.Properties.Append dbs.CreateProperty("Replicable", dbText, "T")
     Dbs.DesignMasterID = dbs.ReplicaID

End Sub
```

Cuando se convierte una base de datos mediante el establecimiento de la propiedad Replicable a True, solo se tiene un miembro en el conjunto de replicas (el Diseño Maestro), y se hace la primera replica a partir de el La primera replica y las replicas subsecuentes se realizan usando el método MakeReplica El siguiente ejemplo ilustra como hacer una replica

```
Sub MakeAdditionalReplica(strReplicableDB, strNewReplica)
     Dim dbs As Database, ws As Workspace

     Set ws = DBEngine(0)
     Set dbs = ws.OpenDatabase(strReplicableDB)
     dbs.MakeReplica strNewReplica, strReplicableDB, dbRepMakeReadOnly
     dbs.Close

End Sub
```

Sincronización de replicas.

Así como se pueden sincronizar replicas mediante el Administrador de Replicas y el Maletín, también pueden sincronizarse mediante código. Puede necesitarse una implementación de sincronización mediante código cuando se esta distribuyendo la aplicación a usuarios menos sofisticados

Se pueden mantener a los miembros de un conjunto de replicas sincronizados mediante el uso del método Syncronize. Mediante la especificación del nombre de la base de datos objetivo, puede sincronizar una replica de usuario con otro miembro del conjunto Puede además realizar intercambios en uno o dos sentidos.

Por ejemplo, puede usar el siguiente código para realizar un intercambio de dos sentidos entre miembros.

```
Sub SyncronizableDBs(strDBName, strSyncTargetDB)
      Dim dbs As Database, ws As Workspace

      Set ws = DBENgine(0)
      Set dbs = ws.OpenDatabase(strDBName)
      Dbs.Syncronize strSyncTargetDB, dbRepImpExpChanges
      Dbs.Close
End Sub
```

Resolución de conflictos y errores de replicación.

Posterior a la sincronización de replicas, debe revisar cada conflicto para determinar que la información correcta se ha aplicado a la base de datos.

Para determinar si se ha generado un conflicto para una tabla especifica, se usa la propiedad ConflictTable. Esta propiedad devuelve el nombre de la tabla conflicto conteniendo los registros de la base de datos conflictivos durante la sintonización.

El siguiente ejemplo encuentra el nombre de una tabla conflicto, examina cada registro conflicto y toma acciones para resolver cada conflicto.

```
Sub Resolve(dbs As Database)
      Dim tdfTest As TableDef, rstConflict As Recordset

      For each tdfText In dbs.TableDefs
            If (tdfTest.ConflictTable <> "") Then
                  Set rstConflict = dbs.OpenRecordset(tdfTest.ConfflictTable)
                  Rst.Conflict.MoveNext
                  While Not rstCOnflict.BOF
                        RstCOnflict.Delete
                        RstConflict.MoveNext
                  Wend
                  Rst.Conflict.Close
            End If
      Next tdfTest
End Sub
```

Si no hay tablas conflicto, o si la base de datos no puede ser replicable, la propiedad devuelve una cadena de longitud cero

Conforme se revisa cada conflicto en la tabla de conflictos, deben tomarse acciones apropiadas. Si la versión del registro seleccionada por Microsoft Jet es la versión correcta y no son necesarias acciones futuras, puede eliminar el registro de la tabla conflicto. Si la versión seleccionada por Microsoft Jet no es la versión correcta, puede hacerse:

- Manualmente introducir los datos desde la tabla conflicto en la base de datos.

- Desarrollar una rutina personalizada de solución de conflictos que siempre asigne una prioridad mayor a cambios en una replica especifica sobre otras replicas.

- Considerar en que momento los cambios procedurales relacionados con como se introducen los datos o se modifican son necesarios.

9.5 Preguntas de repaso.

1. la replicación de bases de datos se emplea para:
   a. reparar una base de datos cuando esta ha sido reportada como corrupta.
   b. Mejorar el desempeño de la base de datos mediante la remoción de espacio en blanco en el archivo.
   c. Hacer actualizaciones masivas a múltiples registros en una base de datos simple.
   d. Sincronizar los datos y la estructura de copias múltiples de la base de datos.

2. ¿Cuál es la única base de datos donde se pueden hacer cambios a la estructura de la base?

3. ¿Cuáles son los objetos que no se llaman replicados?

4. ¿Cómo se hace una base de datos replicable?
   a. Usando el maletín.
   b. Usando objetos de acceso a datos.
   c Usando los elementos del menú Replicación del menú Herramientas en Microsoft Access
   d. Todos los anteriores.

5. Cuando se hace una base de datos replicable, todo lo siguiente ocurre excepto:
   a. Nuevos campos son agregados a cada tabla replicada.
   b. Nuevas tablas son agregadas a la base de datos.
   c. La base de datos es encriptada para agregar seguridad.
   d. Nuevas propiedades son agregadas a la base de datos.

6. ¿Cuál es el termino para cuando Microsoft Jet detecta que un mismo registro ha cambiado en dos miembros del mismo conjunto de replicas?

7. ¿Cómo se resuelven los conflictos de sincronización?

8. ¿En qué circunstancias usaría replicación DAO?

9. Empleando DAO, ¿cómo determina si ocurrió un conflicto de sincronización?

10. ¿Qué hace el siguiente código?

```
Dim db As Database
Set db = CurrentDB
Db.Suyncronize "c:\MyRepica.mdb", dbRepExportChanges
```

Capítulo 10: Seguridad.

Objetivos

Al finalizar el presente capitulo, el participante será capaz de:

- ☑ Describir el modelo de seguridad Microsoft Jet.
- ☑ Diferenciar entre seguridad a nivel usuario y a nivel comparición.
- ☑ Listar los pasos para implementar seguridad.
- ☑ Crear un archivo de información de grupos de trabajo y un nuevo usuario Admin.
- ☑ Emplear el asistente de seguridad a nivel usuario para asegurar una base de datos.
- ☑ Determinar accesos a la aplicación y niveles de acceso mediante la adición de usuarios y grupos.
- ☑ Describir la función de la encriptación de bases de datos.
- ☑ Establecer opciones de seguridad programáticamente.

10.1 ¿Por qué asegurar bases de datos?.

Existen diferentes razones para implementar seguridad en una aplicación. El nivel y complejidad de la seguridad dependerá de los requerimientos del sistema y el ambiente del usuario.

Proteger datos sensibles

La aplicación debe tener medios para prevenir accesos no autorizados a las tablas. Las organizaciones típicamente usan sistemas sonde diferentes grupos de personas requieren tener diferentes niveles de acceso.

Por ejemplo, desea implementar un sistema donde los siguientes grupos tengan los siguientes niveles de acceso a la tabla conteniendo información de los salarios.

- Administradores – pueden actualizar la tabla.
- Personal de nominas – pueden ver mas no actualizar la tabla.
- Todos los demás – no pueden ver la tabla.

Protección de la propiedad intelectual

Debe asegurar de salvaguardar aquellos aspectos de una aplicación que se consideren como propiedad intelectual. Típicamente, esto involucra la estructura de la aplicación, definiciones de objetos y el código fuente de la aplicación. Asegurando una aplicación se previene de la perdida de la propiedad por individuos no autorizados.

Prevenir cambios inadvertidos al diseño de la aplicación

La tercera razón por la cual implementar seguridad en una aplicacion es la necesidad de prevenir que los usuarios inadvertidamente estropeen las aplicaciones mediante modificaciones al código u objetos de los cuales depende la aplicación.

10.2 El modelo de seguridad de Microsoft Jet.

Introducción al modelo de seguridad de Microsoft Jet.

El modelo de seguridad Microsoft Jet proporciona un gran control sobre el acceso de los usuarios a la aplicación.

Modelo robusto y sofisticado El modelo de seguridad Microsoft Jet difiere de otros modelos de seguridad para bases de datos en que proporciona seguridad a nivel de usuario, en contrataste con la seguridad a nivel de compartición. Esto hace un sistema robusto, mientras proporciona un buen nivel de flexibilidad.

Sin embargo, a veces los usuarios encuentran problemas cuando implementan seguridad debido la falta de entendimiento de cómo funciona la seguridad Jet. Un firme conocimiento en la seguridad Jet es importante para poder implementar apropiadamente la segundad en las aplicaciones Microsoft Access

La siguiente lista proporciona ejemplos de concepciones erróneas o limitadas acerca de la seguridad.

- Los usuarios tratan de implementar seguridad sin un total entendimiento de ella, asociado a una falta de familiaridad con la documentación.
- El usuario es incapaz de completar todos los pasos requeridos para asegurar una base de datos debido a una falta de conocimiento de la documentación.
- El usuario no desarrolla los pasos requeridos para la seguridad en el orden correcto.
- El usuario no entiende completamente las diferencias entre identificador personal (PID) e identificador de seguridad (SID).
- El usuario no entiende el papel de la base de datos de grupos de trabajo, la base de datos de usuarios y el motor de base de datos Microsoft Jet.

Seguridad a nivel usuario Teniendo un sistema de seguridad basado en seguridad a nivel usuario proporciona una gran flexibilidad. La seguridad a nivel usuario significa que los usuarios son autentificados cuando inician Microsoft Access mediante un nombre de usuario y una contraseña. Los administradores garantizan permisos específicos tales como leer datos o modificar el diseño para usuarios específicos y grupos sobre objetos específicos (tablas, formularios, reportes, módulos, y otros) Diferentes usuarios pueden tener diferentes permisos para los mismos objetos.

Seguridad a nivel compartición En contraste, los sistemas que implementan seguridad a nivel compartición asocian contraseñas con objetos específicos y usualmente no se requiere que el usuario sea autentificado con una contraseña En estos sistemas, cualquier usuario que conozca la contraseña de un objeto especifico puede accesarlo.

Ventajas de la seguridad a nivel usuario La seguridad a nivel usuario proporciona una gran flexibilidad y control en la asignación de permisos a varios niveles de usuarios. El usuario administra sus propias contraseñas Microsoft Access emplea la seguridad a nivel usuario.

Por ejemplo, suponga que se tienen dos grupos, administradores y trabajadores, los cuales requieres diferentes niveles de seguridad para una tabla conteniendo información salarial. En este escenario, un sistema a nivel de usuario, tal como el creado por Microsoft Jet permitiría:

- Asignar permisos de actualización para los administradores.
- Asignar permisos de lectura para los trabajadores.
- Listar cada usuario en el sistema dentro de cada uno de los dos grupos.
- Proporcionar a los usuarios la libertad de administrar sus propias contraseñas.

Las contraseñas verifican la identidad del usuario, en vez de identificar un permiso para un objeto.

Vista conceptual del modelo de seguridad de Microsoft Jet.

Para entender como trabaja el modelo de seguridad de Microsoft Access, es necesario estar familiarizado con todos los componentes del modelo de seguridad Microsoft Jet.

Microsoft Jet emplea un grupo de elementos que trabajan juntos para constituir el modelo. A un nivel conceptual, existen varios elementos clave:

- Usuario – el usuario de la aplicación.
- Grupo – Colección lógica de usuarios con requerimientos iguales a un conjunto de objetos.
- Base de datos de grupo de trabajo – Contiene usuarios, grupos y sus contraseñas.
- Base de datos de usuario – Contiene objetos y permisos para accesar esos objetos.
- Motor Jet – Revisa permisos sobre objetos para un usuario particular.

Nota Después del proceso de conexión (log-on) el nombre de usuario y contraseña no son empleados para futuras solicitudes de acceso a un objeto Microsoft Jet usa el identificador de seguridad (SID) para recuperar la identidad de un usuario

---

La base de datos de grupo de trabajo.

No toda la información de seguridad esta almacenada en la base de datos. Alguna de ella s almacena en una base de datos de Microsoft Jet conocida como *base de datos de grupo de trabajo.*

Definición de base de datos de grupo de trabajo Esta base de datos es un base de datos de Microsoft Jet que trabaja junto con la base de datos del usuario en una aplicación Microsoft Access. Contiene varios tablas del sistema que el motor emplea par almacenar información de seguridad

La base de datos de grupo de trabajo tiene estas características esenciales

- El nombre predeterminado del grupo de trabajo es SYSTEM.MDW
- La base de datos de grupos de trabajo almacenan objetos de usuario y de grupo. Estos incluyen registros para cada usuario y grupo de trabajo, que usuarios pertenecen a que grupo, contraseñas encriptadas y el identificador de seguridad (SID) para cada usuario y grupo.

Usuarios y Grupos Microsoft Jet define usuarios de una aplicación de dos formas: como usuarios individuales o como grupos de usuarios. Microsoft Jet almacena información sobre usuarios y grupos en SYSTEM.MDW usando los siguientes mecanismos:

- Identificador de seguridad (SID).

Microsoft Jet genera un numero especial conocido como identificador de seguridad (SID) cuando un usuario se registra en el sistema. Cuando un usuario intenta accesar un objeto, Microsoft Jet usa el SID para identificar al usuario o grupo y determinar el acceso del usuario a los objetos.

- Identificador Personal (PID)

Microsoft Jet emplea un identificador personal (PID) en conjunción con un programa de encriptación para crear el SID para una cuenta. Esto permite recrear cuentas de usuario si la base de datos del sistema se corrompe o se pierde

Usuarios y grupos predeterminados No es necesario crear o administrar grupos para necesidades de seguridad que no los requieran. Microsoft Jet define dos usuario y un grupo predeterminados que pueden cubrir muchos casos

La base de datos del usuario.

La base de datos del usuario almacena objetos que son creados y los permisos para esos objetos.

Los permisos reales relacionados a objetos específicos para usuarios y grupos específicos están almacenados en tablas del sistema dentro de la base de datos. Sin embargo, ningún usuario, grupo o contraseña se almacena ahí Tal información esta almacenada en la base de datos del grupo de trabajo.

Para verificar si un usuario posee permisos suficientes para accesar un objeto:

1. El usuario solicita accesar a un objeto.

   Microsoft Jet ya ha recuperado el SID del usuario de la base de datos del grupo de trabajo durante el registro de este.

2. Microsoft Jet compara el SID con el SID en las tablas de sistema de la base de datos.

10.3 Asegurando bases de datos.

Asegurar una base de datos es un proceso de varios pasos que cambia el comportamiento predeterminado de usuarios y grupos mediante el reemplazo de cuentas incorporadas con una versión segura propia. Esta sección cubre cada uno de los pasos en el proceso

Creación de una base de datos para Grupos de trabajo.

Para asegurar una base de datos, primero debe crear una base de datos de grupos de trabajo, para almacenar información del usuario, grupo y contraseña Esto creara un SID único para el grupo Admins.

Usando el administrador de Grupos de Trabajo (WRKGADM.EXE), siga estos pasos:

1. Inicie el administrador de grupos de trabajo.
2. Elija crear un nuevo grupo de trabajo.
3. Especifique un nombre de usuario. una organización y un identificador de grupo de trabajo único y elija Aceptar
4. Especifique el nombre y ruta para la nueva base de datos del grupo de trabajo y elija Aceptar. Las bases de datos de grupos de trabajo tienen la extensión predeterminada .MDW.

Importante Asegúrese de conservar esta información en un lugar seguro Si un grupo de trabajo nuevo se corrompe o se pierde, es necesaria esta información para recrear la base de dato del grupo de trabajo.

5. Elija Aceptar para crear la base de datos.

Creación de un nuevo usuario Admin.

El usuario Admin es el usuario creado predeterminadamente por el Microsoft Jet en cada grupo de trabajo. Debido a que el usuario Admin es el mismo en todas las bases de datos de grupos de trabajo, es deseable asegurar que este usuario posea y mantenga todos los objetos seguros

La forma mas fácil de crear un nuevo usuario Admin es empleando Microsoft Access.

1 Regístrese como el nuevo Administrador del sistema
2. Elimine Admin del grupo de Admins.
3. En el menú Herramientas. seleccione Seguridad y dentro elija Cuentas de usuarios y grupos.
4. Bajo Usuario. elija Nuevo.
5. En la ventana de usuario/grupo nuevo. escriba el nombre y un ID personal para el nuevo usuario Admin, y de click en Aceptar
6. El nuevo usuario debe ser miembro de Admins. Si Microsoft Access no muestra la palabra Admins bajo Miembro de cuando el nuevo usuario Admin esta seleccionado, siga estos pasos:

7 Seleccione el nuevo usuario Admins bajo Grupos disponibles, Admins.

8. De click en Agregar.

Es importante recordar además que debe de:

- Asignar una contraseña al nuevo usuario Admin.
- Escribir y almacenar en lugar seguro esta información.

10.4 Eliminando seguridad de una base de datos.

Usualmente no hay necesidad de eliminar la seguridad de una base de datos. Sin embargo, puede necesitarse trabajar con una base de datos no asegurada o una copia de ella si hay elementos de desarrollo que así lo requieran

Proceso paso a paso De manera similar al proceso para asegurar una base de datos, eliminar la seguridad de una base de datos es un proceso que requiere de varios pasos.

1. Entre a Microsoft Access registrándose como un miembro del grupo Admins.
2. Asigne permisos completos al grupo Usuarios para todos los objetos de la base de datos.
3. Elimine la contraseña para el administrador Admin.
4. Reinicie Microsoft Access y regístrese como Admin.
5. Cree una nueva base de datos e importe todos los objetos desde la base de datos segura.