23
24.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ARAGON

"GUIA DE ACCESO A BASES DE DATOS CON VISUAL BASIC".

T E S I S

QUE PARA OBTENER EL TITULO DE :

INGENIERO EN COMPUTACION

P R E S E N T A

HECTOR GARCIA RIVERA

ASESOR DE TESIS ERNESTO PEÑALOZA ROMERO

MEXICO **1997**

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
CAMPUS ARAGÓN

UNIDAD ACADEMICA

Ing. JUAN GASTALDI PÉREZ
Jefe de la Carrera de Ingeniería en Computación,
P r e s e n t e .

En atención a la solicitud de fecha 5 de marzo del año en curso, por la que se comunica que el alumno HÉCTOR GARCÍA RIVERA, de la carrera de Ingeniero en Computación, ha concluido su trabajo de investigación intitulado "GUÍA DE ACCESO A BASES DE DATOS CON VISUAL BASIC", y como el mismo ha sido revisado y aprobado por usted, se autoriza su impresión; así como la iniciación de los trámites correspondientes para la celebración del Examen Profesional.

Sin otro particular, le reitero las seguridades de mi atenta consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
San Juan de Aragón, México, 6 de marzo de 1997
EL JEFE DE LA UNIDAD

Lic. ALBERTO IBARRA ROSAS

c c p Asesor de Tesis.
c c p Interesado.

AIR'lla.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

HÉCTOR GARCÍA RIVERA
P R E S E N T E .

En contestación a su solicitud de fecha 24 de octubre del año en curso, relativa a la autorización que se le debe conceder para que el señor profesor, Ing. ERNESTO PEÑALOZA ROMERO pueda dirigirle el trabajo de Tesis denominado, "GUÍA DE ACCESO A BASES DE DATOS CON VISUAL BASIC", con fundamento en el punto 6 y siguientes, del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reúne los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
San Juan de Aragón, México., 31 de octubre de 1996
EL DIRECTOR

M en I CLAUDIO C. MERRIFIELD CASTRO

c c p Jefe de la Unidad Académica.
c c p Jefatura de Carrera de Ingeniería en Computación.
c c p Asesor de Tesis.

CCMC/AIR/lla.

Dedicatoria

A mis padres:

Quienes con su ejemplo me enseñaron lo que es un ser integro y solidario, y me guiaron por el camino del bien.

A mis Hermanos:

Con especial agradecimiento por el apoyo que me brindaron por esos momentos que hemos pasado juntos y que seguiremos compartiendo siempre.

A mi Esposa:

Quien con su trabajo y amor me proporcionó los medios para alcanzar la meta propuesta.

A mis amigos:

Quienes me han acompañado en mis buenos y malos momentos y me han sabido comprender.

A DIOS:

Gracias por haberme permitido llegar a este punto tan importante en mi vida.

Indice

Introducción

Los objetivos principales de esta tesis, es adentrar al lector con el mundo de las bases de datos de una manera rápida y práctica, que el lector pueda crear, manipular y modificar las bases de datos con un mínimo de código, así como de que sea capaz de crear aplicaciones visuales que permitan al usuarios tener un ambiente de trabajo muy ameno. Otro de los objetivos que se persiguen, es que el lector se familiarice con la programación orientada a eventos a través de objetos, que es el tipo de programación que usa Visual Basic.

Al explicarle el tipo de programación que se utiliza en Visual Basic, al lector se le ayudará a entender el estilo de programación de algunos otros programas que aunque no son iguales, son similares, como es el caso de Delphi, la forma de programación no es idéntica a Visual Basic pero es muy similar.

Justificación : Los motivos por los cuales se desarrolló esta tesis, es porque hay diversos manejadores de bases de datos, pero casi ninguno consta con un buen ambiente visual. Las aplicaciones que se desarrollan en Visual Basic nos permiten desplazarnos por medio de imágenes, iconos, barras de desplazamiento, etc. que hacen más fácil, familiarizarnos con dicha aplicación. Por otra parte, permite al programador utilizar las sentencias conocidas de SQL y de Basic ya que las sentencias utilizadas son similares a las de estos dos programas.

Por otra parte Visual Basic nos permite incorporar ayuda en línea para cada una de las pantallas de la aplicación, o para cada control, de esta forma nosotros tenemos a la mano toda la ayuda necesaria para utilizar los diferentes controles de los que consta nuestra aplicación.

Hipótesis : Todas las tareas que podemos realizar con nuestra aplicación, son referidas por medio de un control y una imagen. Para que el lector pueda experimentar lo que se le explica en cuanto a términos de bases de datos, esta tesis cuenta con gran número de rutinas que nos ayudan a obtener de diferentes maneras los datos que nosotros queremos. Nos dan la opción de escoger el formato de la base de datos que utilizaremos para nuestra aplicación.

Para tener una visión más clara en cuanto al costo-beneficio que nos proporciona Visual Basic con respecto a la realización de nuestras aplicaciones, se desarrolló un pequeño análisis de este tipo en las conclusiones.

Capitulo I : "Introducción a las bases de datos".
Objetivo : Proporcionar un conocimiento básico de que es un base de datos, y terminología que se usa en la manipulación de los datos dentro de una base de datos.

Capitulo II : "Como acceder a los datos con Visual Basic".
Objetivo : Proporcionar una rápida guía para acceder a los datos, tablas que nos muestran los objetos, propiedades, métodos y eventos que se necesitan para acceder a los datos.

Capitulo III : "Manejando base de datos usando Visual Basic".
Objetivo : Discute los lenguajes de definición de datos y muestra como abrir una base de datos ya existente, crear una nueva base de datos y manejar los datos de la bases de datos.

Capitulo IV : "Creando programas que manipulen datos".
Objetivo : Discute los lenguajes de manipulación de datos y muestra como puedes usar Visual Basic para extraer y manipular datos de bases de datos.

Capitulo V : "Usando controles de Visual Basic con objetos de acceso a datos".
Objetivo : Presenta información sobre el uso de controles que pueden ser ligados y controles que no pueden ser ligados en la edición profesional.

Apendice A : "Usando bases de datos Microsoft Access".
Objetivo : Discute los aspectos para trabajar con bases de datos creadas y manipuladas por Microsoft Access.

Apendice B : "Microsoft Access SQL".
Objetivo : Proporciona descripciones del lenguaje SQL para crear preguntas.

Apendice C : "Accesando bases de datos externas".
Objetivo : Discute los métodos para acceder a los datos en bases de datos externas y tablas ODBC.

Capitulo I

"Introducción a las bases de datos".

Conceptos Básicos

Byte

El byte es el grupo de bits más pequeño con dirección propia. Convencionalmente, el byte comprende ocho bits.

Item de datos

El ítem de datos es el grupo de datos más pequeño. Puede estar formado por cualquier número de bytes o bits.
A menudo llamamos campo o elemento de datos, al ítem de datos. En Cobol llámase a éste, ítem elemental.

Registro

El registro es una colección de ítems o agregados de datos. Cuando un programa de aplicación copia datos de una base de datos, es posible que copie un registro (lógico) completo. A menudo, el registro lógico de la base de datos es una estructura que incluye muchos grupos de ítems de datos que no tienen por qué ser leídos en conjunto.

Archivo

El archivo es la colección de todas las ocurrencias de un tipo de registro (lógico) dado. En los archivos más simples, todos los registros lógicos comprenden el mismo número de ítems de datos. En los archivos más complejos este número puede ser variable, a causa de la presencia de grupos repetitivos.

Bases de datos

La base de datos es una colección de ocurrencias de múltiples tipos de registro, pero incluye las relaciones que existen entre registros, y entre ítems de datos.

Sistemas de bases de datos

En la mayoría de los sistemas, la expresión base de datos no se refiere a todos los tipos de registro, sino a una colección limitada y especifica de éstos. Dentro de un sistema, existen por lo general, varias bases de datos; y se supone que los contenidos de estas bases son independientes. Las colecciones de bases de datos de esta clase se denominan sistemas de bases de datos. Las colecciones de bases de datos, se llaman a veces bancos de datos.

Registro Físico

El registro físico es la unidad básica de datos que se copia o escribe por medio de una única orden de entrada/salida, dada a la computadora. Comprende todos los datos que se registran entre espacios sucesivos en una cinta, o entre marcadores de dirección en un disco. El registro físico comprende a menudo, múltiples registros o segmentos lógicos.
En la mayoría de los sistemas es el programador quien determina la longitud del registro físico, pero hay también dispositivos en los que la longitud del registro físico no cambia.

Bloque de registros almacenados

El bloque es el grupo de datos contenido en un registro físico.

Conjunto de datos

El conjunto de datos es una colección de registros físicos. Incluye los datos necesarios para localizar los registros, por ejemplo, los índices.

Indexación

Algunas técnicas de direccionamiento y de indexación proveen como salida propia la dirección de un registro almacenado. Otros son menos precisos y suministran en cambio la dirección de un área en la que se almacena un grupo de registros. Llamaremos índice al área que contiene un grupo de registros que se direccionan en conjunto. El índice puede ser un registro físico, una pista o una celda, pero muchas veces es un agrupamiento determinado por una técnica de direccionamiento.

Aspectos a considerar de una Base de Datos

Hay quienes conciben la base de datos como un enorme receptáculo en el que un organismo guarda todos los datos procesables que reúne y al cual acuden muy diversos usuarios a consultar. Este gran almacén puede estar concentrado en una localidad determinada o distribuido en varias, todas ellas posiblemente interconectadas mediante un sistema de telecomunicación. Tienen acceso a la base de datos, programas de la más diversa índole.

La base de datos puede definirse como una colección de datos interrelacionados, almacenados en conjunto sin redundancias perjudiciales o innecesarias; su finalidad es la de servir a una aplicación o a más de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los datos almacenados. Un sistema comprende una colección de bases de datos cuando éstas son totalmente independientes desde el punto de vista estructural.

En las organizaciones más sencillas, encontramos casi siempre una colección de registros organizados para una aplicación determinada. La idea básica en la implantación de una base de datos es la de que los mismos datos deben ser aprovechados para tantas aplicaciones como sea posible. Por eso, la base de datos se concibe a menudo como un lugar de almacenamiento donde se reúne la información necesaria, para el ejercicio de las funciones propias de un organismo gubernamental, una empresa, una fábrica o cualquier otra organización. Semejante base de datos permitiría no solo la lectura de los datos almacenados, sino la continua modificación de los datos que son necesarios para el control de las operaciones.

En las grandes operaciones de procesamiento que no cuentan con el apoyo de una base de datos, hay tantos datos redundantes que resulta prácticamente imposible mantenerlos todos en el mismo nivel de actualización. Muchas veces, los usuarios o la gerencia general sufren las consecuencias de la incoherencia a que esto da lugar y dejan de considerar fiable la información que reciben.

Una de las características más importantes de la mayoría de las bases de datos es la de mantenerse en plena crisis de cambio y crecimiento. La base de datos debe prestarse a una fácil reestructuración siempre que haya que agregarle nuevos tipos de datos o utilizarla para nuevas aplicaciones. Esta reestructuración no debe originar la necesidad de volver a escribir los programas de aplicación, y en general, no debe ser fuente de trastornos. La facilidad con que pueda modificarse la base de datos tendrá siempre un efecto directo sobre la capacidad para desarrollar nuevas aplicaciones de procesamiento de datos dentro del organismo que la explota.

Cuando un determinado conjunto de datos sirve a una variedad de programas de aplicación, cada uno de estos percibe en general, diferentes relaciones entre aquéllos. En gran medida, es preocupación principal en la organización de la base de datos, la representación de las relaciones que existen entre ítems de datos y registros, así como también el cómo y el dónde se almacenan los datos. En las bases de datos previstas para aplicaciones diversas pueden existir múltiples interconexiones entre los datos.

Una de las ventajas que más se comenta en el caso de los "sistemas de información en general", es la de que los gerentes y sus colaboradores están en condiciones de plantear a su sistema de computación interrogantes imprevistos.

En lugar de estar organizada a modo de archivos rígidos, previstos para una aplicación determinada y a los cuales se accede siempre del mismo modo, la base de datos debe estar organizada de modo que los datos que contiene puedan ser alcanzados de muchas maneras diferentes y ser utilizados para responder a una diversidad de interrogantes. Lo primero es entonces, almacenar para cada entidad todos los datos de interés, no solo los que se necesitan para una determinada aplicación. El concepto de "depósito de información", hablando de los sistemas de información en general, o de cualquier otro sistema de información, es mucho más fácil de entender que de llevar a la práctica. La implementación de una base de datos basada en ese concepto demuestra ser una tarea larga y compleja; además, con el hardware actualmente disponible, resulta muy costoso diseñarla de manera que permita ser inspeccionada con una rapidez adecuada como para proporcionar respuestas en tiempo real a las interrogantes no previstas de antemano.

Objetivos de la organización de la base de datos

- **Los datos podrán utilizarse de múltiples maneras :**
  Diferentes usuarios, que perciben diferentemente los mismos datos, pueden emplearlos de distintas maneras.

- **Se protegerá la inversión intelectual :**
  No será necesario rehacer los programas y las estructuras lógicas existentes (que representan muchos hombres-años de trabajo) cuando se modifique la base de datos.

- **Bajo costo :**
  Bajo costo de almacenamiento y de uso de los datos, y minimización del costo de los cambios.

- **Menor multiplicidad de datos :**
  Las necesidades de las nuevas aplicaciones se satisfacerán con los datos existentes, más que creando nuevos archivos, evitándose así la excesiva multiplicación de datos que se advierte en las bibliotecas de cintas actuales.

- **Desempeño :**
  Los pedidos de datos se atenderán con la rapidez adecuada según el uso que de ellos habrá de hacerse.

- **Claridad :**
  Los usuarios sabrán qué datos se encuentran a su disposición y los comprenderán sin dificultad.

• **Rápida atención de interrogantes no previstas :**
Los pedidos espontáneos de información se atenderán sin necesidad de escribir un programa de aplicación, sino utilizando un lenguaje de alto nivel para averiguación o generación de reportes.

• **Facilidad para el cambio :**
La base de datos puede crecer y variar sin interferir con las maneras establecidas de usar los datos.

• **Precisión y coherencia :**
Se utilizarán controles de precisión. El sistema evitará las versiones múltiples de los mismos ítems de datos, con diferentes estados de actualización.

• **Reserva :**
Se evitará el acceso no autorizado a los datos. Los mismos datos podrán estar sujetos a diferente restricciones de acceso para diferentes usuarios.

• **Protección contra pérdida o daño :**
Los datos estarán protegidos contra fallos y catástrofes, y contra delincuentes, vándalos incompetentes y personas que intenten falsearlos.

• **Disponibilidad :**
Los datos se hallarán inmediatamente disponibles para los usuarios casi todas las veces que los necesiten.

Los sistemas de bases de datos son propensos a ser inflexibles y problemáticos. El grado de complejidad de muchas bases de datos parece crecer sin límites previstos. Amenos que los diseñadores tengan un muy claro concepto de lo que está ocurriendo.

Por medio de un software de administración adecuado, la vista de los datos que se presenta al usuario, se mantendrá independiente de la representación física, de modo que, hasta el hardware podrán alterar sin afectar la descripción lógica de los datos que interesan al usuario.

Los diferentes usuarios de la misma base de datos percibirán diferentes conjuntos de datos y diferentes relaciones entre ellos. Es por lo tanto necesario extraer de las tablas, los subconjuntos pedidos por algunos usuarios, creando así tablas de menor grado, o por el contrario, a veces es preciso fundir dos o mas tablas en una, creando una de mayor grado. Dado que algunas tablas tendrán muchas filas y columnas, es importante la capacidad para la extracción de subconjuntos.

La vista lógica de la base de datos consistirá, en estos términos, en conjuntos de tablas bidimensionales con operaciones para extraer columnas y combinarlas indistintamente.

Ha habido muchas implementaciones de bases de datos relacionales. Aunque se dice a menudo, que las relaciones binarias son las que ofrecen el mayor grado de flexibilidad, para la mayoría de las aplicaciones comerciales parecen preferibles las relaciones de grado variable. La mayoría de las bases diseñadas a partir de las relaciones binarias fueron diseñadas para aplicaciones que utilizan datos menos sofisticados que los que son propios de la típica base de datos comercial.

Capitulo II

"Como acceder a los datos con Visual Basic".

Las aplicaciones de hoy en día usan una gran cantidad de información, estos datos son manejados por medio de estructuras de bases de datos. El programa Microsoft Visual Basic para Windows opera mediante la creación de bases de datos estructuradas en la aplicación, que sirven como principio y final de muchos formatos populares de bases de datos. Visual Basic crea bases de datos fáciles de manejar, rápidas y libres de problemas.

Visual Basic implementa el acceso a datos igual al mecanismo usado por Microsoft Access, combinado con Visual Basic. Esta tecnología ayuda al ensamblaje para acceder a formatos estandares de bases de datos entre los que podemos mencionar : Microsoft Access, Btrieve, dBase, Microsoft FoxPro, Oracle, Paradox, y Microsoft SQL Server. Si tu tienes una base de datos ya creada en uno de estos formatos, tu puedes usar Visual Basic para manipular esta base de datos ya creada.

Para acceder a objetos de datos tales como bases de datos (**Database**), tablas (**Table**), campos (**Field**) y objetos indexados (**Index**), Visual Basic activa tu base de datos con un mínimo de programación. El modelo de los objetos simplifica el código que tu necesitas escribir para manipular la estructura y el mecanismo de los datos. Tu puedes acceder a los objetos de los datos y a las mismas propiedades, métodos y funciones para trabajar con todos los formatos soportados por las bases de datos .

Las formas para acceder a bases de datos son muy extensas, tu puedes usar manejadores (driver) de bases de datos adicionales con los cuales tu puedes explorar, y dependiendo de la flexibilidad de el modelo de objetos, tu no necesitarás hacer cambios en tu código para implementar estos.

La versión 3.0 de Visual Basic (Edición Profesional) contiene la utileria del ODBC. El acceder por medio del ODBC nos permite conectarnos a bases de datos externas de Visual Basic, el ODBC es usado para conectarnos al Servidor Microsoft SQL y sistemas de manejadores de bases de datos Oracle.

Bases de datos a las cuales se pueden acceder con Visual Basic

Visual Basic activa la aplicación para que pueda manipular los datos, bajo una estructura interna de diferentes formatos de bases de datos. Por ejemplo, tu puedes usar Microsoft Access para crear y manejar una base de datos que tu programa Visual Basic pueda manipular. También puedes crear bases de datos en Visual Basic que Microsoft Access pueda manejar. Visual Basic también proporciona acceso a otros formatos populares de bases de datos. Hay tres categorías de bases de datos que Visual Basic reconoce :

1.- Formatos de base de datos Microsoft Access : Estos archivos de bases de datos son manipulados directamente por Visual Basic y los puedes crear ó manipular con Visual Basic o Microsoft Access. Este es el formato nativo de Visual Basic, este proporciona la mayor flexibilidad y rapidez .

2.- Bases de datos externas : Esta categoría incluye formatos como los de Btrieve, dBase III, dBase IV, Microsoft FoxPro versión 2.0 y 2.5, y Paradox. Tu puedes crear o manipular todos estos formatos de bases de datos en Visual Basic. Puedes consultar la ayuda en línea de los archivos que determinan cuales bases de datos externas son soportadas. El archivo ISAM creado con Visual Basic para MS-DOS no esta soportado directamente, pero este archivo puede exportar formatos ASCII y reimportarlos dentro de Visual Basic.

3.- Bases de datos externas ODBC: Esta categoría incluye bases de datos cliente/servidor, también las de Microsoft SQL Server y sistemas de manejadores de bases de datos Oracle . En el caso de bases de datos ODBC, tu puedes enviar comandos SQL directamente a el servidor externo para procesar.

Alternativas para datos tipo objeto Visual Basic

Los datos tipo objeto no son pocos, tu también puedes manipular los datos tipo objeto de Visual Basic . Tu también tienes acceso a las siguientes fuentes:

• Secuencial, casual y métodos de acceso a archivos binarios. Algunas de las instrucciones son: **Get, Input, Print** y **Put** con las cuales puedes manipular los datos.

• Las librerías de Visual Basic SQL (VBSQL) para Microsoft SQL Server. VBSQL proporciona un enlazamiento parecido a las aplicaciones en lenguaje C, lo que permite la interface con el (API) para funciones de librería DB. Tu puedes usar este API para comunicarte directamente con Microsoft (o SYBASE) SQL Server.

• Las librerías ODBC: Son también para explorar la aplicación que llama directamente el ODBC API. Este método es mas complejo para implementar, este permite cerrar correctamente el enlazamiento entre la aplicación y el ODBC, usando datos tipo objeto.

La tercera parte de los Dll's (Dynamic Link Libraries). No proporcionan la facilidad de acceder a los datos con los controles ligados (controles bound).

Para el uso de esta gran cantidad de bases de datos se recomienda que se utilice la versión profesional de Visual Basic. A continuación se hace una comparación entre la versión estándar y la profesional .

Tabla 1.1 Diferentes caracteristicas entre la Edición estándar y la profesional :

Características	**Edición estándar**	**Edición profesional**
Controles Bound	Si	Si
Data control	Si	Si
Creación de objetos **Database**	Solo con el Data Control	Si
Creación de objetos **Dynaset**	Solo con el Data Control	Si
Creación de objetos **Index**	No	Si
Creación de objetos **QueryDef**	No	Si
Creación de objetos **Snapshot**	No	Si
Creación de objetos **Tables**	No	Si
Creación de objetos para acceso a datos	No	Si
Creación de nuevas bases de datos	No	Si
Modificación de la estructura de una Base de datos	No	Si

En la edición estándar, el acceso a los datos solo lo puedes hacer mediante el Data Control.

Objetos usados para acceder a los datos

Tabla 1.2 Objetos y Colecciones

Objetos o Colecciones	Propiedad	Método
Objeto **Database**	**CollatingOrder** **Connect** **Name** **QueryTimeout** **Transactions** **Updatable**	**BeginTrans** **Close** **CommitTrans** **CreateDynaset** **CreateQueryDef** **CreateSnapshot** **DeleteQueryDef** **Execute** **ExecuteSQL** **ListFields** **ListTables** **OpenQueryDef** **OpenTable** **Rollback**
Dynaset, Table y Snapshot	**BOF** **Bookmark** **Bookmarkable** **DateCreated** **EOF** **Filter** **Index** **LastModified** **LastUpdated** **LockEdits** **Name** **NoMatch** **RecordCount** **Sort** **Transactions** **Updatable**	**AddNew** **Clone** **Close** **CreateDynaset** **CreateSnapshot** **Delete** **Edit** **FindFirst** **FindLast** **FindNext** **FindPrevious** **ListFields** **ListIndexes** **MoveFirst** **MoveLast** **MoveNext** **MovePrevious** **Seek** **Update**
Objeto **Field**	**Attributes** **CollatingOrder** **Name** **OrdinalPosition** **Size** **SourceField** **SourceTable** **Type** **Value**	**AppendChunk** **FieldSize** **GetChunk**

Objetos o Colecciones	Propiedad	Método
Objeto **Index**	**Field** **Name** **Primary** **Unique**	**(Ninguno)**
Objeto **TableDef**	**Attributes** **Connect** **DateCreated** **LastUpdated** **Name** **SourceTableName** **Updatable**	**(Ninguno)**
Objeto **QueryDef**	**Name** **SQL**	**Close** **CreateDynaset** **CreateSnapshot** **Execute** **ListParameters**
TableDefs, Fields, Colecciones de Indexes	**Count**	**Append** **Delete** **Refresh**

Tabla 1.3 Data Control

Propiedades para acceder a datos	Métodos para acceder a datos	Eventos para acceder a datos
Connect **Database** **DatabaseName** **EditMode** **Exclusive** **Options** **ReadOnly** **Recordset** **RecordSource**	**Refresh** **UpdateControls** **UpdateRecord**	**Error** **Reposition** **Valide**

Tabla 1.4 Controles unidos (controles bound)

Propiedades para acceder a datos	Métodos para acceder a datos	Eventos para acceder a datos
DataChanged **DataField** **DataSource**	Ninguno	Ninguno

Tu puedes utilizar las siguientes sentencias y funciones de Visual Basic para acceder a los datos.

- Sentencia **BeginTrans**
- Sentencia **CommintTrans**
- Sentencia **CompacDatabase**

- Funcion **CreateDatabase** (Solo en la edición profesional)
- Sentencia **FreeLocks**
- Funcion **OpenDatabase** (Solo en la edición profesional)
- Sentencia **RegisterDatabase**
- Sentencia **RepairDatabase**
- Sentencia **Rollback**
- Sentencia **SetDataAccessOption**
- Sentencia **SetDefaultWorkspace**

Capitulo III

"Manejando base de datos usando Visual Basic".

Este capitulo se enfoca al lenguaje de definición de datos (dll), tu puedes usarlos para definir estructuras de bases de datos. Este explica como puedes usar las funciones, declaraciones, métodos, propiedades y objetos de acceso a datos, para definir tus tablas de bases de datos, campos e indexados. Incluye técnicas en las cuales tu puedes crear nuevas bases de datos, modificar la estructura de la base de datos y manejar la base de datos.

Hay cuatro aspectos importantes los cuales tu debes de conocer para manejar una base de datos :

• Si tu tienes una base de datos ya existente y tu quieres saber como puedes abrir esta, deberás ir a la sección **" III.1 Abriendo una base de datos "**.

• Si tu quieres crear una nueva base de datos, existe una sección que trata de este tema, como lo es **" III.3 Creando una base de datos"**.

• Si tu quieres modificar una base de datos existente, debes ir a la sección **" III.4 Modificando la estructura de una base de datos"**.

• Si tu quieres realizar diferentes operaciones con la base de datos, debes ir a la sección **" III.5 Manipulando una base de datos"**.

Tu puedes usar el ejemplo de la base de datos Biblio.mdb proporcionada con Visual Basic para explorar muchos de los temas tratados en esta tesis.

Contenido :

III.1 Abriendo una base de datos :

Si tu quieres leer lo que contiene una base de datos y tu quieres acceder a esta con Visual Basic. Tu puedes utilizar dos diferentes métodos para abrir una base de datos en Visual Basic :

- Usando el Data Control.
- Usando la función **OpenDatabase**.

Usando el Data Control

Nosotros podemos usar el Data Control para reducir significativamente el número de líneas de código y hacer mas fácil el abrir y manipular una base de datos. El Data Control tiene unas cuantas limitaciones en su funcionalidad, flexibilidad, o para robustecer nuestras aplicaciones, pero sin embargo es muy fácil de usar en algunas situaciones.

Tu puedes conectar el Data Control a una base de datos para controlar sus propiedades. Una vez que el Data Control es configurado, tu puedes efectuar muchas funciones con un pequeño código :

- Abrir una base de datos y crear una base de datos asociándola a una variable.
- Crear un objeto **Dynaset** conteniendo el dato.
- Mover el puntero del registro grabado con sus respectivos datos.
- Usar los controles que pueden ser ligados (controles bound) para aceptar y hacer cambios a la base de datos.
- Proporcionar un error al puntero.

Si tu no cambias y usas el Data Control para abrir una base de datos, tu no debes usar el control ligado para ver o editar tus datos, hay también restricciones para el uso del Data Control con la función **OpenDatabase** . Para mas información, ver "Usando el **OpenDatabase** con el Data Control", mas adelante en este capítulo.

Usando la función OpenDatabase

La otra alternativa para abrir una base de datos es usar la función **OpenDatabase** para crear un objeto **Database**. Tu puedes crear un objeto **Database** con la función **OpenDatabase**, es idéntico a un objeto **Database** creado con el Data Control. La función **OpenDatabase** ocupa los siguientes datos :

- Nombre de la base de datos.
- Descripción de el método de acceso a la base de datos.
- Indicar si la base de datos va a tener un solo uso o múltiples usos.
- Indicar el acceso a la base de datos, si es para solo lectura o lectura y escritura.
- Abrir el dato especificado y asignarlo a una variable **Database**.
- Crear una base de datos interna (Access) o externa (Btrieve, dBase III, dBase IV, Microsoft FoxPro, Oracle, o Servidor SQL).

La sintaxis de la función **OpenDatabase** es :
OpenDatabase(Databasename [,Exclusive [,ReadOnly [,Connect]]])

El siguiente ejemplo nos muestra, el primer uso de la declaración **Dim**, para crear una variable **Database**, y el uso de la declaración **Set** para asignarle a una variable el resultado de la función **OpenDatabase**.

En el siguiente ejemplo, la función **OpenDatabase** abre una base de datos (Biblio.mdb), para lectura y escritura, de formato Microsoft Access y localizada en el directorio de trabajo.

```
Dim Db As Database
Set Db = OpenDatabase ("Biblio.mdb")
```

Solo tu inicializas la variable **Database** (en este caso Db), tu puedes examinar o modificar la estructura de la base de datos Biblio.mdb, o acceder a lo grabado en la base de datos.

Usando el Databasename y la parte Connect de la función OpenDatabase

El **Databasename** y la parte **Connect** de la función **OpenDatabase**, trabajan en conjunto para identificar la fuente y formato de los datos. La fuente de tu base de datos puede estar en tu WorkStation (estación de trabajo), en un servidor de red compartida, o un servidor ODBC. La base de datos puede ser de forma individual un archivo **.mdb**, un directorio de base de datos, un índice, archivos de soporte, o un registro ODBC de la fuente de datos.

- **Databasename** : Proporciona a Visual Basic el nombre de la fuente de los datos. La sintaxis del **Databasename** depende del tipo de base de datos que se este abriendo.

• **Connect** : Provee a Visual Basic el tipo de base de datos a ser abierta y los parámetros opcionales que pueden ser necesitados para tener acceso a la base de datos. La cadena **Connect** puede ser omitida.

La siguiente tabla muestra el formato de cada base de datos soportada, la función **Databasename** y la parte **Connect** indican la fuente de los datos, tipo y parámetros necesarios.

Formato de la B.D.	Databasename	Connect
Microsoft Access	drive:\path\file.mdb	(nada)
Btrieve	drive:\path\file.ddf	"Btrieve"
dBase III	drive:\path	"dBase III"
dBase IV	drive:\path	"dBase IV"
FoxPro Versión 2.0	drive:\path	"FoxPro 2.0"
FoxPro Versión 2.5	drive:\path	"FoxPro 2.5"
ODBC (SQL Server, Oracle)	usualmente el nombre del servidor	"Odbc;Dsn=Server;UID=User;Pwd=Password"
Paradox	drive:\path	"Paradox"

Nota : Para formatos de bases de datos Visual Basic y Microsoft Access, el nombre completo del archivo (incluyendo la extensión .mdb) es provisto en la parte **Databasename**; no se requiere la parte **Connect**. Si Visual Basic no encuentra el archivo especificado, tu aplicación generará un error, y la nueva base de datos no será creada .

Para bases de datos externas, el **Databasename** especifica la ruta completa del directorio de la base de datos. En el caso de Btrieve, la base de datos es la ruta para el archivo File.ddf.
Si tu red soporta archivos del tipo de la Universal Naming Convention (UNC), tu puedes usar esto para direccionar la base de datos.

Usando la parte Exclusive del OpenDatabase

Si quieres compartir la base de datos con otros usuarios, la parte **Exclusive** de la función debe estar en falso. Si la base de datos ya ha sido abierta por otro usuario, tu no podrás abrirla con la parte **Exclusive** colocada en verdadero. Una vez que tu base de datos ha sido abierta en modo exclusivo, ningún otro usuario podrá abrirla hasta que tu la cierres. La parte **Exclusive** de la función esta por default en falso (modo compartido) .

Si abres la base de datos en modo exclusivo, tu aplicación puede aun abrir la base de datos otra vez. En otras palabras tu puedes abrir una base de datos tantas veces como quieras, aun si ya abriste la base de datos en modo exclusivo.

Usando la parte ReadOnly de la función OpenDatabase

Si quieres abrir una base de datos solo para lectura, ajusta la parte **ReadOnly** en verdadero. Cuando una base de datos es abierta solo para lectura, Visual Basic no tiene que gastar mucho tiempo con impresiones de lectura/escritura, lo cual hace más rápido el acceso a los datos. Esta propiedad por default se encuentra en falso.

Si tu aplicación abre la misma base de datos muchas veces, Visual Basic ignorará la parte **Read/Only** en subsecuentes accesos a un archivo de la misma base de datos. En otras palabras, si tu abres una base de datos como solo lectura, el modo solo lectura será ajustado subsecuentemente cada vez.

Después de que cada una de estas bases de datos abiertas hayan sido cerradas, tu puedes cambiar el estado de solo lectura.

Condiciones especiales cuando usas OpenDatabase

Cuando usas la función **OpenDatabase** y especificas una base de datos externa, Visual Basic encuentra todos los archivos similares y los abre, o delega esta operación a un proceso de base de datos externa. Si especificas una base de datos externa (no ODBC) y no existe, Visual Basic trata de crear una base de datos con el formato indicado por el **Databasename** y la parte **Connect** de la función.

Nota : Condiciones especiales pueden aplicarse cuando te conectas a la base de datos de Microsoft Access que usa **login** y **passwords**, o a bases de datos externas no ODBC . Para más información de como conectarse a la base de datos Microsoft Access, ver la sección "Manejando una base de datos" que se encuentra más adelante en este capítulo. Para más información de como conectarse a bases de datos ODBC, ver la sección "Consideraciones cuando uses bases de datos ODBC" más adelante en este capítulo .

Una vez que Visual Basic localiza y abre la bases de datos, la función **OpenDatabase** provee variables objeto **Database**. Cada base de datos esta únicamente identificada por su propia variable **Database**. Una variable **Database** abierta es requerida para todas las operaciones que involucran acceso a los datos, ya sea que tu estés creando una nueva base de datos o modificando la estructura de una base de datos, también tu puedes crear nuevas bases de datos o modificar las estructuras de las bases de datos existentes. Tu puedes abrir más de 256 bases de datos simultáneamente.

Tu no podrás abrir bases de datos si :

- La base de datos ya a sido abierta por otro usuario (no incluyendo tu propia aplicación) en modo exclusivo.
- Un archivo especifico de la base de datos no fue encontrado (Recuerda incluir la ruta completa).

Abriendo diferentes tipos de bases de datos

La siguiente tabla muestra las partes que tu puedes usar con la función **OpenDatabase** para abrir muchos tipos de bases de datos estandar .

Tipos de Bases de Datos	Función OpenDatabase y sus partes
Microsoft Access	**OpenDatabase**("Mydb.mdb",False,False,"") o **OpenDatabase**("Mydb.mdb")
Btrieve	**OpenDatabase**("C:\Mybee.ddf",False,False, "Btrieve;")
dBase III	**OpenDatabase**("C:\Mydbx",False,False, "dBase III;")
FoxPro versión 2.5	**OpenDatabase**("C:\Myfox",False,False, "FoxPro 2.5;")
ODBC (Dialogo Login)	**OpenDatabase**("",False,False,"ODBC;")
Paradox	**OpenDatabase**("C:\Mypdx",False,False, "Paradox;")
Registrando una Fuente del ODBC("Accounts")	**OpenDatabase**("Accounts",False,False, "ODBC;")
Definición del ODBC DSN en el Odbc.ini con la información de la cadena **Connect**	C$="ODBC; UID=sa; PWD=zzx; DSN=Accounts; **OpenDatabase**("",False,False,C$)

Usando OpenDatabase con el Data Control

Tu no puedes usar la función **OpenDatabase** para crear una variable **Database** y proporcionar la propiedad **Database** de un Data Control. Esto es porque la propiedad **Database** y **Recordset** de los Data Control's son de solo lectura .

Consideraciones cuando usas bases de datos ODBC

Los siguientes párrafos describen algunas de las únicas consideraciones que debes de tomar en cuenta cuando estés trabajando con bases de datos ODBC. Para mas información sobre bases de datos ODBC, ver el apéndice C, "Acceso a bases de datos externas".

La función OpenDatabase y las fuentes de datos ODBC

Si la información proporcionada en la función **OpenDatabase** es insuficiente para Visual Basic para abrir tu bases de datos, la interface ODBC proporcionará una caja de diálogo para recoger la información perdida (tales como el nombre del usuario, password y bases de datos por default).

Si tu no quieres que aparezca la caja de diálogos del ODBC, asegúrate de checar la propiedad **Connect** después de que la base de datos a sido abierta por primera vez (durante el desarrollo). Esto tendrá los valores proporcionados en la caja de diálogo en el ODBC. Estos valores pueden ser puestos dentro de la parte **Connect** de la función, así el diálogo no necesitará aparecer otra vez.

Tu puedes forzar la caja de diálogo del ODBC a aparecer usando una cadena vacía para **Databasename** y ajustando el **Connect** a "ODBC;" o también "". Tu puedes tambien usar este método para ver todos los datos ODBC disponibles una vez que hayas provisto un login o un password válido.

En el caso de fuentes de datos ODBC, la parte del **Databasename** corresponde a un registro de entrada en el archivo Odbc.ini, especificando un nombre de fuente de datos (DSN) valido.

Para mas información de como establecer fuentes de datos válidos, buscar **RegisterDatabase** en la ayuda en línea provista por el soporte de las bases de datos ODBC.

Parámetros adicionales en la cadena Connect

Para bases de datos ODBC y otros casos donde necesites proporcionar parámetros adicionales para bases de datos externas, tu debes especificar parámetros adicionales en la cadena **Connect** de la función **OpenDatabase**, la sintaxis es la siguiente:

Tipo de fuente de dato; [parámetro connect; [parámetro connect;]]

Para colocar parámetros adicionales, tu debes primero especificar un tipo de fuente de dato valido, seguido por un punto y coma (;), y entonces indica uno o más de los siguientes parámetros **Connect**. Teniendo cuidado de separar cada parámetro con un punto y coma.

Valores de los parámetros de la conexión	Especificaciones
UID	Nombre del ID
PWD	Password
DSN	Fuente de datos o nombre del servidor. Puede especificarse el nombre de la base de datos.

Valores de los parámetros de la conexión	Especificaciones
DATABASE	Base de datos por default que utiliza el servidor
LOGINTIMEOUT	Números de segundos que espera el servidor para responder
APP	Nombre de la Aplicación
WSID	ID de la Workstation
LANGUAGE	Lenguaje por default

Por ejemplo, si para la base de datos externa requieres un password, necesitas especificar este password usando el parámetro **PWD** en la parte **Connect** de la función. El siguiente código muestra el **Connect** usado para especificar a la base de datos Paradox, el password "mydbase":

```
Set Db = OpenDatabase ( Databasename | " Paradox; PWD = mydbase; " )
```

La propiedad QueryTimeout del ODBC

La propiedad **QueryTimeout** determina el número de segundos que Visual Basic espera para hacer una consulta al ODBC o para completar una operación de una ddl antes de que se produzca un mensaje de error. Tu puedes ajustar este valor para cada base de datos abierta; esto se aplica a todas las fuentes ODBC.

La forma en que el servidor remoto ODBC reaccionará a tu aplicación con un tiempo fuera depende del servidor. En algunos casos la transacción en proceso será cancelada y regresará; En otros casos la transacción puede haber sido parcialmente completada o totalmente completada .

Por ejemplo, suponiendo que quieras acceder a dos fuentes de datos separadas, a un servidor local y un servidor remoto, en dos servidores ficticios ODBC. Para ajustar el retraso del **QueryTimeout** en cada fuente de datos, deberás usar el siguiente código:

```
Dim Db1 As Database
Dim Db2 As Database
Dim Ds As Dynaset
Dim R As Integer
Dim Con As String
On Error GoTo MyTimeOutHandler
Con$="ODBC;PWD=tgXpt44;UID=Acnt;"
Set Db1 = OpenDatabase("LocalServer",False,False, Con$)

Con$ = "ODBC;PWD=y5ttRw;UID=MgrAct;"
Set Ds2 = OpenDatabase("RemoteServer",False,False, Con$)

Db1.QueryTimeout = 20
Set Ds = Db1.CreateDynaset("Titles")
Db2.QueryTimeout = 120
R% = Db2.ExecuteSQL("Update Authors Set Status=1 Where Status=6")
```

El parámetro LoginTimeout del ODBC

Para cambiar el número de segundos que Visual Basic debe esperar a que un servidor ODBC responda la petición de apertura antes de entrar a un tiempo fuera, use el parámetro **LoginTimeout** de la función, en la parte **Connect**. Por ejemplo, el siguiente código muestra el **LoginTimeout** para 120 segundos:

```
Dim Db As Database, C As String
C$ = "ODBC;PWD=tgXpi44;UID=Acnt;LoginTimeout=120"
On error GoTo DBOpenError
Set Db = OpenDatabase("Dbtopics",False,False, C$)
```

Cuando ODBC abre una tabla de bases de datos directamente, o une tablas de las bases de datos ODBC, tu puedes encontrar que el valor por default del tiempo fuera ajustado en los archivos de inicialización no son suficientes para tus necesidades. Algunos servidores pueden tomar varios minutos para responder en redes lentas, mientras que otros responden instantáneamente, esto es importante para ajustar el **LoginTimeout** a un límite de tiempo apropiado para una situación específica. Debes ajustar los tiempos apropiados, y estar seguro que los usuarios no serán encerrados innecesariamente o abandonen un llamado de la función del **OpenDatabase** que será completada normalmente si no abandonas la corrida a unos pocos segundos .

Cuando el límite del **LoginTimeout** es alto, tu aplicación obtendrá un error de tiempo fuera. En otros casos Visual Basic te indica que el servidor especificado no pudo ser encontrado. Esto puede ser porque el nombre del servidor que usaste es incorrecto, el servidor esta inhabilitado, por que la red que lo liga esta inhabilitada, o no tienes los permisos apropiados de la red para acceder al servidor.

Cerrando Objetos Database

Una vez que se has usado una base de datos, debes cerrar la misma. Cerrando una base de datos liberas los recursos ocupados por esta, liberas llaves u otros recursos compartidos que la base de datos estaba reteniendo.

Si ocupaste la función **OpenDatabase** para abrir tu base de datos, debes cerrar todos los **Recordset** abiertos y previo a esto, haber cerrado todos los **Tables**, **Dynaset** y **Snapshots**, antes de cerrar tu base de datos. Una vez que han sido cerrados todos los **Recordset**, puedes cerrar la base de datos con el método **Close**:

```
Db.Close
```

Cerrar una base de datos es especialmente importante si usas variables de bases de datos globales o estáticas. Tu no necesitas cerrar explícitamente una base de datos o abrir un **Recordset** con el Data Control, estas bases de datos son cerradas automáticamente cuando tu descargas el Data Control de la forma. Tu no necesitas cerrar bases de datos explícitamente que han sido declaradas usando variables locales en un procedimiento, ya que todos los objetos de datos son cerrados y las variables borradas cuando el procedimiento termina.

Porque Visual Basic no usa memoria libre para datos de variables objeto, hasta que tu limpies las variables, finalice tu programa, reinicie tu código, o Visual Basic se cierre por si mismo, ninguna base de datos referida con estas variables ha sido automáticamente cerrada.

Cuando tu programa termina y la base de datos ha sido cerrada ya no puedes hacer nada al respecto, todas las transacciones incompletas están perdidas.

III.2 Analizando la estructura de una base de datos

Una vez abierta la base de datos o esquema para determinar los nombres y propiedades de el objeto **Database**, se dice que se a mapeado la base de datos. Una vez mapeada la base de datos, puedes encontrar mucho mas fácil escribir aplicaciones para manipular datos.

La siguiente información sobre mapeo de estructuras de bases de datos asume que tu entiendes como Visual Basic usa objetos relacionados para referenciar los elementos de la estructura de la base de datos .

Aunque puedes usar el Data Control para mapear estructuras de la base de datos, usando objetos es mas fácil y requiere menos código para mapear tu base de datos, necesitas hacer algo de lo siguiente:

- Usar la aplicación manejador de datos "Data Manager".
- Correr un corto procedimiento de Visual Basic para mapear tablas de bases de datos, campos e índices.

Cuando mapeas una base de datos, expones un número de propiedades que han sido totalmente discutidas en la edición profesional. Para mayor información sobre estas propiedades, busca en el Help alguna propiedad específica, o busca la sección "Creando una base de datos" más adelante en este capítulo.
Todos los ejemplos en esta sección asumen que tu has abierto la base de datos de ejemplo usando el siguiente código:

```
Dim i As Integer
Dim Db As Database, Td As TableDefs
Set Db = OpenDatabase("Biblio.mdb")
Set Td = Db.TableDefs
```

Analizando propiedades de un objeto Database

Visual Basic ajusta las propiedades del **Database** cuando tu abres la base de datos. Solo la propiedad **QueryTimeout** puede ser alterada durante la corrida. La siguiente tabla muestra las propiedades expuestas por el objeto **Database**.

Propiedades	Descripción
CollatingOrder	Tipo de ordenamiento de la Base de Datos
Connect	La cadena **Connect** usada para abrir la base de datos
Name	El nombre de la base de datos
QueryTimeout	Tiempo en que responde el ODBC
Transactions	Indica cuales transacciones son soportadas por la base de datos
Updatable	Indica cuando la base de datos fue actualizada

Tu puedes usar el siguiente código para ver las propiedades de los objetos **Database**:

```
Print "Fuente de dato :", Db.Name
Print "Los parámetros de la cadena Connect :", Db.Connect
Print "Soporta transacciones ?:", Db.Transactions
Print "Esta la base de datos actualizada ?:", Db.Updatable
Print "Tipo de ordenamiento ?:", Db.CollatingOrder
Print "Segundos que se tarda el QueryTimeout ?:, Db.QueryTimeout
```

Analizando Definiciones de Tablas

El nombre de cada tabla de la base de datos esta contenido en la colección de los **TableDefs**, junto con otros detalles de la Tabla. Cada contenido u objeto **TableDef** puede ser referido por un número o por un nombre.

Propiedades de los TableDefs

La propiedades **Count** de un **TableDefs** contiene el número de objetos en la colección del **TableDef**, así como el número de tablas en la base de datos. La siguiente tabla muestra las propiedades del **TableDef**.

Propiedad	Descripción
Attributes	Opción y estados de banderas.
Connect	La cadena **Connect** es usada para unirlas.
DateCreated	Fecha y hora en que fue creada la tabla.
LastUpdated	Fecha y hora en que fue cambiada la estructura de la tabla.
Name	Nombre de la tabla.
SourceTableName	El nombre de la tabla de la B.D. ODBC o externa.
Updatable	Indica si la tabla es actualizable o no.

Atributos TableDef

Las propiedad de un objeto **TableDef** describen la fuente de los datos, indica si la **Tabledef** esta abierta en modo exclusivo, y determina si el password para acceder esta tabla debe ser salvado con la información revisada. La siguiente tabla muestra los ajustes para la propiedad **Attributes**, estas constantes y todas las otras constantes para acceder a los datos están definidas en el archivo Visual Basic Datacons.txt.

Atributo	Descripción
DB_ATTACHEDODBC	Una tabla asignada para base de datos ODBC tales como Microsoft SQL Server u Oracle.
DB_ATTACHEDTABLE	Una tabla asignada para una base de datos no ODBC tales como Microsoft Access o Paradox.
DB_ATTACHEXCLUSIVE	Una tabla abierta para acceso exclusivo Microsoft Access.
DB_ATTACHSAVEPWD	Salva el password con el que se ligo la información, si este es falso, el password debe ser suministrado en otra parte
DB_SYSTEMOBJECT	Un sistema de tablas (este no debe ser modificado).

El siguiente ejemplo muestra el código que puedes usar para checar los ajustes para las propiedades **TableDef**.

```
For i% = 0 To Td.Count - 1
        If ( Td( i ) . Attributes and DB_SYSTEMOBJECT ) = 0 Then
                Print "Nombre de la Tabla :",Td ( i ) . Name
                Print "Dato creado :",Td ( i ) . DateCreated
                Print "Siguiente dato actualizado :",Td ( i ) . LastUpdated
                Print "Tabla actualizada :",Td ( i ) . Updatable
                Print "Atributos :",Td ( i ) . Attributes
                Print "Connect :",Td ( i ) . Connect
                Print "Nombre de la tabla fuente :",Td ( i ) . SourceTableName
        End if
Next i%
```

Sistemas de tablas

Las bases de datos de Visual Basic contiene todos los sistemas de tablas usados para llevar la cuenta de todos los otros objetos en la base de datos y para manejar su integridad referencial. Estos sistemas de tablas están incluidos en el objeto **Snapshot** creado por el método **ListTables**, y en la colección **TableDefs**. Usualmente, tu quieres código para ignorar los sistemas de tablas.

Visual Basic usa los atributos de la propiedad **TableDef** para describir cada tabla. Los atributos están también contenidos en el método **ListTables** del **Snapshot**.

Para checar los atributos para determinar si una tabla es un sistema de tablas, usa el operador **And** y la constante apropiada predefinida para examinar los bits. Puedes también usar esta técnica con los campos de atributos regresados por los métodos **List**.

Precaución .- No todos los bits regresados por Visual Basic están documentados aquí. Cuando ajustes las opciones de Attributos, no debes alterar bits indefinidos.

Mapeo de propiedades TableDef

El siguiente ejemplo muestra como puedes usar el campo de Atributos en el objeto **TableDef** para mapear las propiedades **TableDef**. Los objetos **QueryDef** y sus parámetros no aparecerán en la lista generada de propiedades **TableDef**; ellas son expuestas solo usando el método **ListTables**, discutido mas tarde.

```
Sub DisplayTables ( )
    Dim Db As Database
    Dim i As Integer
    Set Db = OpenDatabase("Biblio.mdb")
    Debug.Print "Nombre de la tabla", "Creado", "Actualizado", "Tabla actualizada ?", "Flags"
        For i% = 0 To Db.TableDefs.Count - 1
                Debug.Print Db.TableDefs ( i% ) . Name,
                Debug.Print Db.TableDefs ( i% ) . DateCreated,
                Debug.Print Db.TableDefs ( i% ) . LastUpdated,
                If Db.TableDefs ( i% ) . Updatable = True Then
                    Debug.Print "True",
                Else
                    Debug.Print "False",
                End If

        ' Show para ver los bits de los atributos
        Debug.Print Hex$(Db.TableDefs ( i% ) . Attributes)

        If ( Db.TableDefs ( i% ) . Attributes And DB_SYSTEMOBJECT ) <> 0 Then
                Debug.Print "Sistema de objetos"
        End If
        If ( Db.TableDefs ( i% ) . Attributes And DB_ATTACHEDTABLE ) <> 0 Then
                Debug.Print "Tabla unida"
        End If
        If ( Db.TableDefs ( i% ) . Attributes And DB_ATTACHEDODBC ) <> 0 Then
                Debug.Print "Tabla ODBC unida"
        End If
        If ( Db.TableDefs ( i% ) . Attributes And DB_ATTACHEXCLUSIVE ) <> 0 Then
                Debug.Print "Tabla unida abierta en modo exclusivo"
        End If
        If ( Db.TableDefs ( i% ) . Attributes And DB_ATTACHSAVEPW ) <> 0 Then
                Debug.Print "Tabla unida con un PWD linkeado"
        End If
        Next i%
        End Sub
```

Analizando definiciones de los campos

Los detalles de cada campo están guardados en la colección **Fields**. Recuerda que la colección **Fields** es un miembro de la colección **TableDefs**, así que tendrás que indicar a cual tabla te estas refiriendo en tu código.

Como la **TableDef**, cada objeto **Field** puede ser referido por un número o por un nombre. La propiedad **Count** de la colección de los **Fields** contiene el número de campos de la base de datos. La siguiente tabla muestra las propiedades **Field** más usadas comúnmente.

Propiedades	Descripción
Attributes	Banderas opcionales.
CollatingOrder	Campos ordenados (comparados). El valor regresado en esta propiedad puede ser pasado dentro de las funciones **StrCmp** o **InStr**.
Name	Nombre del campo
OrdinalPosition	Ordenamiento en que los campos fueron agregados a la **TableDef**.
Size	Máximo tamaño de este campo.
SourceField	Nombre del campo en la tabla unida.
SourceTableName	Para tablas unidas, el **TableName** para bases de datos externas o bases de datos ODBC
Type	Tipo de datos para el campo.
Value	Contiene los datos para este campo cuando el campo es parte de un objeto **Recordset**. No referenciado cuando mapea

Para asegurar que Visual Basic construya una estructura **Field** para tu aplicación, debes ejecutar un método **Refresh** antes de tratar de acceder a alguna de las propiedades. Note que cuando se mapea un objeto **Field** del **TableDefs**, el valor no esta disponible, solo esta disponible cuando el objecto **Field** esta referido para un **Dynaset**, **Table** o **Snapshot**.

El siguiente ejemplo muestra una lista de propiedades **Field** para una llamada a una tabla:

```
Dim Fld As Fields
Set Fld = Td ( "Authors" ) . Fields

For i% = 0 To Fld . Count - 1
        Print "Name:", Fld ( i% ) . Name
        Print "Type Code:", Fld ( i% ) . Type
        Print "Size:", Fld ( i% ) . Size
        Print "Attribute Bits:", Hex$ ( Fld ( i% ) . Attributes )
        Print "Collating Order:", Fld ( i% ) . CollatingOrder
        Print "Ordinal Position:", Fld ( i% ) . Ordinalposition
        Print "Source Field:", Fld ( i% ) . SourceField
        Print "Source Table:", Fld ( i% ) . SourceTable
Next i%
```

En el ejemplo anterior, la tabla "Authors" es llamada por un índice de la colección **TableDefs**. Nota que pudiste haber usado el número **TableDef**, como en el ejemplo anterior.

Usando el método List para mapear objetos Database

Otro método para mapear un objeto **Database** es usar un método **List**. Cada uno de estos métodos crea un **Snapshot** que tu puedes manipular para mapear tus tablas de bases de datos, preguntas, índices , y parámetros en las variables **QueryDef**.

Otra manera de mapear una base de datos, es usar el método **List** mostrando el nombre de la pregunta y parámetros en las variables **QueryDef**.

Tu puedes usar dos métodos **List** con el objeto **Database**:

- El método **ListTables** : Lista los objetos **Database**, **Table** y **Query**.

- El método **ListParameters** : Puede ser usado con variables **QueryDef** para crear una variable **Snapshot** conteniendo un registro por cada parámetro en un **QueryDef**.

También puedes usar los dos siguientes métodos en contraste con los objetos **Table** y **Recordsets**:

- El método **ListIndexes** : Lista los índices para objetos **Table**.

- El método **ListFields** : Lista los campos para un objeto **Recordset**.

Para mas información acerca de estos métodos, busca en la ayuda el nombre de cada método.

Usando el método ListTables

El método **ListTables** te permite ejecutar operaciones en todas las tablas, en preguntas en una base de datos, o presentar las tablas o preguntas en una lista a el usuario. Puedes hacer esto aun si las tablas o preguntas son agregadas o borradas después de que hayas escrito tu código.

El método **ListTables** retorna a una variable **Snapshot** que contiene un registro para cada tabla o pregunta en la base de datos. Los campos en cada grabación contienen la siguiente información.

Campo	Contenido	Tipo de dato
Attributes	Atributos de la propiedad **TableDef**	**Long**
DateCreated	Día en que la tabla o la pregunta fue creada.	**Date/Time**
LastUpdated	Día en que la tabla o diseño de la pregunta fue cambiada.	**Date/Time**
Name	Nombre de la tabla o pregunta	**String**
RecordCount	Número de grabaciones en la tabla	**Long**
TableType	Una de las constantes que identifica el tipo de tabla o pregunta: **DB_TABLE** (tabla nativa) **DB_QUERYDEF** Todos los atributos constantes **TableDef**	**Long**

Nota : Las variables **Snapshot** regresadas por los métodos **List** no soportan algunos de los métodos que otras variables **Snapshot** hacen. En particular, tu no puedes usar los **ListField**, **CreateSnapshot**, o encontrar métodos en un **Snapshot** regresado por el método **List**. Los métodos **Chunk** no son soportados en ninguno de estos objetos, porque ninguno de los métodos **List** produce un **Snapshot** conteniendo un largo binario o un campo Memo.

Usando el método ListParameters

Tu puedes obtener los parámetros que una pregunta requiere para usar el método **ListParameters** en una variable **QueryDef**. El método **ListParameters** regresa una variable **Snapshot** con un registro para cada parámetro de uso de una pregunta.

Para más información sobre como usar el método **ListParameters**, ver el capítulo **IV** "Creación de programas para manipular datos".

III.3 Creando una base de datos

Si todavía no tienes una base de datos, tu puedes usar Visual Basic para crear una.

Hay cuatro métodos diferentes para crear una base de datos:

• Uso de la aplicación del manejador de datos "Data Manager": Con el manejador de datos, puedes crear una base de datos con formato Visual Basic/Microsoft Access.

• Uso de la función **CreateDatabase**: Tu puedes usar **CreateDatabase** cuando tu quieras, para proporsionarte una gran profundidad de control programático sobre una base de datos. La función **CreateDatabase** soporta solo formatos Visual Basic/Microsoft Access.

• Usa Microsoft Access, una base de datos externa o una aplicación ODBC para crear una nueva base de datos externa. Para muchos formatos de bases de datos externas, Visual Basic considera un directorio para ser una base de datos, así tu puedes crear una nueva base de datos externa creando un directorio en tu disco.

• Usa la función **OpenDatabase** para crear una nueva base de datos externa.

Nota : Para crear una base de datos ODBC, debes usar herramientas de la base de datos proporcionadas por un manejador ODBC.

La siguiente sección discute solo la función **CreateDatabase**. Para mayor información acerca de la aplicación Data Manager, ver la ayuda en línea de el Data Manager.

Usando la función CreateDatabase

La función **CreateDatabase** crea un nuevo formato de archivo de bases de datos Visual Basic/Microsoft Access y retorna un objeto **Database** que tu puedes usar para manipular nuevamente la base de datos creada. La función **CreateDatabase** incluye tres partes, como se muestra en la siguiente sintaxis :

Set Database = CreateDatabase (Databasename, Locale [,options])

La nueva base de datos a sido abierta en modo exclusivo, así otros usuarios no les será permitido el acceso hasta que tu hayas finalizado lo que estabas creando. Tu puedes crear una nueva base de datos en tiempo de diseño o programáticamente en tiempo de corrida. Una vez que la base de datos ha sido creada, tu necesitarás agregar las definiciones de las tablas y del índice como se describió anteriormente.

El siguiente procedimiento muestra como crear una base de datos usando la función **CreateDatabase**.

Para crear una base de datos :

1.- Usa la sentencia **Dim** para crear una nueva variable objeto **Database**. Por ejemplo, el siguiente código crea la variable **Database** MyDB (de una base de datos).
Dim MyDB **As Database**

2.- Usa **Dim** y el parámetro **New** para crear objetos **TableDef**, **Field** e **Index**. Necesitas un **TableDef** para cada tabla, un **Field** para cada campo en cada tabla, y un **Index** para cada índice en cada tabla. Por ejemplo, para crear una tabla con tres campos y dos índices, podrías usar el siguiente código :
Dim NewTd **As New TableDef**
Dim F1 **As New Field**, F2 **As New Field**, F3 **As New Field**
Dim I1 **As New** Index, I2 **As New** Index

3.- Usa la sentencia **Set** y la función **CreateDatabase** para crear una nueva base de datos. En este ejemplo, la función tiene dos partes: una para especificar el nombre de la base de datos y otra para especificar la localidad.
Set MyDB = **CreateDatabase** (" C:\MyDB.mdb", **DB_LANG_SPANISH**)

Usando la parte Databasename de CreateDatabase

La parte **Databasename** es una cadena que especifica la ruta completa y nombre de el archivo de la base de datos o directorio. Si tu red soporta nombres de archivos del tipo Universal Naming Convention (UNC), tu puedes usar esto para direccionar tu base de datos.

Usando la parte Locale de CreateDatabase

La parte **Locale** especifica el lenguaje y código de información usado por la base de datos. Visual Basic usa esta información para determinar la secuencia de comparación utilizada cuando se ordenan los datos. Este ajuste puede ser cambiado solo por recreación de la base de datos o por uso de la función **CompactDatabase**. No existe ajuste por default, así que tu puedes proporcionar una de las constantes validas como se muestra en la siguiente tabla.

Constantes	Localidad
DB_LANG_DUTCH	Dutch
DB_LANG_GENERAL	English, German, French
DB_LANG_ICELANDIC	Icelandic
DB_LANG_NORWDAN	Norwegian, Danish
DB_LANG_SPANISH	Spanish, Italian
DB_LANG_SWEDFIN	Swedish, Finnish

Usando la parte Options de CreateDatabase

Si tu quieres crear una base de datos encriptada, o crear una base de datos que pueda ser leída por Microsoft Access versión 1.0, tu debes incluir la parte Options. (El ajuste por default es que no esta encapsulada y un formato de base de datos compatible es Microsoft Access 1.1). Los ajustes **Options** se muestran en la siguiente tabla :

OPCION	VALOR	PROPOSITO
DB_ENCRYPT	2	Nueva base de datosencapsulada.
DB_VERSION10	1	Crear base de datos Microsoft Access 1.0

Por ejemplo, para crear una nueva base de datos encapsulada llamada "Business.mdb" que puede ser manipulada por Microsoft Access versión 1.0, usa la parte **Locale** en "Dutch", tienes que usar el siguiente código:

```
Dim Db As Database
Dim Opt As Long

Opt = DB_VERSION10 + DB_ENCRYPT
Set Db = CreateDatabase ( "Business.mdb" , DB_LANG_DUTCH , Opt)
```

Cerrando una base de datos

Cuando terminas de usar una base de datos, debes cerrarla. Para cerrar una base de datos, usa el método **Close**:

```
MyDB.Close
```

Creando una base de datos externa

Para crear una base de datos con un archivo de formato externo, usa la función **OpenDatabase** en conjunción con una sentencia MKDIR en MS-DOS. Muchas bases de datos son simplemente directorios en tu disco. Una vez que existe un directorio, usando **OpenDatabase** con el nombre del directorio en la parte **Databasename** de la función, podrás crear nuevas tablas en el objeto **Database** en tu código. Los parámetros usados en la función **OpenDatabase** para crear una nueva base de datos son los mismos que usas para abrir una base de datos externa.
Para crear una nueva base de datos ODBC accesible, debes utilizar los procedimientos establecidos por el fabricante de tu base de datos externa. Una vez que haya sido creada la nueva base de datos o establecido un nuevo servidor ODBC, necesitas registrar el nombre de la fuente de los datos en el archivo ODBC.INI . Esto puede ser hecho de varias maneras, incluyendo el uso de las utilidades proporcionadas por el ODBC o la función **RegisterDatabase** para tu aplicación .

Para mayor información acerca de la función **RegisterDatabase** busca en la ayuda para **RegisterDatabase**. Para mayor información acerca del uso de bases de datos ODBC, busca en los archivos de ayuda ODBC proporcionadas por Visual Basic.

Borrando una base de datos

Las bases de datos Visual Basic y todas las estructuras y componentes se encuentran en un solo archivo, tu puedes borrar la base de datos borrando el archivo con extensión .mdb . Para borrar otros archivos de base de datos, tu puedes borrar todos los archivos asociados con la base de datos. Para borrar una base de datos ODBC, debes someter las sentencias SQL correctas para el proceso externo.

III.4 Modificando la estructura de una base de datos

Una vez que ha sido creada la base de datos, o tienes una base de datos que necesita ser modificada, puedes usar Visual Basic para cambiar la estructura. Antes de intentar modificar la estructura de una base de datos, debes iniciar por cerrar cualquier variable **Recordset** abierta .

En general, puedes agregar nuevos objetos **TableDef** a una base de datos, o agregar nuevos objetos **Field** y objetos **Index** a tablas existentes. También puedes borrar un **TableDef** de una base de datos, o borrar un **Index** de un **TableDef**. Ciertas restricciones se aplican para el borrado de objetos **Field** . Para mayor información, ver "Cambiando o borrando un campo", mas adelante en este capítulo.

Usando métodos de Colecciones

Una colección es un juego de variables objeto. Puedes usar los métodos de colección para agregar o borrar variables objeto de una base de datos. Visual Basic soporta tres tipos de colecciones:

- La Colección **TableDef** mantiene una descripción de cada tabla en la base de datos. Descripciones de tablas se encuentra mantenidas en objetos **TableDef** como miembros de la colección **TableDefs**.

- La colección **Fields** mantiene los juegos de todos los objetos **Field** para una **TableDef**.

- La colección **Indexes** mantiene los juegos de todos los objetos **Index** asociados con un **TableDef**.

Los objetos en una colección están referidos como miembros de la colección. Cuando tu quieres agregar miembros o borrar miembros de una colección, usa los métodos de la colección. Estos métodos solo cambian la estructura de la base de datos, y los datos se encuentran sin cambio alguno. Por ejemplo, si tu usas el método **Delete** contra una tabla local en una colección **TableDefs**, no solo la estructura de la tabla será removida, los datos que se encuentran en la tabla se perderán permanentemente. Los métodos de colecciones trabajan en forma diferente con tablas externas. Por ejemplo, usando el método **Delete** contra una tabla externa que ha sido unida a una base de datos local simplemente ligada a la tabla; la tabla no es afectada.
La siguiente sección describe los métodos de colección soportados.

Método Append

El método **Append** agrega un miembro a la colección que tu especifiques, si existe un miembro con el mismo nombre en la colección, un posible error es generado . Un posible error es también generado si los nombres **Field** especificados para una propiedad **Index** de un campo no existe, o si la propiedad **Size** no es del tipo cuando uses el método **Append** contra la colección **TableDef**.

Por ejemplo, el siguiente código agrega un nuevo objeto **Field** a la colección de una nueva tabla :

```
Dim F1 As New Field
Dim NewTd As New TableDef
F1 . Name = "AU_ID"
F1 . Type = DB_LONG                    ' Este es el Long en Tabla Author
NewTd . Fields . Append F1             ' Agregar este a la colección del campo.
```

Método Delete

El método **Delete** remueve un miembro de la colección que tu especifiques. Si no hay ningún miembro en la colección con el nombre especificado (o número), un posible error es generado. Por ejemplo, el siguiente código remueve un índice llamado por la colección **Indexes** :

Db.**TableDefs**("Author Address").**Indexes**.**Delete** "Area_code_Index"

El método Refresh de un Data Control

El método **Refresh** puede ser aplicado solo para el Data Control. Tu puedes usar el método **Refresh** para inicializar la apertura de un Data Control de la base de datos y para crear un **Recordset**. También puedes usar **Refresh** para asegurar que el **Recordset** creado por el Data Control tiene información común. Desde una base de datos puede ser sometido un cambio por otros usuarios o por tu propia aplicación, tu puedes utilizar el **Refresh** para renovar periódicamente los contenidos de los **Recordset**. El método **Refresh** llevará a cabo diferentes series de acciones en tu **Recordset**, dependiendo cuales propiedades de el Data Control hayas cambiado la última vez que renovaste tu información .

Cambio esta propiedad	Acción de el método Refresh
Connect	**Reabre Recordset y Database. Cuando-**
DatabaseName	**reabres, el Recordset es reidentificado-**
Exclusive	**con nuevas-**
ReadOnly	**propiedades.**
Options	Reidentifica solo el **Recordset**
(Ninguna)	Reidentifica el mismo **Recordset**

Nota : Si se ha cambiado las propiedades del Data Control, el **Recordset** es creado con las propiedades existentes. El número de miembros y orden del **Recordset** puede cambiar para reflejar el estado común de la base de datos .

Uniendo tablas de bases de datos externas

Visual Basic puede unir tablas de base de datos externa soportadas. Visual Basic soporta comúnmente los siguientes formatos de bases de datos :

- Microsoft Access
- Btrieve (Con archivos File.ddf y Field.ddf).
- dBase III y dBase IV.
- FoxPro y FoxPro para windows.
- Paradox.
- Bases de datos SQL, tales como SQL Server y Oracle.

Generalmente, tienes dos maneras de escoger o seleccionar, fuentes de datos externas. Puedes dejar los datos en la base de datos fuente y unir tu aplicación a la tabla fuente, o puedes traer los datos dentro de tu base de datos Visual Basic usando programas lógicos.

Si dejas los datos en tablas unidas, la base de datos fuente continuará con el manejo y compartiendo estos datos . Puedes usar tablas unidas como cualquier otra tabla en tu base de datos Visual Basic, y puedes usar estas mientras están siendo compartidas por otras aplicaciones usando el mismo servidor. También puedes combinar operaciones que incluyan datos externos de tablas unidas con datos almacenados en la base de datos local. Si deseas también puedes acceder a datos de tablas externas a tablas Visual Basic.

Cuando estas uniendo una tabla que requiera un password, puedes ligar el password dentro de la información. Esto te permitirá acceder a la tabla basada en las reglas lectura/escritura establecidas en tu aplicación, sin tener que proveer un password. Para salvar el password en la información encadenada, ajusta el bit **DB_ATTACHSAVEPWD** en la propiedad **TableDef**.

Otra opción que puedes usar cuando uses tablas externas, es la propiedad del atributo **DB_ATTACHEXCLUSIVE**. Si este bit esta colocado cuando la tabla esta siendo definida, la tabla estará siempre abierta en modo exclusivo. Esto restringirá a otros usuarios el acceso a la tabla mientras tu base de datos esta abierta .
Para mayor información, ver el apéndice C, "Acceso a bases de datos externas".

Creando una nueva tabla de definiciones

El siguiente ejemplo ilustra como crear una nueva tabla, poblar los campos y construir nuevos índices (Index).

El ejemplo agrega un nuevo objeto **TableDef** "NewTd" a una base de datos creada "Test.mdb". El método **Append** primero agrega nuevos objetos **Field** e **Index** a la **TableDef**, entonces subsecuentemente agrega la nueva **TableDefs** a la base de datos de la colección **TableDefs**.

```
Dim Db As Database
Dim NewTd As New TableDef              ' Nuevo objeto TableDef.
Dim NewIx As New Index                 ' Nuevo objeto Index.
Dim F1 As New Field, F2 As New Field   ' Nuevo objeto campo .
Dim F3 As New Field, F4 As New Field   ' Nuevo objeto campo .
Dim F5 As New Field, F6 As New Field   ' Nuevo objeto campo .

Set Db = CreateDatabase ("C:\Test1.mdb" , DB_LANG_GENERAL)
        If Db Is Nothing Then MsgBox "No se creo la base de datos": Exit Sub
        If Err <> 0 Then MsgBox "Error cuando se abria:" & Err & "/" & Error$ (Err)
On Error GoTo 0

NewTd . Name = "Author Address"              ' Nombre de la nueva tabla
F1 . Name = "AU_Id"
F1 . Type = DB_LONG                  ' Este es un Long en la tabla Authors

NewTd . Fields . Append F1                   ' Agregando esto a la colección.
F2 . Name = "Address"
F2 . Type = DB_TEXT
F2 . Size = 40
NewTd . Fields . Append F2
F3 . Name = "City"
F3 . Type = DB_TEXT
F3 . Size = 20
NewTd . Fields . Append F3
```

```
F4 . Name = "State"
F4 . Type = DB_TEXT
F4 . Size = 2
NewTd . Fields . Append F4
F5 . Name = "Zip"
F5 . Type = DB_TEXT
F5 . Size = 12
NewTd . Fields . Append F5
F6 . Name = "Date Changed"
F6 . Type = DB_DATE

NewTd . Fields . Append F6
NewIx . Name = "Author_ID_Zip_Index"
NewIx . Fields = "AU_ID; ZIP"
NewIx . Primary = True
NewTd . Indexes . Append NewIx

Db . TableDefs . Append NewTd
Db . Close
```

Agregando una tabla a la base de datos

Para agregar una tabla a la base de datos, crea una nueva **TableDef** a la colección existente **TableDefs**. Por ejemplo, refiere las instrucciones para crear una tabla con campos e índices. El siguiente ejemplo muestra las líneas de código más importantes:

```
Dim db As Database
Set db = OpenDatabase ("Test1.mdb")
Dim NewTd As New TableDef              ' Creando un nuevo objeto TableDef
Dim NewFld As New Field                ' Creando un nuevo objeto Field

NewTd . Name = "New Table Name"
NewFld . Name = "New Field Name"
NewFld . Type = DB_INTEGER
NewTd . Fields . Append NewFld
db . TableDefs . Append NewTd          ' agregando un TableDef a la colección.
db . Close
```

Agregando un campo a la tabla

Tu puedes agregar un campo a una tabla existente, agregando un campo a una colección de **Fields** existentes. El siguiente código agrega dos nuevos campos, "Area Code" y "Phone" a la dirección de la tabla "Author".

```
Dim Db As Database
Dim Td As TableDef
Dim F1 As New Field , F2 As New Field      ' Nuevas variables Field.
Set Db = OpenDatabase ("C:\Test.mdb, True)
                                          ' Abriendo la base de datos
                                          ' Para usarla de modo exclusive.
```

```
F1 . Name = "Area Code"
F1 . Type = DB_TEXT
F1 . Size = 3
F2 . Name = "Phone"
F2 . Type = DB_TEXT
F2 . Size = 25

Db . TableDefs ("Author Address") . Fields . Append F1
Db . TableDefs ("Author Address") . Fields . Append F2

Db.Close
```

Borrando una tabla

Para borrar un tabla, usa el método **Delete**, borrando una tabla local consecuentemente borrarás todos los campos, índices y datos contenidos en la tabla; debes usar este método con cuidado. El siguiente ejemplo borra la tabla "Author Address" :

```
Db.TableDefs.Delete "Author Address"
```

Cambiando o borrando un campo

No puedes borrar o cambiar un **Field** (campo) individual de un **TableDef**. Para borrar campos individuales de una tabla, debes construir una nueva tabla que no incluye los campos a ser borrados. Entonces tu puedes mover todos los datos hacia la nueva tabla y borrar la tabla anterior. También puedes necesitar crear una nueva tabla y copiar todos los datos cuando quieres cambiar el tamaño o tipo de los campos definidos en una tabla.

Del mismo modo, no puedes borrar un **Field** (campo) individual miembro de una colección **Field**. Para cambiar las propiedades de un objeto **Field**, debes borrar la **TableDef** y reconstruirla. Por supuesto; esto significa que puedes extraer cualquier dato almacenado ahí y recargar la tabla reconstruida con los datos originales, siempre que proporciones una rutina de conversión que reconozca los nuevos objetos **Field**, tipo de datos, tamaño u otras diferencias.

Trabajando con la propiedad Field

La siguiente información discute las propiedades individuales de el objeto **Field**. Nota que ninguna de estas propiedades puede ser cambiada una vez que la tabla a sido creada.
Visual Basic provee varios caminos para direccionar las propiedades de los objetos **Field** con colecciones **Fields**. También puedes utilizar estas técnicas para direccionar objetos **Index** con colecciones **Indexes**. Para mayor información sobre como direccionar propiedades de estos objetos, ver el capítulo **IV** "Creando programas para manipular datos".
La siguiente información describe las propiedades especificas de un objeto **Field**.

Propiedad Name

La propiedad **Name** determina el nombre del campo. Esta propiedad esta puesta cuando la base de datos es creada o modificada y no puede ser cambiada. El nombre del campo esta conformado para los estándares de una base de datos especifica. En las bases de datos con formato Visual Basic/Microsoft Access, los nombres de **Field** están limitados a 64 caracteres, cada nombre debe ser único dentro de la colección **Field**.

Propiedades Type y Size

La propiedad **Type** determinan en el objeto **Field** el tipo de datos. Por ejemplo, este valor indica si el campo es usado para almacenar números, caracteres, o información binaria. Esta propiedad es enviada cuando la base de datos es creada o modificada y no puede ser cambiada a menos que estés creando un nuevo **Field**.

La propiedad **Size** determina cuanto puedes guardar (en bytes), este **Field** se usará en la base de datos. Para más tipos de datos, este es un valor fijado automáticamente por Visual Basic que no puede ser alterado. Para objetos **Field** que pueden tener una variable ancha (como texto y campos de memoria), la propiedad **Size** indica la cantidad máxima de almacenamiento que este **Field** ocupará.
Valores validos para la propiedad **Type**, significado y tamaño, son mostrados en la siguiente tabla:

Tipo de Constante	Valor	Tamaño	Uso
DB_BINARY	9	0	Binary (no soportado)
DB_BOOLEAN	1	1	valores True/False
DB_BYTE	2	1	Numero (TinyInt)
DB_CURRENCY	5	8	Número (Currency)
DB_DATE	8	8	Date/Time
DB_DOUBLE	7	8	Numero (Double)
DB_INTEGER	3	2	Numero (Integer)
DB_LONG	4	4	Numero (Long)
DB_LONGBINARY	11	0	objetos Binary
DB_MEMO	12	0	Campos Memo.
DB_SINGLE	6	4	Numero (single)
DB_TEXT	10	n	Carácter (String).

Nota que la propiedad **Size** esta ajustada a 0 para constantes binarias (binary) y campos de memoria. La cantidad de datos almacenados en estos campos esta en función de la capacidad de la computadora a la que estas conectado.

La limitación interna en datos tipo **String** en Visual Basic es de 64K. En el servidor Microsoft SQL (conectado vía ODBC), aproximadamente 2 GBytes pueden ser almacenados en los campos de memoria. Para campos especialmente largos necesitas usar los métodos descritos en la siguiente tabla para recuperar o ajustar los contenidos del campo.

Métodos	Descripción
AppendChunk	Añade una cadena a el final del campo Memo.
FieldSize	Regresa el tamaño del campo.
GetChunk	Accede al campo Memo.

Para mayor información acerca de estos métodos, ver la sección "Manejo de campos largos" en el capítulo IV, "Creando programas para manipulación de datos".

Propiedad Attributes

La propiedad **Attributes** determina como un campo será tratado por Visual Basic. Nota que estos son valores bit y desarrollaras operaciones sobre la propiedad **Attribute** para determinar el valor de cualquier bit escogido. La siguiente tabla lista los posibles valores para la propiedad **Attributes**.

Constantes	Valor	Proposito
DB_AUTOINCRFIELD	16	Indica que el campo es un tipo de dato contador. Visual Basic actualiza estos campos automáticamente.
DB_FIXEDFIELD	1	Indica que tipo de datos hay en los campos. (no Text, Memo, o Binary.)
DB_UPDATABLEFIELD	32	Indica que campos pueden ser modificados.

Valores bit para la propiedad **Attributes** son puestos por la aplicación que crea o modifica la base de datos. Cuando instalas tu propia base de datos, puedes probar o ajustar uno o más de estos campos bit para determinar como Visual Basic manejará tu objeto **Field**. Por ejemplo, si quieres establecer un campo **Counter** que es automáticamente incrementado cuando nuevos registros son agregados, ajusta el bit **DB_AUTOINCRFIELD** en la propiedad **Attributes**. Campos **Counter** son usados en casos donde no tienes ninguna otra llave definida para una tabla.

CollatingOrder

La propiedad **CollatingOrder** determina como un campo es ordenado. Visual Basic soporta un número de sentencias **Collating** que tu puedes usar para cambiar el orden en el cual los datos de los registros son regresados después de un proceso (pregunta). Nota que este puede ser un posible valor **Null** (Nulo), así tu código necesitará tratar con esta contingencia. El valor por default esta basado en el parámetro **Locale** usado cuando la base de datos fue creada. La siguiente tabla lista los valores de la propiedad **CollatingOrder**.

Valor	Significado
NULL	**CollatingOrder** no tiene aplicación para este campo (tipo numérico).
0	El campo es ordenado sobre una estricta base binaria.
256	El campo es ordenado con reglas EFGPI (English, French, German, Portuguese, Italian).
257	El campo es ordenado con reglas Microsoft Access V 1.0.
258	El campo es ordenado con reglas Spanish.
259	El campo es ordenado con reglas Dutch.
260	El campo es ordenado con reglas Swedish y Finnish.
261	El campo es ordenado con reglas Norwegian y Danish.
262	El campo es ordenado con reglas Icelandic.
-1	El campo es ordenado con reglas desconocidas.

OrdinalPosition (Posición Ordinal)

La propiedad **OrdinalPosition** determina el orden lógico en el cual los campos serán regresados a tu aplicación cuando un **Recordset** es creado. Cuando tu creas por primera vez una tabla, el valor de la **OrdinalPosition** esta basado en el orden en el cual los objetos individuales **Field** son agregados a el objeto **TableDef**. Por ejemplo, si tu construyes un **Recordset** usando una pregunta SQL para regresar todos los campos a una tabla, los campos deberán ser regresados en el orden que especifique **OrdinalPosition** .

Propiedades SourceTable y SourceField

Las propiedades **SourceTable** y **SourceField** son valores de solo lectura que indican la fuente de la tabla original y campo fuente para datos de tablas unidas o fuentes de datos externas. Esta información es útil si el nombre original fue cambiado en una pregunta SQL y la fuente original de los datos no es evidente.

Creando nuevos Indexes (índices)

Agregar un **Index** (índice) a tu base de datos puede incrementar la velocidad con la cual obtienes acceso a la información. Un índice trabaja buscando los valores dentro de un campo por el método **Sort**. Una tabla puede tener hasta 10 índices, y cada índice puede referir uno o mas campos o llaves.

Cada índice en la tabla esta definido por un objeto **Index**; Los detalles de cada objeto **Index** están contenidos en la colección **Indexes**. Recuerde que la colección **Indexes** es también un miembro de la colección **TableDefs**.

Por ejemplo, el siguiente código muestra como podrías crear un simple índice para una nueva tabla :

```
Dim Td As New TableDef
Dim Ix1 As New Index
Td . Name = "New Table"
Ix1 . Name = "Table Index One"
Ix1 . Unique = True
Tx1 . Primary = True
Ix1 . Fields = "Name; [Last Name]; City; [Zip Code]"
Td . Indexes . Append Ix1
```

Nota que los campos llave son llamados Name, Last Name, City y Zip Code. Desde que la propiedad **Unique** esta ajustada a **TRUE** (verdadero), dos registros con la misma llave de campos no se permitirán existir en esta tabla. Desde que la propiedad **Primary** es **True** (verdadera), cada uno de los campos llave deberán contener algunos valores (aun espacios), o el registro no será alojado. En otras palabras no deberás permitir agregar registros que tienen un valor **Null** (Nulo) en un campo **Primary** (primario). Si Visual Basic descubre un registro duplicado o un valor Nulo en un campo **Primary** (primario) en el proceso de creación de un índice, el índice no será construido y tu aplicación generará un posible error.

Agregando un índice a la Tabla

Puedes usar el método **Append** para agregar un nuevo objeto **Index** a una colección **Indexes**. El siguiente ejemplo muestra como agregar un **Index** (índice) adicional a la tabla "Author Adress":

```
Dim NewIdx As New Index
Dim Db As Database

Set Db = OpenDatabase ("C:\Biblio.mdb", True)    ' Abre la base de datos
NewIdx . Name = "Area_Code_Index"
NewIdx . Fields = "Area Code"
NewIdx . Unique = False
Db . TableDefs ("Author Address") . Indexes . Append NewIdx
Db . Close
```

Borrando un índice

La sintaxis para borrar un **Index** (índice) es similar a la usada para borrar un **table** (tabla). El código en el siguiente ejemplo remueve el **Index** (índice) "Area_Codigo_Indice" de la colección **Indexes** en el **table** (tabla) "Author Adress" :

```
Db.TableDefs ("Author Adress") . Indexes . Delete "Area_Code_Index"
```

Trabajando con propiedades Index

Visual Basic proporciona varios caminos para direccionar las propiedades de objetos **Index** dentro de colecciones **Indexes** Para mayor información sobre como direccionar propiedades de estos objetos, ver el Capítulo 4 "Creando Programas que Manipulan Datos".

Las siguientes secciones describen las propiedades de los objetos **Index**.

Propiedad Name

La propiedad **Name** (Nombre) determina un único nombre para el **Index** (índice).

Propiedad Fields

La propiedad **Field** determina los campos en la tabla que constituye el **Index** (índice) para la tabla de la base de datos. Esta propiedad es una cadena que no debe ser más larga de 254 caracteres. Cada campo llave del **Index** (índice) debe ser un nombre de campo valido. Múltiples campos llave están separados por punto y coma. El **Index** (índice) inducirá a los registro a ser ordenados de manera ascendente por default, si tu especificas un ordenamiento ascendente precedido del nombre del campo con un signo (+) más. Para ordenar un campo en secuencia descendente, debes predecir el nombre del campo con un signo menos (-). Por ejemplo, para especificar un **Index** (índice) con el nombre del campo ordenado en forma descendente, combinado con el campo City ordenado en forma ascendente, puedes escribir el siguiente código :

MyIndex . **Field** = " -Name; +City"

Propiedad Primary (Primaria)

La propiedad **Primary** puede tomar un valor **True/False** que determina si este **Index** es el **Index Primary** (índice primario) para esta tabla. Solo un **Index** puede ser marcado como **primary** (primario). Visual Basic crea un **Index** como la llave primaria (**primary key**) de la tabla y usa este para encontrar un archivo grabado.

La propiedad **Primary** especifica que la propiedad **Fields** determina el nombre de el campo o campos que Visual Basic usa como llave primaria para la tabla. Una llave primaria de una tabla es el factor determinante cuando probamos para ver si el registro es único dentro de la tabla. Ajustando la propiedad **Primary** a **True** (verdadero), implica que este **Index** (índice) es único y que valores no **Null** (nulos) serán aceptados en los campos especificados.

Propiedad Unique (Unico)

La propiedad **Unique** es un valor **True/False** que determina ya sea que quiera o no la base de datos prohibir duplicar registros en la tabla basados en este índice. Si la propiedad **Primary** esta ajustada a **True**, Visual Basic asume que la propiedad **Unique** es también True.

La propiedad **Unique** es usualmente recibida como **Index** (índice) secundario. Un **Index** (índice) secundario es usado para incrementar la velocidad de acceso a la tabla usando uno o mas campos alternativos como llave. Sobre una tabla larga, puedes ajustar un **Index** (índice) secundario sobre uno de los **Fields**. Si quieres asegurar que no haya renglones duplicados basados sobre esta llave secundaria, ajusta la propiedad **Unique** a **True**.

Por ejemplo, si tienes una tabla de "empleados" que tiene un **Index Primary** (índice primario) en el número de empleados, puedes poner un **Index** (índice) secundario en el departamento "empleados", para hacer más facil el acceso a estos campos cuando son creados los **Recordsets**.

Checando las propiedades de los índices

El siguiente código muestra la lista de propiedades del **Index** para la tabla de títulos :

```
Dim Db As Database
Set Db = OpenDatabase ("Biblio.mdb")
Dim Idx As Indexes
Set Idx = Db . TableDefs ("Titles") . Indexes

For i%=0 To Idx . Count -1
        Print "Name:", Idx ( i% ) . Name
        Print "Fields:", Idx ( i% ) . Fields
        Print "Unique:", Idx ( i% ) . Unique
        Print "Primary:", Idx ( i% ) . Primary
Next i%
```

III.5 Manejando una base de datos

Cuando inicias primero en Visual Basic en tiempo de diseño, o cuando cargas una de tus aplicaciones en tiempo de ejecución, necesitas asegurar que ningún otro programa tenga acceso a la información en un archivo de inicialización (.ini). El archivo de inicialización envía los parámetros que controlan el medio ambiente para la aplicación. Cuando tu estas trabajando con acceso a datos, puedes necesitar algunas veces usar una sentencia de inicialización, en adición a un archivo .INI, para que el puntero especifique los requerimiento de la base de datos.

Esta sección discute sentencias de inicialización especial que puedes usar para abrir Visual Basic, Microsoft Access, o bases de datos de formato externo. También puedes encontrar información sobre como puedes compactar y reparar tu base de datos, y como manejar el tiempo ocioso de Visual Basic y como trabaja este con Microsoft Windows .

Abriendo una base de datos Visual Basic o Microsoft Access

Visual Basic proporciona dos sentencias, **SetDataAccessOption** y **SetDefaultWorkSpace**, que puedes utilizar para controlar la inicialización de la base de datos. Necesitas usar estas sentencias de inicialización en las siguientes situaciones :

- Tiempo de diseño, si el archivo de inicialización (Vb.ini) esta almacenado en algún lugar del directorio Windows.

- Tiempo de corrida, si el nombre de tu archivo ejecutable difiere del nombre del archivo de inicialización. Por ejemplo, si tu aplicación es nombrada Biblio.exe, Visual Basic buscará la información de inicialización en Biblio.ini. Visual Basic esperará encontrar estos archivos en el directorio Windows, solo como otro archivo .ini .

- Cuando uses una base de datos externa en un formato que no sea Visual Basic/Formato Microsoft Access. Por ejemplo, formatos tales como : Btrieve, dBase, FoxPro, ODBC y Paradox .

- Si estas usando un Password protegido Microsoft Access.

Debes llamar estas sentencias de inicialización antes de que llames algún otra función o sentencia de acceso a la base de datos. Esto significa que si estas usando un Data Control, solo podrás usar las sentencias de inicialización en el primer evento **Form_Load** o procedimiento **Sub_Main** antes de llamar cualquier otro código de datos tipo objeto.

Inicializando la base de datos

Visual Basic inicializará la base de datos justo antes de que la primer operación de acceso a datos es ejecutada.

- Si estas usando el Data Control, y no hay otras operaciones de acceso de datos en el primer evento **Form_Load** o procedimiento **Sub_Main**, el procedimiento será inicializado inmediatamente después del evento **Form_Load**.

- Si estas usando la función **OpenDatabase** o el método **Refresh**, el procedimiento será inicializado justo antes de que la base de datos es abierta.

Usando la sentencia SetDataAccessOption

La sentencia **SetDataAccessOption** aprueba los parámetros de inicialización a la base de datos Visual Basic. Abajo se muestra esta opcion que puedes ajustar y los efectos sobre la operación de la base de datos.

Opcion	Valor	Propósito
DB_OPTIONINITPATH	1	Determina el nombre y la localización del archivo de inicialización de la base de datos.

La opción **DB_OPTIONINITPATH** nombra el archivo .INI y dice a Visual Basic donde encontrarlo proporcionandole la ruta completa y el nombre del archivo. Una vez que el archivo de inicialización es localizado y enviado, este no puede ser cambiado sin rearrancar la aplicación. Solo podrás ajustar la ruta de inicialización antes de arrancar cualquier operación de acceso a datos. Por ejemplo, la siguiente sentencia especifica la localización de el archivo de inicialización.

SetDataAccessOption DB_OPTIONINITPATH, "C:\MYDIR\MYPROG.INI"

Usando SetDefaultWorkspace para indicar el Usernames y Password

Si tu aplicación es para abrir una base de datos Microsoft Access que tiene un esquema de permisos, necesitas usar la sentencia **SetDefaultWorkspace** para indicar un apropiado nombre de usuario (**user name**) y **password**. Una vez más, esta sentencia puede proceder a cualquier otra sentencia que pudiera principiar la inicialización de la base de datos.

Solo Microsoft Accesss mismo puede ser usado para ajustar o modificar los esquemas de permisos en una base de datos Microsoft Access. Si el archivo correcto es System.mda, Microsoft Access solo te permitirá usar por default el nombre de usuario (**user name**) "Admin" sin **password** para acceder a la base de datos. Puedes especificar en Microsoft Access solo un nombre de usuario (**user name**) y un password en tu programa. A causa de esto, podrás abrir solo bases de datos Microsoft Access que compartan el mismo nombre de usuario (**user name**) y **Password**. Por ejemplo, la siguiente sentencia **SetDefaultWorkspace** indica que el nombre de usuario (**user name**) es "Chrissy" y el **password** es "Soccer Champ".

SetDefaultWorkspace "Chrissy", "Soccer Champ"

Compactando una base de datos

En caso que tu base de datos use términos muy largos, puedes encontrar que el archivo .mdb crece más de lo tu habías esperado, basado en la cantidad de datos que tu hayas agregado y borrado. Para mantener un alto estado de desempeño, Visual Basic pospone o retrasa el traslado de páginas descartadas hasta que tu pares el trabajo de la base de datos y compactes las páginas descartadas. Este diseño mantiene a tu base de datos interactiva con un alto desempeño, a costa de recobrar espacio en disco.

Puedes usar la sentencia **CompactDatabase** para copiar todos los datos de una base de datos dentro de otra, y en el proceso, organizar los datos en la base de datos resultante, adyacentemente el espacio de disco puede ser recuperado. Tu puedes compactar solo bases de datos Visual Basic y formatos Microsoft Access. Un error podrá ser generado si intentas usar esta sentencia sobre alguna base de datos abierta, o en una base de datos que contiene tablas unidas, o en una base de datos de formato diferente a Visual Basic/Microsoft Access.

CompactDatabase : También te proporciona la opción de cambiar el estado de "encapsulamiento", versión, y localización de una base de datos mientras esta siendo compactada. En otras palabras, puedes convertir una base de datos encapsulada a una base de datos no encapsulada, o viceversa. También puedes convertir de una local a otra, o de una versión a otra.

Antes de compactar una base de datos, esta debe ser cerrada. Por otra parte, nunca debes dar los mismos nombres a la fuente y destino para la base de datos que esta siendo compactada, tu base de datos será borrada si la sentencia **CompactDatabase** no es completada. El proceso de compactado no podrá ser hecho dentro de una transacción.

La sintaxis para la sentencia **CompactDatabase** es la siguiente :

CompactDatabase Sourcename, Destinationname [, Locale [, Options]]

Por ejemplo, el siguiente código compacta una base de datos Microsoft Access llamada Pubs.mdb en C:\Directorio y crea una base de datos Visual Basic/Microsoft Access versión 1.1 con un **Locale DB_LANG_SPANISH** y un nombre destino Newspubs.mdb :

CompactDatabase "C:\Pubs.mdb", C:\Newpubs.mdb, **DB_LANG_SPANISH**

La sentencia **CompactDatabse** acepta cuatro partes diferentes :

Propiedad Sourcename

La parte **Sourcename** proporciona la ruta completa y nombre del archivo de la base de datos. Si tu red soporta nombres de archivos UNC, puedes usar esto para direccionar el archivo .

Propiedad Destinationname

La parte **Destinationname** es la ruta completa y el nombre del archivo de la base de datos compactada. Si tu red soporta nombres de archivos UNC, puedes usar esto para direccionar el archivo. No especificar el mismo nombre para ambos archivos fuente y destino .

Propiedad Locale

La parte **Locale** dice a Visual Basic que localice para convertir a. Si **Locale** no es proporcionado, la conversión no tomará lugar. Para convertir tu base de datos **Locale** y codificar los ajustes de página, debes proveer una de las constantes validas **Locale**, como se muestra en la siguiente tabla.

Constante	Locale
DB_LANG_DUTCH	Dutch
DB_LANG_GENERAL	English, German, French
DB_LANG_ICELANDIC	Icelandic
DB_LANG_NORDIC	Nordic countries*
DB_LANG_NORWDAN	Norwegian, Danish
DB_LANG_SPANISH	Spanish, Italian
DB_LANG_SWEDFIN	Swedish, Finnish

Aplicada solo en formato de bases de datos Microsoft Access versión 1.0.

Propiedad Options

Si quieres cambiar el esquema de encapsulamiento o crear una base de datos que puede ser leída por Microsoft Access versión 1.0, necesitas incluir la parte **Options**. La siguiente tabla especifica los valores disponibles.

Opcion	Proposito
DB_DECRYPT	Descompactada la base de datos.
DB_ENCRYPT	Compactar la base de datos.
DB_VERSION10	Crea formato de base de datos Microsoft Access versión 1.0

Por ejemplo, el siguiente código compacta un base de datos Microsoft Access llamada Pubs.mdb en C:\Directorio y crea una base de datos, "Encapsulada" Microsoft Access versión 1.0, con un **Local** en (**English, French, German**) con el nombre C:\Newpubs.mdb :

```
Opt% = DB_VERSION10 + DB_ENCRYPT
CompactDatabase "C:\PUBS.MDB", C:\NEWPUBS.MDB, DB_LANG_GENERAL, Opt%
```

Reparando una base de datos

Si tu base de datos con formato Visual Basic/ Microsoft Access esta dañada, puedes usar la sentencia **RepairDatabase**. Esta sentencia podrá validar, una base de datos que fue invalidada por una operación incompleta de lectura/escritura. Esta clase de daño puede ocurrir cuando el sistema no es parado o apagado normalmente (Tal como una falla de potencia). Recuerda grabar backups de tu base de datos regularmente, para evitar perdida de datos que no pueden ser recuperados con la sentencia **RepairDatabase**.

La sentencia **RepairDatabase** checa todas las páginas en la base de datos para corregir linkeamientos, validar todas las tablas del sistema, y validar todos los índices. Para completar con éxito, la base de datos es entregada, y todas las páginas que no pudieron ser salvadas (tal como invalidar referencias a otras páginas) son descartadas.

RepairtDatabase toma solo una parte de la ruta completa de el archivo de la base de datos que quieres reparar. Los nombres de archivos UNC son asignados si la red los soporta.

Por ejemplo, para reparar la base de datos Biblio.mdb, podrías usar el siguiente código:

RepairDatabase "Biblio.mdb"

Cuando una base de datos es reparada, puede incrementarse su tamaño. Esto es porque el proceso de creación de índices puede dejar algunas páginas borradas en la base de datos. Esto es siempre una buena idea para correr **CompactDatabase** después de alguna reparación, para eliminar páginas innecesarias .

Manejando el tiempo perdido

Multitareas con tu base de datos en tu aplicación y Microsoft Windows dependen por encima de tu aplicación ocasionalmente dando el control al sistema operativo Windows. Tu aplicación automáticamente cede el control cuando Visual Basic llama algunas de las funciones Windows.

También puedes ceder el control ocasionalmente llamando la función **DoEvents**. Una de las tareas que Visual Basic cuida durante este tiempo **"IDLE"** es asegurar que las páginas que estas trabajando están siendo compartidas apropiadamente.

En casos donde tu aplicación esta procesando una gran cantidad de datos sin llamar funciones Windows, otras aplicaciones que están compartiendo la base de datos puede sufrir daños y los datos con los que estas trabajando pueden estar o irse fuera de lugar. En este caso debes llamar la sentencia **FreeLocks**, justo como llamaste la sentencia **DoEvents**.

Capitulo IV

"Creando programas que manipulen datos".

Una vez creada una base de datos, tu necesitas saber como extraer los datos y como manipularlos. Para ayudarte a manejar tus datos, Visual Basic proporciona un comprensivo lenguaje de manipulación de datos. Este lenguaje incluye variables objeto de acceso a datos que representan bases de datos, tablas, preguntas (queries), etc. . Para usar estas variables objeto en tu código, puedes manipular y pasar tablas y preguntas fácilmente, como enteras o cadenas (strings). Usando los métodos y propiedades de las variables objeto, puedes extraer, ordenar, y buscar subgrupos de datos. Puedes pasar a través de registros y agregar, modificar, o borrar sus contenidos; y puedes crear, modificar, y ejecutar preguntas.

Contenido :

IV.1 Creando Recordsets

Uno de los caminos más fáciles para manipular información en tu base de datos es usando una variable **Recordset**. Una variable **Recordset** hace referencia a objetos tales como un **Table**, **Dynaset** o un **Snapshot**. Usa variables **Recordset** para acceder y manipular datos en un objeto. Por ejemplo, puedes usar variables **Recordset** para cambiar el orden de los datos regresados a tu aplicación, seleccionar registros grabados o hacer cambios a los datos. No confundirla con la propiedad **Recordset** de un Data Control, la cual retorna un objeto **Dynaset** con una variable **Recordset**, la cual puede estar en cualquiera de los objetos **Table**, **Dynaset** o **Snapshot**. La siguiente tabla lista las variables **Recordset** y sus diferentes capacidades.

Variable Recordset	Asociación	Registro grabado	Resultado de una pregunta
Table	Puede cambiar	Puede agregar, cambiar o borrar	No
Dynaset	Fijo	Puede agregar, cambiar o borrar	Si
Snapshot	Fijo	Fijo	Si

En el caso de variables **Table**, los cambios, adiciones y borrado de registros que otros usuarios hagan, aparecen inmediatamente. En el caso de variables **Dynaset** o **Snapshot**, las adiciones y correcciones que otros usuarios hacen no aparecen inmediatamente en tu **Dynaset** o **Snapshot**. Cambios que tu y otros usuarios hagan a registros existentes en un **Dynaset** son vistos por quien mueve el registro cambiado.

Tu base de datos proporciona la fuente de datos para variables **Recordset**. Visual Basic trabaja con tu base de datos para definir la información en términos de tres estructuras básicas de datos:

- **Tables** (Tablas). Una tabla es un juego de información almacenado en una base de datos. El termino "tabla base" se refiere a una tabla de una base de datos, mientras que una variable **Table** se refiere a un objeto, el cual directamente se refiere a una tabla de una base de datos.

• **Records** (Registros). Un registro contiene una tabla que almacena información acerca de un artículo individual. Un conjunto de registros (o **Recordset**), es un grupo de registros que tienen todas las características similares. Por ejemplo, un **Recordset** puede contener una lista de todos los empleados que trabajan en una división particular.

• **Queries** (Preguntas). Una pregunta es un juego de registros temporales, definidos por una pregunta en términos de una sentencia definida en base al lenguaje estructurado de preguntas (SQL). Cuando tu creas un **Recordset**, el dato es presentado en una pregunta a tu aplicación como si este fuera una tabla. Las preguntas tiene campos y registros solo como tablas; sin embargo, hay solo una representación temporal de los datos de una o más tablas.

Visual Basic crea variables **Recordset** para usar tablas base en el archivo de base de datos, como la fuente de información. Un **Recordset** puede estar compuesto de cero ó mas registros basados en un número de tablas.

Creando variables Database

Usa la variable **Database** para presentar una base de datos. Todas las funciones que usas con las variables **Recordset** requieren un objeto **Database**, así siempre iniciarás por declarar una variable **Database**. Después de que declaras la variable **Database**, usa la sentencia **Set** para asignar el resultado de la función **OpenDatabase** a la variable. Por ejemplo, el siguiente código abre la base de datos Biblio.

```
Dim db As Database
Set db = OpenDatabase ("Biblio.mdb")
```

Variables **Database** pueden ser creadas sobre cualquiera de las bases de datos soportadas:

• Bases de datos Nativas, tales como Microsoft Access/Visual Basic.
• Bases de datos Externas, tales como Btrieve, dBase, FoxPro ó Paradox.
• Bases de datos ODBC, tales como Microsoft SQL server u Oracle.

Para mayor información sobre como crear y abrir bases de datos, ver el capítulo 3, "Manejo de Bases de Datos usando Visual Basic".

Creando variables Table

Tu puedes declarar una variable **Table** y entonces usarla para acceder a los datos de una tabla base, de una base de datos. Para hacer esto, usa el método **OpenTable** sobre una variable **Database** existente y asigna el resultado a tu variable **Table**:

```
Dim Db As Database, TableVar As Table
Set Db = OpenDatabase ("Biblio.mdb")
Set TableVar = Db . OpenTable ("Titles")
```

La sintaxis para el método **OpenTable** es:

```
OpenTable ( Name [ , Options ] )
```

Esto abre una tabla base para acceso directo. La parte **Name** de la función puede ser el nombre de cualquier tabla base de la base de datos. La parte **Options** controla parámetros tales como lectura/escritura, procedimiento de actualización de la tabla y procesamientos de preguntas SQL. Para mayor información sobre los argumentos que se pueden utilizar en la parte **Options**, ver la sección "Especificando argumentos para métodos **Recordset**". No puedes usar el método **OpenTable** en una tabla ODBC.

Usando la variable Table

Las variables **Table** proporcionan los datos mas actuales, a costo de un poco de flexibilidad. Los datos en una variable **Table** siempre reflejan todos los cambios, incluyendo la adición de nuevos registros o el borrado de registros existentes. El número de registros en la variable **Table** puede cambiar constantemente, reflejando las acciones de otros usuarios. Cuando no puedas usar las propiedades **Filter** o **Sort** sobre una variable **Table**, puedes reordenar los registros en una variable **Table**, usando los índices definidos por la tabla. Cambiando la propiedad **Table.Index**, los registros de la tabla pueden ser inmediatamente accesados en el orden indicado por el nuevo índice. Si el índice no es especificado, los registros son regresados en el orden en el cual fueron ingresados a la tabla.

Usando tablas unidas

En casos donde la tabla de datos esta almacenada en otra base de datos, pero mas o menos permanece ligada aquella tabla (**Table**) y tu base de datos, la tabla se dice estar unida a tu base de datos.

Las tablas unidas son tratadas como si sus datos fueran almacenados como una parte de tu base de datos. No puedes sin embargo, crear una variable **Table** sobre una tabla unida ó sobre una tabla almacenada en una base de datos ODBC.

Para unir una tabla, usa la sentencia **Dim** para crear un nuevo objeto **TableDef**, ajusta las propiedades **Connect** y **SourceTableName** (y opcionalmente la propiedad **Attributes**), y entonces anexala a la colección **TableDefs** para una base de datos dada. Para mayor información, ver el apéndice C, "Accediendo bases de datos externas".

Creando variables Dynaset

Los **Dynasets** proporcionan más flexibilidad que las variables objeto **Table**. Usando el Data Control o una variable **Database** creada con la función **OpenDatabase**, puedes crear un **Dynaset** sobre cualquier tabla, incluyendo una tabla unida, o sobre el resultado de cualquier pregunta. La pregunta puede extraer datos de más de una tabla, incluyendo tablas unidas. De esta forma puedes usar un **Dynaset** para cambiar o unir datos de más de una base de datos. También puedes crear una variable **Dynaset** sobre otra variable **Dynaset**, habilitando el **Filter** mas lejano y ordenando los datos en el **Dynaset**.

Para crear una variable **Dynaset** sobre una tabla en la misma forma que creas una variable **Table**:

```
Dim Db As Database, dsSomeData As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set dsSomeData = Db. CreateDynaset ("Titles").
```

La sintaxis para el método **CreateDynaset** es :

CreateDynaset (Source [, Options])

La parte **Source** para un **Dynaset** puede ser un nombre de una tabla, un nombre de un objeto **QueryDef**, o una pregunta SQL en forma de una cadena. Los nombres de tablas pueden referirse a una tabla local o a tablas unidas. La parte **Options** controla parámetros tales como lectura/escritura, procedimientos de actualización de tablas, y procesamiento de preguntas SQL. Para mayor información sobre **Source** y **Options**, ver la sección "Especificando Argumentos para Variables **Recordset**".

Nota : La propiedad **Recordset** del Data Control también regresa un **Dynaset**. Aunque el editar y actualizar operaciones, están construidas dentro de un Data Control, puedes necesitar agregar rutinas similares para detectar errores en situaciones multiusuario.

Cuando el método **CreateDynaset** ejecuta la pregunta, la variable **Dynaset** retorna el resultado de tu pregunta. En el siguiente ejemplo, el código abre un objeto **QueryDef** llamado "Title Author Publisher" y crea un **Dynaset** de registros resultantes:

```
Dim Db As Database, dsSomeData As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set dsSomeData = Db. CreateDynaset ("Title Author Publisher")
```

También puedes crear una variable **Dynaset** usando una cadena SQL en lugar de el nombre de una tabla existente ó **QueryDef**. Preguntas SQL ó **QueryDefs** pueden referirse a más de una tabla, y cuando se refieren a tablas unidas, ellos pueden referir a más de una base de datos. En el siguiente ejemplo, el código crea un **Dynaset** usando una pregunta SQL, que une datos de dos tablas de la base de datos Biblio.mdb:

```
Dim Db As Database, dsSomeData As Dynaset, SQLQ As String
Set Db = OpenDatabase("Biblio.mdb")
SQLQ = "SELECT Titles.Title, Publishers.Name"
SQLQ = SQLQ & " FROM Titles, Publishers"
SQLQ = SQLQ & " WHERE Titles.PubId = Publishers.PubId"
Set dsSomeData = Db.CreateDynaset(SQLQ)
```

Nota : Cuando creas un **Dynaset** sobre una pregunta o en una cadena SQL, tu código suspende la ejecución hasta que la pregunta retorna el primer registro. Para mayor información sobre creación de preguntas, ver la sección "Usando variables **QueryDef**" más tarde en este capítulo.

Las variables **Dynaset** pueden también ser creadas usando el Data Control. En este caso, el **Dynaset** esta definido por tablas u otras fuentes de datos indicadas en la propiedad **RecordSource** del Data Control. Puedes acceder al **Dynaset** creado por el Data Control usando la propiedad **Recordset**:

```
Dim dsCurrent As Dynaset
Set dsCurrent = Data1. Recordset
```

Si quieres usar un **Recordset** sin usar un **Dynaset**, puedes referir al **Recordset** directamente:

```
FirstColumn$ = Data1.Recordset( 0 )
SecondColumn$ = Data1.Recordset( 1 )
```

Una vez que tienes una variable **Dynaset** (construida de una propiedad **Recordset** o con el método **CreateDynaset**), puedes crear otra variable **Dynaset** sobre esta, usando el método **CreateDynaset** en contra del **Dynaset** original. Usando esta técnica, puedes ajustar las propiedades **Sort** y **Filter** de un **Dynaset**, habilitándo la restricción y reordenamiento de los registros en el nuevo **Dynaset** creado. Por ejemplo, el siguiente código crea una segunda variable **Dynaset** que contiene un subgrupo de los registros del primer **Dynaset**:

```
Dim Db As Database, DsSomeData As Dynaset, DsSorted As Dynaset
Dim SQLQ As String
Set Db = OpenDatabase("Biblio.mdb")
```

```
SQLQ = "SELECT Titles.Title, Publishers.Name "
SQLQ = SQLQ & " FROM Titles, Publishers "
SQLQ = SQLQ & " WHERE Titles.PubId = Publishers.PubId "
Set DsSomeData = Db. CreateDynaset ( SQLQ )
DsSomeData. Filter = "Title LIKE '* SQL* '"          ' Regresa los titulos SQL.
DsSomeData. Sort = "[Name]"                          ' Ordenados por nombre.
Set DsSorted = DsSomeData . CreateDynaset ( )
```

En el ejemplo, si el primer **Dynaset** fue el resultado de una pregunta, entonces al crear el segundo **Dynaset** no exhibe de nuevo la pregunta. El registro que Visual Basic accesa, esta contenido en el primer **Dynaset**. Para mayor información, ver las secciones "Ordenando **Recordsets**" y "Seleccionando Registros".

También puedes usar el método **Clone** para crear una variable **Dynaset** que duplica a una variable **Dynaset** existente. Para mayor información, ver la sección "Variables **Recordset** duplicadas".

Usando variables Dynaset

Los **Dynasets** proporcionan un subgrupo dinámico de todos los registros en una o más tablas. El grupo de registros incluidos en el **Dynaset** es llamado, el conjunto de componentes de el **Dynaset**. Los Filtros son usados para mostrar los datos de un **Dynaset** existente, así que solo los subgrupos de registros deseados son regresados. El orden en el cual estos registros son regresados esta determinado por la propiedad **Sort**, la pregunta SQL, el **Index** usado, o el orden en el cual los registros aparecen en las tablas. Puedes cambiar el orden o el criterio del filtro de un **Dynaset** existente, cambiando primero la propiedad **Sort** o la propiedad **Filter**, creando otro **Dynaset** usando el método **CreateDynaset** contra el **Dynaset** existente.

Una vez creado el **Dynaset**, las propiedades **Sort** y **Filter** son limpiadas y no estarán activas. Si son agregados registros en este punto, ellos están anexados al final del **Dynaset** sin hacer caso del orden **Sort**. Si nuevos registros son agregados a la base de datos o se cambian registros existentes, se descalifican del filtro original y ninguna acción es tomada para excluirlos de tu **Dynaset**.

Cuando un **Dynaset** es creado, se refleja el estado de la base de datos. Puedes agregar a el grupo de elementos de tu **Dynaset** mas registros, y puedes reducir estos grupos de elementos borrando registros. Sin embargo, la adición y borrado de registros en la base de datos hechos por otros usuarios, no afectan tus grupos de objetos **Dynaset**. Aun cuando registros que han sido incluidos son agregados o borrados de la base de datos, tu **Dynaset** no reflejará estos cambios hasta que sean reconstruidos. En otras palabras, una vez que este subgrupo de la base de datos ha sido seleccionado usando el método **CreateDynaset**, estos grupos de elementos permanecen fijos, aun si registros son borrados ó agregados a la base de datos por otros usuarios.

Editando el Dynaset

Cambios hechos a registros en la base de datos son siempre reflejados en todos los **Dynaset** basados sobre aquellos registros. En otras palabras, si editas un registro en tu **Dynaset** y actualizas aquellos cambios usando el método **Update** (o el Data Control), todos los otros usuarios que tienen incluidos aquellos registros en su **Dynaset** verán el cambio hecho cuando se posicionan en aquel registro.

Similarmente, cuando otros usuarios hacen cambios a registros contenidos en tu **Dynaset**, tu verás aquellos cambios cuando te posiciones en el registro cambiado.
Si tratas de usar el método **Edit** para cambiar uno de los registros en tu **Dynaset**, y aquellos registros que han sido borrados o cambiados en la base de datos (quizás por otro usuario), producirán un posible error. Un posible error también resultará si tu mueves o borras un registro en tu **Dynaset**. Si solo un registro ha cambiado, puedes reejecutar la sentencia **Edit**. Si el registro fue borrado, tu tendrás que agregarlo nuevamente a el **Dynaset** con nuevos valores usando **AddNew**.

Manteen en mente que en situaciones multiusuarios, muchos usuarios pueden tratar de agregar, cambiar y borrar los mismos registros, y debes estar preparado para posibles errores que resultan cuando estan trabajando con el **Dynaset**.

Analizando un Dynaset

Cuando manipulas bases de datos de formato Microsoft Acces/Visual Basic, Visual Basic usa un esquema de visualización por páginas. Cada página es de 2 k y contiene muchos registros que se ajustaran en una página. Cuando una página es cerrada, todos los registros sobre aquella página son cerrados hasta que tu o Visual Basic los habrán. Cuando accesas a bases de datos externas o bases de datos ODBC, necesitarás usar otros esquemas para visualizar sus contenidos, como los determinados por la base de datos externa.

El mecanismo por default para el método **Edit** es pesimista (**Pessimistic**), esto significa que la página contiene el registro grabado que podemos observar cuando ejecutamos el método **Edit**. Esto también deja fuera a otros usuarios de la página hasta que ejecutes el método **Update**.

Cuando usas el método Optimista (**Optimistic**) para visualizar los datos, Visual Basic no contiene los registros que esta visualizando de la página de la base de datos cuando, ejecutas el método **Edit**, esto permite a otros usuarios cambiar o borrar los registros. Cuando ejecutas el método **Update**, Visual Basic verifica que el registro que estas editando no ha cambiado en la base de datos, mientras estabas editándolo. Si no hay cambios hechos por otros usuarios, la página es cerrada. Después de hacer los cambios, Visual Basic abre el registro. Si los cambios fueros hechos al registro mientras estabas editándolo, resultará un posible error.

Agregando, cambiando y borrando registros Dynaset

Cuando agregas registros a un **Dynaset**, los registros son siempre agregados en el final de el **Dynaset**, sin hacer caso del orden original. Por otra parte, Visual Basic inmediatamente agrega los registros a la base de datos, a menos que tengas abierta una transacción con **BeginTrans**. Si una transacción esta pendiente, los registros serán agregados a la base de datos cuando la transacción es ejecutada con **CommitTrans**, o descartada con **Rollback**.

Si un nuevo registro viola uno de los índices únicos, o una llave primaria contiene un valor **Null**, un posible error resultará. Esto puede suceder si existe ya un registro con aquel valor del índice de la llave o tu no incluyes todos los campos de un índice primario.

Cuando borras registros de tu **Dynaset**, Visual Basic tambien los borra de la base de datos. Si otros usuarios tienen creados **Dynaset** que comparten los registros que tu borraste, los usuarios no serán notificados a menos que ellos traten de mover o de editar uno de los registros borrados.

Si cambias registros en tu **Dynaset**, los registros cambiados son actualizados en el lugar. Esto es, los registros cambiados permanecen en el **Dynaset** en el orden original, aun si el índice del campo que determino el orden del **Dynaset** fue afectado por el cambio. Por ejemplo, si el orden de tu **Dynaset** fue determinado por el índice de un apellido, y el último **Edit** modificó los apellidos, la localización de aquel registro en el **Dynaset** podría ser afectado. Si reconstruyes tu **Dynaset**, tus datos serán regresados en el orden correcto.

Cuando haces cambios a un **Dynaset** creado sobre los resultados de una pregunta, tus cambios modifican los datos en las tablas sobre los cuales la pregunta (query) fue basada. Si creas un **Dynaset** que comparte registros con otro **Dynaset**, ninguna adición o borrado hecho en un **Dynaset** aparecerá en el otro hasta que los **Dynasets** son reconstruidos. Cualquier cambio hecho en tu **Dynaset** será reflejado en todos los otros **Dynasets**, como la posición de otros usuarios para el registro cambiado. Esto es importante de notar, esto no es común para muchos usuarios que crean **Dynasets** que comparten registros.

Reconstruyendo el Dynaset

Para que este en orden tu **Dynaset** para reflejar las adiciones, cambios y borrados que otros usuarios están haciendo a la base de datos, querrás periódicamente reconstruir tu **Dynaset**. Para hacer esto, ejecuta el método **Refresh** de tu Data Control, borra y vuelve a crear la variable **Dynaset**, o ejecuta el método **CreateDynaset** sin argumentos. Por ejemplo la siguiente sentencia reconstruye un **Dynaset** abierto llamado "MyDynaset":

```
Set MyDynaset = MyDynaset.CreateDynaset( )
```

Hay un límite de número de veces que tu puedes reasignar un **Dynaset**. Existen dos diferentes accesos para manejar los objetos **Dynaset**.

1.- Creando un nuevo **Dynaset** usando la misma variable que contenga el resultado.

```
Dim Ds As Dynaset
Dim Db As Database
Set Db = OpenDatabase ("Biblio.mdb")
Set Ds = Db. CreateDynaset ("Titles")
' Interviene este paso
Set Ds = Ds. CreateDynaset ( )   ' Implicitamente cierra el Ds.
' Interviene este paso
Set Ds = Ds. CreateDynaset ( )   ' Implicitamente cierra el Ds.
' Interviene este paso
Set Ds = Ds. CreateDynaset ( )   ' Implicitamente cierra el Ds.
```

2.- Crea una nueva variable **Dynaset** usando otra variable **Dynaset** para el resultado.

```
Dim  Ds1 As Dynaset, Ds2 As Dynaset, Ds3 As Dynaset, Ds4 As Dynaset
Dim Db As Database
Set Db = OpenDatabase("Biblio.mdb")
Set Ds1 = Db. CreateDynaset("Titles")
' Interviene este paso
Set Ds2 = Ds1. CreateDynaset( )
' Interviene este paso
Set Ds3 = Ds2. CreateDynaset( )
' Interviene este paso
Set Ds4 = Ds3. CreateDynaset( )
```

En el primer caso, donde los objetos **Dynaset** están alojados, podrás reasignar un **Dynaset** a el mismo, cerca de 10 veces. En el segundo caso, encontrarás que puedes volver a crear el **Dynaset** solo 5 veces.

Creando variables Snapshot

Creando una variable **Snapshot** es similar a crear una variable **Dynaset**. Tu puedes crear una variable Snapshot de una tabla o de una pregunta en la base de datos:

```
Dim Db As Database, SnapData As Snapshot
Set Db = OpenDatabase ("Biblio.mdb")
Set SnapData = Db. CreateSnapshot ("Titles")
```

La sintaxis para el método **CreateSnapshot** es:

CreateSnapshot (Source [, Options]).

El **Source** para un **Snapshot** puede ser un nombre de tabla, un nombre de tabla unida, un nombre de objeto **QueryDef**, o una sentencia SQL expresada como una cadena. La parte **Options** controla parámetros como son lectura/escritura, procedimientos de actualización de la tabla , y proceso de preguntas SQL. Para mayor información sobre **Source** y **Options**, ver la sección **"Especificando argumentos para variables Recordset".**

También puedes crear un **Snapshot** de un **Dynaset** existente o de una variable **Snapshot**. No puedes, crear un **Dynaset** de un **Snapshot**. El siguiente código muestra como crear un **Snapshot** de un **Dynaset** existente:

```
Dim Db As Database, dsSomeData As Dynaset
Dim Snap1 As Snapshot, Snap2 As Snapshot
Set Db = OpenDatabase ("Biblio.mdb")
Set dsSomeData = Db . CreateDynaset ("Titles")
dsSomeData . Sort = "[PubID]"
Set Snap1 = dsSomeData . CreateSnapshot ( )
Snap1 . Filter = "[Title] LIKE ' *VB* ' "
Set Snap2 = Snap1 . CreateSnapshot ( )
```

Este código crea una segunda variable **Snapshot** que contiene el mismo conjunto de registros que el primer **Snapshot**. Este segundo **Snapshot** puede ser reordenado cuando lo necesites, cambiando las propiedades **Sort** y **Filter** sobre el primer **Snapshot** y reconstruyendo el **Snapshot** (como ya se a explicado). Si el primer **Snapshot** fue el resultado de una pregunta, entonces el segundo **Snapshot** no puede recibir el resultado de otra pregunta.

Las variables **Snapshot** como las anteriormente descritas, contienen una copia de los datos que existen cuando el **Snapshot** fue creado. Si los datos en tu base de datos son cambiados, estos no son reflejados en el **Snapshot** hasta que el **Snapshot** es reconstruido. Esta puede ser una ventaja, ya que los datos son recuperados cuando se presenta una falla en los datos, siempre que estés trabajando con este. Por ejemplo, si tu haces tus cálculos sobre los datos en un **Snapshot**, tu no tienes ningún problema durante los cálculos.

Tu puedes crear una variable **Snapshot** sobre una tabla, incluyendo tablas unidas, sobre el resultado de una pregunta, sobre un **Dynaset**, o sobre otro **Snapshot**. Tu no puedes ejecutar los métodos **Edit**, **AddNew**, o **Update** mas de una vez sobre un **Snapshot**.

Tu puedes construir un **Snapshot** al mismo tiempo que construyen un **Dynaset**. Ver "Construyendo un **Dynaset**".

Una de las mas significativas diferencias entre un **Snapshot** y un **Dynaset** es que un **Snapshot** regresa el dato seleccionado para tu estación de trabajo (workstation), en cambio un **Dynaset** regresa solo un grupo de llaves que indirectamente se refieren a los registros de la base de datos. Si un **Snapshot** contiene muchas líneas, la estación de trabajo (workstation) tendrá mucha carga de trabajo para manejar los datos. Nota que también los datos de los **Snapshot** pierden rapidez en sistemas multitareas, cuando otros usuarios cambian la base de datos.

Especificando Argumentos para Métodos Recordset

Las siguientes secciones discuten argumentos para los métodos **Recordset**.

Usando el argumento Source

El argumento **Source** especifica la fuente de datos para un método **CreateDynaset** ó **CreateSnapshot**. La fuente puede ser cualquiera de las siguientes:

- El nombre de una tabla base de la base de datos.
- El nombre de una tabla unida.
- El nombre de un objeto **QueryDef**.
- Una sentencia SQL.

Uso del argumento Options

Usa el argumento **Options** para indicar como crear o dirigir una variable **Recordset**. El argumento **Option** se aplican a los metodos **CreateDynaset**, **CreateQueryDef**, **CreateSnapshot**, **OpenTable** y **Execute**.

La siguiente tabla muestra las constantes que puedes usar como argumentos validos para los parámetros **Options**. Nota que puedes agregar múltiples valores de opciones juntas y proporcionales a un solo parámetro.

Opción	Términos
DB_APPENDONLY	Puedes escribir pero no leer del **Recordset**.
DB_CONSISTENT	Las actualizaciones no se haran a todos los campos.
DB_DENYREAD	No permite leer el **Recordset**.
DB_INCONSISTENT	Las actualizaciones se haran a todos los campos.
DB_DENYWRITE	No permite escribir al **Recordset**.
DB_READONLY	Puedes leer pero no escribir a el **Recordset**.
DB_SQLPASSTHROUGH	La pregunta es procesada por una base de datos externa.

Por ejemplo, si quieres abrir un objeto **Table** sobre una tabla **"Titles"** y limitar el acceso para que otros usuarios puedan leer pero no escribir a la tabla, debes ajustar los argumentos de **Options** usando el siguiente código.

```
Dim Options%
Dim Db As Database
Dim Ds As Table
Set Db = OpenDatabase ("Biblio.mdb")
Options% = DB_DENYWRITE
Set Ds = Db.OpenTable ("Titles", Options%)
```

Aplicando las constantes a métodos Recordset

La siguiente tabla resume cuales argumentos son usados con cuales métodos **Recordset**.

La primer tabla muestra cuales leen, escriben y cuales constantes se aplican a los diferentes objetos **Recordset**.

Métodos	Escritura (Write)	Lectura (Read)	Solo lectura	Solo añade (Append)
CreateDynaset	Si	Ignorado	Si	Si
CreateSnapshot	Si	Ignorado	Si	Ignorado
Execute	Si	Ignorado	Ignorado	Ignorado
OpenTable	Si	Si	Si	Ignorado

La siguiente tabla muestra cual actualiza y pasa atravez de las condiciones de los mismos métodos.

Método	Actualización Consistente	Actualización Inconsistente	Por medio de SQL
CreateDynaset	Si (default)	Si	Si
CreateSnapshot	Ignorado	Ignorado	Si
Execute	Si	Si (default)	Si
OpenTable	Ignorado	Ignorado	Ignorado

Usando las constantes

Para hacer uso exclusivo de objetos **Recordset**, puedes ajustar los bits de las opciones **DB_DENYWRITE** y **DB_DENYREAD**. Usando ambas en combinación (como se muestra en el ejemplo anterior) prevendrás a otros usuarios de abrir las tablas que se estan utilizando. Puedes permitir a otros usuarios leer pero no modificar la tabla usando solo un bit "**DB_DENYWRITE**". Nota que el bit **DB_DENYREAD** es ignorado para todos los **Recordsets** excepto objetos **Table**.

El bit **DB_READONLY** indica que intentas leer datos **Recordsets** pero no cambiarlos. Usando este bit puedes mejorar el desempeño, ya que Visual Basic no soporta mucha carga de trabajo,

Los bits **DB_INCONSISTENT** y **DB_CONSISTENT** son provistos para determinar como unir tablas actualizadas. Esta opción se aplica solo cuando usas los métodos **CreateDynaset** o **Execute**. En el caso de **CreateDynaset** si escoges la opción **DB_INCONSISTENT** y ejecutas el método **Update**, los cambios se aplicarán a todos los campos de el **Dynaset**, aun si ellos también afectan a otros registros del **Dynaset**. Esto puede peligrar la integridad de tu base de datos y debe ser usado con cuidado. Cuando estas usando el método **Execute**, la opción **DB_INCONSISTENT** es por default.

Usando la opción SQLPassThrough

Los métodos **CreateDynaset**, **CreateSnapshot**, **CreateQueryDef** y **Execute** pueden usar la opción **DB_SQLPASSTHROUGH** si la pregunta contiene una sentencia SQL. Si esta opción es ajustada, Visual Basic no procesará la pregunta SQL localmente, pero pasará atravez de un servidor externo tal como Microsoft SQL u Oracle server via ODBC. Si tu no ajustas esta opción, Visual Basic interpretara la sentencia SQL e intentará ejecutar la operación localmente. Si la base de datos externa no entiende las preguntas SQL, el bit es ignorado y la pregunta es procesada por Visual Basic.

También puedes ajustar el bit **DB_SQLPASSTHROUGH** en la propiedad **Options** de un Data Control. Nota que no puedes usar **SQLPassThrough** sobre el método **OpenTable**, este no acepta una cadena SQL, y el método **Execute** no puede regresar datos a los registros.

Los resultados de una pregunta son depositados en un objeto **Snapshot** que puedes examinar pero no cambiar. Puedes tomar los resultados de el **Snapshot** y anexarlos a otros objetos **Recordset**.

Por ejemplo, para enviar la pregunta SQL del ejemplo anterior a un servidor ODBC, podrías usar el siguiente código:

```
Dim Db As Database, dsSomeData As Dynaset, SQL, Connect As String
Connect = "ODBC; DSN =Work; DBQ=Pubs; Uid=Vicky; PWD=Scholar"

Set Db = OpenDatabase ("", False, False, Connect)
SQL = "SELECT * FROM Publishers WHERE Publishers.City = ' Redmond'"
Set dsSomeData = Db.CreateDynaset (SQL, DB_SQLPASSTHROUGH)
```

La sintaxis de la sentencia SQL debe ser igual a la de la sintaxis de la maquina que ejecutará la sentencia SQL. Una vez que el servidor remoto ha procesado la pregunta, el servidor regresa todos los resultados a Visual Basic, y en algunos casos, regresa el número de registros afectados por la pregunta. Visual Basic convierte todos los resultados de una pregunta dentro de estructuras **Recordset**.
Para mayor información sobre el uso de estas opciones, buscar ayuda para **Options**.

Usando **SQLPassThrough** se hace un análisis cuando la implementación de Visual Basic no reúne las necesidades de situaciones específicas, y la base de datos externa soporta el dialecto SQL. En algunos casos, no todas las sentencias SQL pasan atravez de un servidor externo, los resultados pueden ser interpretados por Visual Basic.

Resumiendo, tu puedes usar el método **ExecuteSQL** para pasar acciones de preguntas a un servidor ODBC. Ambos métodos son discutidos en la sección "Usando métodos SQL", más adelante en este capitulo.

Aunque esto es posible para someter preguntas que contienen sentencias SQL que regresan múltiples resultados, cuando usas los métodos **CreateDynaset** o **CreateSnapshot** para someter estas preguntas, el resultado contiene solo el primer grupo de resultados de tu pregunta, cualquier resultado subsecuente será ignorado. Por ejemplo, si sometes una sentencia SQL que tuvo dos sentencias **SELECT**, solo resultados de la primera sentencia serán procesados por Visual Basic. En algunos casos, servidores externos usan procedimientos de almacenamiento que típicamente contienen varias sentencias SQL **SELECT**.

Usando propiedades Recordset

Después de que creaste un objeto **Recordset**, puedes usar las propiedades **Recordset** para tareas como controlar el puntero del registro grabado, exponer información acerca de como un **Recordset** fue construido, o manejo de **Bookmarks**. Las siguientes tablas resumen las propiedades **Recordset**, puedes ver muchas de estas propiedades aplicadas en ejemplos atravez de este capitulo. No todas las propiedades se aplican a todos los objetos **Recordset**. Para mayor información sobre tópicos individuales, busca ayuda para el tópico correspondiente, o consulta el lenguaje de referencia.

Las siguientes propiedades determinan la fuente de datos y atributos de el **Recordset**.

Propiedad	Descripción
Connect	La cadena **Connect** para tablas unidas. Para tablas base está siempre en blanco.
DateCreated	La fecha de cuando la tabla interna fue creada
LastUpdated	La fecha de la ultima modificación de la tabla
LockEdits	Afecta el comportamiento de visualizado de el método **Edit**. Valores **True** para **Pessimistic Locking**. **False** para **Optimistic Lockig**.
Name	El valor del argumento **Source** o **Name** usado para abrir o crear el **Recordset**.

Propiedad	Descripción
SourceTableName	El **SourceTableName** para tablas unidas. Para tablas base está siempre en blanco.
Transactions	Indica si un **Table** o un **Dynaset** soporta procesos de transacción. El valor es verdadero si soporta transacciones y falso si no soprta transacciones.
Updatable	Indica si el **Recordset** es actualizable o no. Para **Snapshot**, el valor de esta propiedad es siempre falso.

Para determinar como es el filtrado y almacenado el **Recordset**, se refieren las siguientes tres propiedades. Nota que estas propiedades son limpiadas una vez que el **Recordset** es creado.

Propiedad	Descripción
Filter	Una condición **Filter** puede ser aplicada a un **Recordset** creado,este **Recordset** (esencialmente la cláusula **WHERE** de una sentencia SQL) se aplica solamente a objetos **Snapshot** y **Dynaset**.
Index	El nombre de un índice. Cuando una tabla es abierta primero, el valor de esta propiedad esta en blanco. Aplicable solamente a objetos **Table**.
Sort	Es una condición de ordenación que puede ser aplicada a un **Recordset**, (esencialmente la cláusula ORDER BY de una sentencia SQL). Aplicable solo a objetos **Dynaset** y **Snapshot**.

Dos propiedades que permiten manejar el puntero del registro grabado.

Propiedad	Descripción
BOF	True si el puntero del registro grabado esta antes del primer registro en el **Recordset**. Si es True, el registro grabado es invalido.
EOF	True si el puntero del registro grabado esta después del ultimo registro en el **Recordset**. Si es True el registro grabado es invalido.

Tres propiedades son proporcionadas para manejar **Recordset Bookmarks**.

Propiedad	Descripción
Bookmark	El **Bookmark** de el registro grabado en el **Recordset**. Ajustar el valor de esta propiedad causará que el registro grabado se mueva hacia el registro con el valor **Bookmark** suministrado.
Bookmarkable	Indica ya sea que el **Recordset** soporte **Bookmarks** o no.
LastModified	El **Bookmark** de el último registro que estuvo en lugar dentro de la tabla o actualizado en la tabla.

Para determinar el resultado de el método **Seek** o uno de los métodos **Find**, puedes checar la propiedad **NoMatch**.

Propiedad	Descripción
NoMatch	True si el último método **Seek** o uno de los métodos **Find** sobre el **Recordset** fallaron.

Para determinar cuantos registros han sido procesados, checa la propiedad **RecordCount**.

Propiedad	Descripción
RecordCount	El número aproximado de registros procesados en un **Recordset** (ver nota de abajo).

Nota : Para bases de datos Visual Basic y Microsoft Access, el valor de la propiedad **RecorCount** es calculado como sigue: Cada vez que un registro es agregado a una tabla, el valor de esta propiedad es incrementado. Cada vez que un registro es borrado de una tabla, el valor de esta propiedad es decrementado. Si agregas o borras registros a una tabla dentro de una transacción, y entonces regresas la transacción, el valor de esta propiedad no esta ajustada para reflejar el hecho de agregar o borrar operaciones. El valor de esta propiedad no debe ser demasiado pesada.

Para **Dynaset** y **Snapshot**, la propiedad **RecorCount** es el número de registros que han sido introducidos por tu aplicación. En otras palabras, cuando una variable **Dynaset** o **Snapshot** es creada primero, el valor de esta propiedad es 1 (o 0 si el **Recordset** esta vacio); cada vez que el usuario ejecuta un método **MoveNext** sobre el **Dynaset** o **Snapshot**, el valor de esta propiedad es incrementado en uno. Ejecutar un metodo **MoveLast** sobre el **Dynaset** o **Snapshot** causa que el valor de esta propiedad sea ajustado a el número total de registros en el **Snapshot** o **Dynaset**. Usando cualquier método **Move** que no accesa a un registro, no tiene efectos sobre la propiedad **RecordCount**. En algunos casos, y especialmente en el caso de algunas preguntas ODBC, la primera vez que el **Recordset** es accesado, el **RecordCount** refleja el número total de registros en el **Recordset**.

Ordenando Recordsets

Para cambiar el orden de los registros en tu **Recordset**, puedes fácilmente controlar como debe estar presentado el dato en tu aplicación. Debes querer especificar el orden en el cual la información debe ser presentada al usuario. Por ejemplo, ejecutar cálculos en un **Recordset** ordenado nuevamente.

Hay diferentes opciones de ordenamiento que puedes usar para construir tu **Recordset**.

- Usa el orden fisico en la tabla.
- Usa uno de los índices de la tabla.
- Usa la propiedad **Sort** de un **Recordset**.
- Expresa una pregunta SQL para ordenar un grupo de registros en tu **Recordset**.

Cada una de estas técnicas es discutida en los siguientes párrafos.

Ordenando registros en una tabla

Inicialmente, los registros en una variable **Table** pueden aparecer en un orden impredecible. El orden fisico de registros en una tabla está determinado por el orden en el cual los registros fueron agregados a la tabla. Si no hay necesidad de presentar los registros en una secuencia dada, puedes simplificarlo usando el orden de la tabla.

Tu puedes acceder a los registros en una variable **Table** de acuerdo a el orden definido por cualquiera de los índices para aquella tabla, incluyendo la tabla primaria, si esta disponible.
Para ordenar los registros en una variable **Table** de acuerdo a un índice particular, ajusta la propiedad **Index** de la tabla, a el nombre de un índice definido para esa tabla. Por ejemplo:

```
Sub IndexTest ( )
        Dim Db As Database, T As Table
        Set Db = OpenDatabase ( "Biblio.mdb")
        Set T = Db.OpenTable ("Titles")
        Debug . Print T ("Company Name")
        T . Index = "City"
        Debug . Print T ("Company Name")
        T . Close
End Sub
```

Tu puedes ordenar los registros en una variable **Table** solo de acuerdo a los índices definidos por esa variable. Para ver que índices existen para tu tabla, usa la colección de índices. Si es necesario, puedes agregar índices adicionales a una tabla. Ajusta la propiedad **index** con una cadena, que no corresponda a un índice existente para que la tabla no cause un posible error. Para ordenar los registros en una tabla de alguna otra manera, crea una variable **Dynaset** o **Snapshot**, o crea un nuevo índice. Puedes entonces especificar un nuevo orden para el nuevo objeto.

Ordenando registros en un Dynaset o Snapshot

El orden de los registros en variables **Dynaset** o **Snapshot** esta determinado por la fuente de los registros :

- Si el **Dynaset** o **Snapshot** esta abierto sobre una tabla base, los registros son ordenados de acuerdos a la llave primaria para la tabla.

- Si la tabla base no tiene una llave primaria, y si la pregunta no especifica un orden, y la cláusula **WHERE** no especifica un indice usable, los registros son regresados en el orden en el cual fueron anexados a la tabla.

- Si el **Dynaset** o **Snapshot** esta abierto sobre una pregunta o en una sentencia SQL, el orden de los registros esta definido por las cláusulas **ORDER BY** (si es especificada).

- Si el **Dynaset** o **Snapshot** esta abierto sobre otro **Dynaset**, el orden de los registros esta determinado por la propiedad **Sort** del segundo **Dynaset**.

- Si el **Dynaset** o **Snapshot** esta abierto sobre un **Snapshot**, el orden esta determinado por la propiedad **Sort** de el **Snapshot** huesped.

- Si tu no ajustas la propiedad **Sort** de la fuente **Dynaset** o **Snapshot**, los registros en el **Dynaset** o **Snapshot** resultante, tienen el mismo orden que los registros en la fuente **Dynaset** o **Snapshot**.

Tu no puedes reordenar directamente un **Dynaset** o un **Snapshot** una vez creado. Sin embargo, puedes crear otra variable **Dynaset** o **Snapshot** que tenga un orden diferente. Debes hacer el ajuste de la propiedad **Sort** en el primer **Dynaset** o **Snapshot** y usar el método **CreateDynaset** o **CreateSnapshot** para crear un nuevo **Dynaset** o **Snapshot**. Por ejemplo:

```
Dim Db As Database, SomeData As Dynaset, SortedData As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set SomeData = Db . CreateDynaset ("Titles")
SomeData . Sort = "[Company Name]"
Set SortedData = SomeData . CreateDynaset ( )
```

La cadena que tu asignas a la propiedad **Sort**, debe ser una pregunta valida para ordenar típicamente o el nombre de un campo. La sintaxis es la misma a la usada para la cláusula **ORDER BY** de una sentencia SQL. Si quieres ordenar mas de un campo, lista los campos en el orden en el cual quieres que se ordenen, separando los nombres de los campos con columnas. Nota el uso de brakets [ ], cuando el nombre del campo tiene unidos espacios o columnas:

```
Dim Db As Database, SomeData As Dynaset, SortedData As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set SomeData = Db . CreateDynaset ("Products")
SomeData . Sort = "[ Category ID ], [ Product Name ]"
Set SortedData = SomeData . CreateDynaset ( )
```

Tu puedes usar expresiones como ajustes para la propiedad **Sort**. Puedes usar las mismas expresiones que utilizaste en la cláusulas **ORDER BY** de una sentencia SQL, usada cuando creaste una pregunta.

La ordenación por default es ascendente, pero tu puedes especificar una ordenación descendente para un campo con **DESC** después del nombre del campo.

```
Dim Db As Database, SomeData As Dynaset, SortedData As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set SomeData = Db . CreateDynaset ("Products")
SomeData . Sort = "[ Category ID ], [ Unit Price ] DESC [ Supplier ID ]"
Set SortedData = SomeData . CreateDynaset ( )
```

Para reordenar los registros en un **Dynaset** o **Snapshot** existente, sin crear un nuevo objeto, puedes asignar el resultado de el método **CreateDynaset** o **CreateSnapshot** a la misma variable.

```
Dim Db As Database, SnapData As Snapshot
Set Db = OpenDatabase ("Biblio.mdb")
Set SnapData = Db . CreateDynaset ("Sales for 1991")
SnapData . Sort = "[ Product Name ]"
Set SnapData = SnapData . CreateSnapshot ( )
```

Sin embargo, una vez que tu has reordenado los registros en esta forma, la única manera de remover el efecto de la propiedad **Sort** es relacer el original **Dynaset** o **Snapshot** (o almacenar el original en otra variable). Nota que cuando una variable **Snapshot** o **Dynaset** contiene el resultado de una pregunta, reordenando el registro en esta forma no reinicia la pregunta. Como con objetos **Dynaset**, no puedes reasignar un **Snapshot** así mismo mas de 10 veces sin cerrarlo y reconstruirlo.

Cuando nuevos registros son agregados a un **Dynaset**, estos son agregados al final de el **Dynaset**, sin hacer caso de como fue ordenado originalmente el **Dynaset**. Cambios que pudieran afectar la posición de ordenamiento del registro no tienen efecto sobre la posición del registro.

Selección de registros

Cuando tu quieres llegar mas allá del alcance de los registros regresados por un **Dynaset** o un **Snapshot** tu puedes:

• Crear un nuevo **Dynaset** o **Snapshot** después de ajustar la propiedad **Filter** sobre un **Dynaset** o **Snapshot** existente.

• Incluye una pregunta SQL que limite los registros, como los argumentos a un método **CreateDynaset** o **CreateSnapshot** ejecutado en contra de un **Dynaset** o **Snapshot** existente.

Analizando un **Recordset**, instruye a Visual Basic para que excluya los registros que no reúnen los criterios indicados por la propiedad **Filter** o la pregunta SQL.

Para filtrar un **Dynaset** o un **Snapshot** existente, puedes asignar una cadena conteniendo cualquier expresión valida de criterio a la propiedad **Filter** de una variable **Dynaset** o **Snapshot**.

La expresión de criterio puede ser tan simple como:

```
" [ NAME ] > 'M' "
```

Esta expresión regresa solo registros donde el campo "Name" contiene nombres mas grandes que "M". El criterio puede ser algo más complejo como :

```
"Publishers ! [Name] > 'M' AND Publishers ! [State] = 'NY'
```

Aquí la expresión acepta solo registros donde el nombre "Publishers" es mas grande que "M" y el estado Publishers es New York.

Nota : Expresiones en la propiedad **Filter** de un **Recordset** siguen la sintaxis de la cláusula **WHERE** de Microsoft Access SQL. Detalles acerca de Microsoft Access y Visual Basic sobre la implementación de SQL, son discutidos en el Apendice B "Microsoft Access SQL". Generalmente, cadenas de criterios validas usan el signo !, para delimitar las tablas y los campo en la expresión. Por ejemplo :

```
Table ! [ Field Name ]
```

El siguiente código filtra los resultados de la pregunta "Publishers" Para contener aquellos títulos cuyo nombre de compañía es "Microsoft Press":

```
Dim Db As Database, SnapData As Snapshot
Set Db = OpenDatabase ("Biblio.mdb")
Set SnapData = Db . CreateSnapshot("Publishers")
SnapData . Filter = "[Company Name] = 'Microsoft Press' "
Set SnapData = SnapData . CreateSnapshot ( )
```

Como puedes ver en el código, primero ajusta la propiedad **Filter**, entonces crea el **Dynaset** o **Snapshot** sobre si mismo. (Recuerde tener cuidado con respecto al número de veces que puedes reasignar una variable **Dynaset** a si misma.)
Una vez que has agregado un filtro a un **Dynaset** o **Snapshot** en esta forma, no puedes remover el filtro. El único camino para restaurar el **Dynaset** o **Snapshot** original es volverlo a crear, o para usar una variable **Recordset** que refiera a el **Recordset** sin filtrar.

En el ejemplo anterior, nota el uso de comillas simples en la cadena asignada a la propiedad **Filter**. Visual Basic acepta comillas simples o comillas dobles para delimitar cadenas dentro de SQL, incluyendo las cadenas asignadas a las propiedades **Filter** y **Sort** . Esto lo hace conveniente para incluir una cadena dentro de una cadena . Si la cadena ya incluye comillas simples, usa comillas dobles para delimitar la cadena dentro de la cadena

```
Q.Filter = " Authors.[ Name] = "" O' Keane, Bill """
```

Como con la propiedad **Sort**, tu puedes usar expresiones en la propiedad **Filter**. Puedes también ejecutar **Filter** y **Sort** juntos. Por ejemplo el siguiente código filtra la tabla "Titles" para producir un **Dynaset** que contiene solo Titulos que tienen cadenas SQL vinculadas en el nombre, y entonces clasifica los titulos por el dia que fueron publicados (date Published) y la primer letra de el nombre (Name) :

```
Dim Db Database, Somedata As Dynaset, SortData As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set Somedata = Db . CreateDynaset ("Titles")
SomeData . Filter = "[Name] LIKE ' *SQL* ' "
SomeData . Sort = "[Date Published], Left ([Name] , 1)"
Set SortedData =  SomeData . CreateDynaset ( )
```

Tu puedes hacer este ajuste sobre una variable **Table**. Si quieres restringir los registros en una Tabla, debes crear una variable **Dynaset** o **Snapshot** sobre la tabla.

Nota : Esto puede ser más eficiente para almacenar una pregunta o cadenas SQL, como un argumento para el **Recordset** ó **QueryDef** para producir un filtro **Dynaset** o **Snapshot**. Sin embargo, con un **Dynaset** o **Snapshot** existente, puedes usar la propiedad **Filter** para restringir registros sin especificar una sentencia SQL.

Cerrando variables Recordset

Para cerrar una variable **Recordset**, usa el método **Close**. Visual Basic automáticamente cierra las variables **Table**, **Dynaset** o **Snapshot** si destruyes la variable asociada. Las variables son destruidas cuando la forma o el procedimiento donde estan definidas son descargadas o terminadas. Cuando un **Recordset** es cerrado con una transacción pendiente, Visual Basic desarrollará todos los cambios pendientes a la base de datos y cerrará el **Recordset**. Usar el método **Close** es importante si usas variables Globales (nivel-modular) o variables **Recordset** Staticas (local). Porque Visual Basic no remueve esas variables, hasta que tu termines tu programa, reinicies tu código o cierres Visual Basic. Los **Recordsets** referidos con estas variables no son cerrados automáticamente.

Si usas el Data Control para abrir una base de datos, no tienes que cerrar explícitamente el **Recordset** o la base de datos, estos son cerrados automáticamente cuando descargas la forma en la cual se encuentra el Data Control. Cuando la forma es descargada, el evento **Validate** de el Data Control envía una acción del argumento **DATA_ACTIONUNLOAD**. Esta vez puedes ajustar la acción del argumento a cero para cancelar la operacion de descarga y mantener cualquier base de datos y variables **Recordset** que están aun abiertas.

Esto es una buena practica de programación, sin embargo, para cerrar explícitamente variables **Recordset** cuando estas terminando con ellas:

```
TableVar . Close
dsSomeData . Close
SnapData . Close
```

4.2 Posicionando el puntero del registro grabado en un Recordset

Aunque un **Recordset** puede tener de cero a cualquier número de registros, Visual Basic puede solo tener un registro **Recordset** disponible a la vez. Este registro es un puntero al registro; debes hacer un puntero al registro antes de poder acceder a los datos. Para hacer un puntero al registro especifico, necesitas usar el método **Seek**, o uno de los métodos **Move** o **Find** como los descritos en las siguientes secciones.

Cada registro en un **Recordset** tiene un registro previo y un registro posterior, con dos excepciones, el primer registro no tiene un registro previo, y el último registro no tiene registro posterior. Si el puntero al registro queda indefinido y tu aplicación trata de editar (**Edit**) o actualizar(**Update**) este registro, un posible error será generado y necesitarás reposicionar el puntero a un registro valido o cerrar el **Recordset**.

El puntero al registro quedará indefinido e inutilisable cuando es posicionado como sigue:

- Adelante de el primer registro, cuando la propiedad **BOF** es **True**.

- Detras del antepenúltimo registro, cuando la propiedad **EOF** es **True**.

- Encima de un registro que ha sido borrado, por no usar el método **Move** o **Bookmark**, después de borrar registros o pocisionarlos a un registro que otro usuario borró.

- Cuando usas el método **Seek** o el método **Find** y no pudieron ser encontrados registros que igualan el criterio o el valor llave. Si la propiedad **NoMatch** es verdadera (**True**) después de usar uno de estos métodos, puedes posicionar a un registro valido, pero no hay garantía de ello. Es mejor salvar el registro grabado valido en un **Bookmark** y restaurar el registro grabado para el valor **Bookmark**, después de uno de estos métodos.

- En un **Recordset** donde no hay registros. Esto se aplica cuando tu creas una variable **Recordset** sin registros o borras todos los registros en una tabla.

Nota : Si estas usando el Data Control y tiene especificado un válido **DatabaseName**, pero no un válido **RecordSource**, obtendrás la base de datos pero no el **Recordset**

El orden en el cual los datos son presentados a tu aplicación esta determinado por la forma en que tu construyas la variable **Recordset** y el orden de las tablas. Si no forzas cualquier otro orden, los registros son regresados en el orden determinado por la llave primaria, el orden en el cual los datos son anexados a la tabla, o en el orden determinado por tu cláusula **WHERE** SQL y la pregunta optima.

Cuando trabajas con el Data Control, recuerda que solo datos de registros grabados son movidos dentro de el control unido (bound control). Cambios hechos a los controles unidos son también automáticamente actualizados a los registros grabados y son salvados automáticamente.

Decidiendo sobre el mejor método de posicionamiento

Tu tienes muchas opciones cuando decides como y donde reposicionar el puntero del registro grabado. El que tu elijas depende de que tan cerca tu te quieras mover y con que clase de objetos estas trabajando. Tu querrás traer el método que toma el menor tiempo para ejecutarse y usar Visual Basic eficientemente.

Si estas saltando a el final de un **Recordset** para averiguar cuantos registros están incluidos, Visual Basic procesa cada registro por un largo camino. Para un gran número de registros, esto puede tomar un periodo largo de tiempo. Una vez posicionandonos en el final del **Recordset**, puedes checar la propiedad **RecordCount** para determinar aproximadamente el número de registros en el **Recordset**. Ambos métodos **FindLast** y **MoveLast** requieren de leer el número de **Recordset** antes de continuar procesando.

La siguiente tabla resume cuales métodos trabajan mejor para mover registros específicos.

Para mover	Usa
Para un registro advacente o próximo	Métodos **MoveNext** o **MovePrevious**
Para el principio o final de un **Recordset**	Métodos **MoveFirst** o **MoveLast**
Para especificar el registro en un objeto **Table** con llave para un índice.	Método **Seek**. Este es solo valido para objetos **Table**
Para un grupo especifico de registros que recibe un criterio dado en un **Dynaset** o **Snapshot**	Métodos de busqueda (solo validos para un **Dynaset** o un **Snapshot**)
Después de especificar un registro en un **Recordset**	Propiedad **Bookmark**
Para un registro o un grupo de registros que no pueden tener un correcto funcionamiento con el método usado	Una pregunta (query).

Generalmente, debes tratar de reposicionar un registro con pocas sentencias Visual Basic y métodos como sea posible. Cada llamada a la base de datos cuesta a tu aplicación tiempo y eficiencia. El método más rápido para actualizar un selecto juego de registros es usar una acción de una pregunta (query). Cuando te quieres posicionar a un registro distante antes de cambiar los datos, el camino más eficiente es usar el método **Seek** (sobre **Tables**), el método **Find** (sobre **Dynasets** ó **Snapshot**) y los métodos **Move** (sobre cualquier **Recordset**), usalos repetidamente en ese orden. Cada uno de estos métodos y técnicas son discutidos posteriormente.

Usando la acciones de las preguntas para hacer grandes cambios

En muchos casos es más eficiente modificar datos a través de una acción de una pregunta en lugar de usar repetidamente **Move**, **Change** muchas veces. La acción de una pregunta que lleva sentencias SQL que llevan a cabo algunas operaciones sobre tablas seleccionadas en la base de datos. Mas específicamente, las acciones de las preguntas son una categoria de preguntas que incluyen preguntas **Append**, **Delete** y **Update**. **Delete** y **Update** cambian datos existentes; **Append** agrega datos a tablas existentes.

Por ejemplo, si quieres buscar una tabla de una base de datos para un valor y repetirlo con otro, esto es más fácil usando una acción de una pregunta . La acción de una pregunta te permite tomar ventaja de la optimización de preguntas que emplea Visual Basic automáticamente. Tu puedes crear nuevas acciones de preguntas como tu las necesites usando variables **QueryDef**. Para más información, ver "Usando variables **QueryDef**" mas adelante en este capitulo.

Usando los métodos Move

Visual Basic proporciona cuatro métodos **Move** para cambiar el registro grabado.

Método	Proposito
MoveFirst	Nos posiciona en el primer registro grabado.
MoveLast	Nos posiciona en el segundo registro grabado.
MoveNext	Nos posiciona en el próximo registro grabado.
MovePrevious	Nos posiciona en el anterior registro grabado

Los métodos **MoveNext** y **MovePrevious** operan relativamente al registro grabado. Si hay un registro siguiendo el registro grabado, **MoveNext** hace esto en el registro grabado. Cuando un registro grabado esta posicionado pasado el último registro; **MoveNext** ajusta la propiedad **EOF** a verdadero y el registro grabado es indefinido. Una vez que la posicion del registro ha pasado el ultimo registro, usando **MoveNext** otra vez causa un posible error.

Cuando abres una nueva variable **Recordset**, la posición del registro esta ajustada al primer registro. Si no hay registros en el **Recordset**, ambas propiedades **BOF** y **EOF** son verdaderas, y el registro grabado esta indefinido.

Recuerda que usando un Data Control para abrir tu base de datos tendrá el mismo efecto, y que esos mismos métodos **Move** pueden ser usados sobre la propiedad **Recordset** del Data Control para manipular la base de datos.

El siguiente ejemplo usa el método **MoveNext** para pasar hacia adelante de los registros en un **Recordset**.

```
Sub MoveForward ( )
        Dim Db As Database, Ds As Dynaset
        Set Db = OpenDatabase ("Biblio.mdb")
        Set Ds = Db . CreateDynaset ("Titles")
        Do Until Ds . EOF
          Debug . Print Ds ("Year Published"), Ds ("Title")
          Ds . MoveNext
        Loop
        Ds . Close
        Db . Close
End Sub
```

Tu podrás retroceder registros usando el método **MovePrevious**.

```
Sub MoveBack ( )
        Dim Db As Database, Ds As Dynaset
        Set Db = OpenDatabase ("Biblio.mdb")
        Set Ds = Db . CreateDynaset ("Titles")
        Ds . MoveLast
        Do Until Ds . EOF
          Debug . Print Ds ("Year Published"), Ds("Title")
          Ds . MovePrevious
        Loop
        Ds . Close
        Db . Close
End Sub
```

Nota : Los dos ejemplos anteriores son solo para ilustrar. En otros lenguajes de programación es común que se pase a través de todos los registros en un **Recordset**, pero es raro cuando se usa Visual Basic. Cuando creas un reporte que debe contener valores que reflejen los valores de las tablas, podrías ejecutar una pregunta que haga los cálculos en lugar de intentar leer todos los registros de la tabla. El camino más eficiente para ejecutar una operación **Update**, **Delete** o **Append** sobre todos los registros en un **Recordset**, es usar una acción de una pregunta; el más eficiente camino para posicionarnos en un registro escogido, es usar el método **Seek** (para **Tables**) o los métodos **Find** (para **Dynasets** o **Snapshots**).

Usando los métodos Find

Con los registros en un **Dynaset** o un **Snapshot**, tu puedes hacer tu código mas eficiente, encontrando solo aquellos registros que igualan criterios. Para hacer esto, Visual Basic proporciona cuatro métodos **Find**, similares a los métodos **Move** discutidos anteriormente. En muchos casos, debes usar una pregunta para seleccionar un subgrupo general de registros para un **Dynaset** o un **Snapshot**, entonces usa los métodos **Find** para buscar dentro de aquel subgrupo. Cada método toma un criterio y busca atravez del **Dynaset** o el **Snapshot** por parejas. Los métodos **Find** no pueden ser usados sobre objetos **Table**.

El punto de partida para buscar, es mostrado en la tabla siguiente. Nota que el método **FindLast** lee todos los registros del **Recordset**, previo al registro buscado al inicio.
Los otros métodos tampoco inician hasta arriba (**FindFirst**) o en la posición del registro grabado.

Método	Que estas encontrando	Punto de partida
FindFirst criteria	Primer registro en criteria	Primer registro en el **Recordset**.
FindLast criteria	Ultimo registro en criteria	Ultimo registro en el **Recordset**.
FindNext	Próximo registro en criteria	Registro grabado.
FindPrevious	Previo registro en criteria	Registro grabado.

La sintaxis para la expresión criteria es la misma usada en la cláusula **WHERE** de una sentencia SQL, pero sin el "**WHERE**". Por ejemplo, podrías usar la siguiente cadena como el criterio para uno de los métodos **Find**.

```
" [ Last  Name ] > 'M' "
```

Los métodos **Find** no cambian el ajuste existente de los registros; ellos meramente localizan un registro en un **Recordset** que iguala los criterios.

El siguiente ejemplo encuentra solo libros cuyos títulos ("Title") contienen "SQL" y sitúa aquellos títulos dentro de un **List Box**:

```
Sub SQLBooks ( L As ListBox )
  Dim Db As Database, Ds As Dynaset, Criteria
  Set Db = OpenDatabase ("Biblio.mdb")
  Set Ds = Db . CreateDynaset ("Titles")
  Criteria = "[Title] LIKE '*SQL*'"
  Ds . FindFirst Criteria
  Do Until Ds . NoMatch
        L . AddItem Ds("Title") & " " & Ds ("Year Published")
        Ds . FindNext Criteria
  Loop
  Ds . Close
  Db . Close
End Sub
```

La propiedad **NoMatch** esta ajustada a verdadero cuando cualquiera de los métodos falla al encontrar un registro. Cuando esto ocurre, el registro grabado es indefinido. Si tu necesitas regresar a el registro que fue grabado antes de un método **Find** fallado, tu puedes usar un **Bookmark**. Para más información, ver la sección "Usando **Bookmarks**".

El siguiente ejemplo usa los métodos **Find** para eliminar valores **Null** en un campo dado de una tabla. Esto elimina la necesidad de una sentencia **Loop** a través de todos los registros de la tabla, uno por uno.

```
Sub EliminateNulls (FieldName, TableName, DatabaseName)
        Dim Db As Database, dsTable As Dynaset
        Dim Counter As Variant, Criteria As Variant
        If Fi  ldName = "" Or TableName = "" Or DatabaseName = "" Then
          E>  t  Sub
        End If
```

```
        Set Db = OpenDatabase(DatabaseName)
        Set dsTable = Db.CreateDynaset (Tablename)
        dsTable . MoveLast
        Counter = dsTable.RecordCount
        Criteria = FieldName & " Is Null"
        dsTable . FindFirst Criteria
        Do Until dsTable . NoMatch
          dsTable . Edit
         dsTable(FieldName) = Counter
          dsTable . Update
          Counter = Counter + 1
          dsTable . FindNext Criteria
        Loop
        dsTable . Close
        Db . Close
End Sub
```

Usando el método Seek

El método **Seek** proporciona un camino más rápido para encontrar registros en una variable de una tabla. El método **Seek** encuentra datos en un objeto **Table**, usando el índice correspondiente, como se definió en la propiedad **Index** de la variable **Table**. Si no hay un índice propio para usar, debes crear uno en orden para usar el método **Seek**.

El método **Seek** toma un número de variables de argumentos, el primero de los cuales esta siempre acompañado a la cadena que indica el tipo de comparación que será ejecutada.

Cadena comparativa	Primer registro encontrado
=	Igual al valor de la llave
>	Mas grande del valor de la llave
>=	Mas grande o igual al valor de la llave
<	Menor que la llave
<=	Menor o igual al valor de llave

El método **Seek** toma argumentos de la llave (índice primario). Por ejemplo, el siguiente código encuentra el primer Publisher con un nombre que empieza con la letra "U".

```
Dim Db As Database, Pubs As Table
Set Db = OpenDatabase("Biblio.mdb")
Set Pubs = Db.OpenTable("Titles")
Pubs.Index = "AU_ID"
Pubs.Seek ">=", "20"
If Pubs.NoMatch Then
   MsgBox "No encontre registro"
Else
   Debug.Print Pubs("Title") & " " & Pubs("AU_ID")
End If
Pubs.Close
Db.Close
```

Si el método **Seek** no encuentra un conjunto de registros, la propiedad **NoMatch** es ajustada a verdadero. Si la variable **Table** no tiene un índice ajustado, el método **Seek** causa un probable error en tiempo de ejecución.

Debes suministrar un valor de llave (índice primario) para cada campo, en el índice que quieres buscar. Cada valor llave debe tener el mismo tipo de datos que el campo. Por ejemplo, la llave primaria para la tabla "Titles" en la base de datos Biblio esta indexada sobre el campo ISBN. Tu podrías buscar un ISBN particular y regresar el nombre de el título en el campo "Title", usando este código:

```
Funtion FindTitle (ISBN As String) As String
  Dim Db As Database, Title As Table, CurrRecord As Variant
  Set Db = OpenDatabase("Biblio.mdb")
  Set Title = Db . OpenTable ("Titles")
  CurrRecord = Title.Bookmark
  Title . Index = "PrimaryKey"
  Title . Seek "=", ISBN
  If Title . NoMatch Then
        FindTitle = "< No hay ISBN >"
        Title . Bookmark = CurrRecord
  Else
        FindTitle = Title ("Title")
  End If
  Title . Close
  Db. Close
End Function
```

No tienes que suministrar valores para todos los campos en el índice; si tu omites un valor de un campo, el método **Seek** encuentra el primer registro que iguala los valores suministrados. Sin embargo, debes suministrar valores para al menos un campo en el índice. Nota que el método **Seek** no se puede mover a registros grabados anteriores a través de un índice, ni puede buscarlo en el último artículo indexado.

Usando Bookmarks

La propiedad **Bookmark** de un objeto **Recordset** habilita una marca en el **Recordset**, así tu puedes regresar después. Esto es similar al número del registro grabado, usado en otras bases de datos. Por ejemplo, puedes mantener un puntero en el registro grabado para salvar un **Bookmark** antes de usar **Seek** o uno de los métodos **Find** para moverte a otro registro. Si el moverte no resulta exitoso y abandona el registro grabado como indefinido, el **Bookmark** habilitará el retorno a aquellos registros salvados. También puedes determinar las más resientes adiciones ó modificaciones a un registro con la propiedad **LastModified**, la cual retorna un valor **Bookmark** a aquel registro.

Algunas tablas unidas o bases de datos que no tienen formato Microsoft Access no pueden soportar **Bookmarks**. Por ejemplo, cuando tablas Paradox no tienes una llave primaria, no puedes usar **Bookmarks** con un **Dynaset** o una pregunta basada en aquellas tablas. Tu puedes determinar si un **Dynaset** soporta **Bookmarks** por inspección, esta propiedad es **Bookmarkable**:

```
Function GetBookmark (ds As Dynaset)
  If ds.Bookmarkable Then
        GetBookmark = ds.Bookmark
  Else
        GetBookmark = Null
  End If
End Function
```

Un **Snapshot** siempre soporta **Bookmarks**, sin hacer caso de las tablas internas.

Los **Recordsets** que tu creas usando Visual Basic o de tablas Microsoft Access, tienen una propiedad **Bookmark** que tu puedes usar para marcar el lugar de los registros. La propiedad **Bookmark** proporciona un único valor binario que tu puedes almacenar en una variable **String** o **Variant**:

```
Dim Db As Database, Ds As Dynaset, CurrentRec
Set Db = OpenDatabase ("Biblio.mdb")
Set Ds = Db . CreateDynaset (TableName)
. . .
CurrentRec = Ds.Bookmark
```

Cuando tu quieres regresar a un registro marcado con un **Bookmark**, ajusta la propiedad **Bookmark** igual a el valor de el **Bookmark** que quieres encontrar.

```
Ds. Bookmark = CurrentRec
```

Usando la propiedad **Bookmark**, tu puedes escribir código para regresar a el registro grabado, si un método **Find** no sucede.

```
CurrentRec = Ds.Bookmark
Ds.FindFirst criteria
If Ds.NoMatch Then Ds.Boookmark = CurrentRec
```

También puedes usar el Data Control para crear **Bookmarks** para un **Dynaset**. Esto habilita la marca a un registro mostrado por los controles unidos a la forma o para cambiar el registro mostrado para un ajuste del **Recordset** del Data Control. Por ejemplo, tu podrías querer agregar un botón a la forma para acceder a la base de datos Biblio, para permitir a un usuario mostrar el Publisher de un libro. Tu podrías hacer esto llamando a la siguiente función.

```
Function ShowPublisher ( )
  Dim Ds As Dynaset , MyBookmark
  Set Ds = data1.Recordset
  MyBookmark = Ds.Bookmark
  Ds . FindFirst "[Publisher] = Ds.PubId"
  If Ds . NoMatch Then
        Ds.Bookmark = MyBookmark
  End If
End Function
```

Cuando cierras una variable **Recordset**, todos los **Bookmarks** quedan invalidados. Tu no puedes usar un **Bookmark** obtenido de un **Recordset** en otro **Recordset**, aun si ambos están abiertos sobre la misma tabla o pregunta. Haciendo esto causará un probable error en tiempo de ejecución. Tu puedes compartir **Bookmarks** entre copias de **Recordsets**, como se discutió en la sección **"Copias de variables Recordset"**.

IV.3 Usando Recordsets para manipular datos

Una vez que definiste la variable **Recordset**, tu puedes acceder a los datos. La siguiente sección discute como usar la propiedad **Value** de un campo para examinar los valores de los datos.

Usando sintaxis Visual Basic para direccionar objetos

Visual Basic proporciona una sintaxis flexible que tu puedes usar para referir a los objetos de acceso a datos. Tu encontrarás que hay muchos caminos para referir un objeto de acceso a datos. Esta sección explica como modificar sentencias Visual Basic que direccionan correctamente el objeto al dato que tu quieres referir.

Para empezar, Visual Basic te permite poner los nombres de los objetos por default de expresiones que se refieren a un objeto especifico. Por ejemplo, para referir a la propiedad **Text** de el Text1 Control de la Form1, podrias usar cualquiera de las siguientes sentencias:

```
A$ = Form1 ! Text1.Text
ó
A$ = Form1.Text1.Text
ó, si la forma por default es Form1:
A$ = Text1. Text
```

ó desde la propiedad de un Text Control como es la propiedad **Text**, podrías escribir aun menos código:

```
A$ = Form1. Text1
ó
A$ = Text1
```

Tu puedes usar esta misma sintaxis trabajando con un objeto que refiere a un dato. Para hacer esto, necesitas estar consiente de la colección por default y de las propiedades de cada uno de los objetos que serán definidos.

Objeto	Colección por default	Propiedades por default
Database	**TableDefs**	**Connect**
Field	(ninguna)	**Value**
Index	(ninguna)	**Name**
QueryDef	(ninguna)	**Name**
Table, Dynaset, Snapshot	**Fields**	**Name**
TableDefs	**Fields**	**Name**

Cuando te refieres a una colección, debes siempre indicar el miembro de la colección que quieres direccionar por el número ordinario o por el nombre. Por ejemplo, podrias usar cualquiera de la sentencias siguientes para referir a el primer miembro **TableDef** ("Titles") en la colección **TableDefs** de una variable **Database** llamada "Books":

```
Books. TableDefs ( 0 )
ó
Books. TableDefs ("Titles")
ó
Books ! Titles
```

Si el nombre del objeto tiene espacios, debes encerrar el nombre con brackets [Last Name] cuando uses la sintaxis !.

Puedes simplificar el direccionamiento de objetos, creando variables de acceso a datos para referir a objetos especificos. Una vez creadas y ajustadas estas variables, puedes substituirlas por los objetos referidos, por ejemplo.

```
Dim Db As Database
Dim Ds As Dynaset
Dim F As Field
Dim i%
Set Db = Opendatabase("Biblio.mdb")
Set Ds = Db.CreateDynaset ("Titles")
Set F = Ds.Field (0)
```

Todas las siguientes líneas hacen lo mismo : Muestran los datos del campo 0 de la tabla "Titles" de la base de datos Biblio.

```
Print Ds.Fields ( 0 ) .Value
Print Ds.Fields ( 0 )
Print Ds ( 0 )
Print F.Value
Print F
Print Ds.Fields ("Title") .Value
Print Ds ! Title
Print Ds.Fields ("Title")
Print Ds ("Title")
```

Direccionando la propiedad Value de los campos

La propiedad **Value** de un objeto **Field**, determina los datos almacenados en un campo de la base de datos. Solo los datos del registro grabado del **Recordset** están disponibles. La propiedad **Value** es válida solo cuando el objeto **Field** es parte de una variable **Recordset** y el registro grabado es valido. Cuando tu accedes a variables **Recordset**, la colección **Fields** es la colección por default y la propiedad **Value** es la propiedad por default. Esto habilita la dirección de la propiedad **Value** en una variedad de formas. Por ejemplo; cada Text Box en el siguiente código, obtiene datos de la propiedad **Value** de un campo en el **Dynaset**:

```
Dim Ds As Dynaset
Dim Db As Database
Set Db = OpenDatabase("Biblio.mdb")
Set Ds = Db.CreateDynaset("Titles")
Text1.Text = Ds ("Title")                    ' Se refiere al campo por el nombre.
Text2.Text = Ds ( 0 )                        ' Se refiere al campo por el número ordinario.
Text3.Text = Ds ! [Year Published]   ' Se refiere al campo por el nombre.
' Los Brackets ( [ ] ) son necesarios por los espacios en blancos.
Text4.Text = Ds ! AU_ID                      ' Se refiere al campo por el nombre.
Text5.Text = Ds.Fields ( 3 )
' Se refiere a el cuarto miembro de la colección fields.
Text6.Text = Ds.Fields ("PubId")
' Se refiere a el miembro de la colección Fields por el nombre.
```

En muchos casos, encontrarás que usando el número ordinario para acceder a los campos es el más eficiente. Otro efectivo camino para obtener datos en un **Dynaset**, sin embargo, es para referir directamente a los campos por nombre, se muestra en la siguiente sintaxis:

```
Dim CoNar  eS, FieldNameS, i%
Dim Db As Database, Ds As Dynaset
```

```
Set Db = OpenDatabase ("Biblio.mdb")
Set Ds = Db . CreateDynaset ("Publishers")
Print Ds ("Company Name"), Ds ("PubId")
```

Tu puedes acceder a los campos en el registro grabado de una variable **Table**, **Dynaset**, o **Snapshot** usando el operador ! y el nombre del campo. Cuando usas el operador ! agrega brackets [ ] a la estructura del campo que contiene espacios o de otro modo Visual Basic invalida los identificadores:

```
Text3. Text = Ds ! [Year Published]
```

También puedes usar una cadena conteniendo el nombre de un campo (sin hacer caso de espacios), para acceder a los contenidos de aquel campo en el registro grabado. Por ejemplo, las siguientes sintaxis son equivalentes:

```
Print Ds ! [Company Name ], Ds ! PubId
Print Ds ("Company Name"), Ds ("PubId")


FieldName$ = "Company Name"
Print Ds ( FieldName$ ), Ds ! PubId.Value.
```

Nota que puedes leer y mostrar los valores de esos campos, pero no puedes asignar nuevos valores a ellos, sin usar los métodos **Edit** y **Update**.

Nota : En terminología de Bases de datos, un valor **Null**, significa que el dato no es reconocido, no que el dato es una cadena vacia o nula. Por ejemplo, un valor **Null** debe ser asignado a el campo conteniendo el dato.

Si tu estas trabajando un campo en un **Recordset** que puede contener valores **Null**, debes usar una variable **Variant** para almacenar los contenidos del campo. Variables **Variant** Visual Basic pueden contener valores **Null**, mientras que otros tipos de datos no. Un error ocurre si tu asignas un **Null** a cualquier otro tipo de variable, fuera del alcance de un campo.

Usando la propiedad Ordinal

Otro camino para acceder al dato de el **Recordset**, es usar la propiedad **Ordinal** de el objeto **Field**. La propiedad **Ordinal** determina el orden conocido de los objetos **Field** en el **Recordset**. Esto esta basado sobre el orden en el cual los campos aparecen en el **Recordset**. (La propiedad **Ordinal** para el primer campo es 0). Por ejemplo, tu podrias usar el siguiente código para direccionar la propiedad **Value** del tercer campo de una variable **Dynaset** Ds:

```
CoName = Ds (2)
```

Usando la propiedad **Ordinal**, tu puedes también usar un contador para indexar a través de los campos:

```
For i% = 0 To Ds.Fields.Count - 1
        Print Ds (i%)
Next  i%
```

Accediendo campos usando el Data Control

Cuando tu estas usando el Data Control con una base de datos abierta, la propiedad **Recordset** de el Data Control contiene un **Dynaset**. Para acceder a este **Dynaset**, puedes usar las siguientes técnicas. Si tu estas usando la edición estandar, tu no puedes usar la sentencia **Dim** para crear una variable **Dynaset** para mantener el **Recordset** (debes usar el nombre completo).

```
Data1 . DatabaseName = "Biblio.mdb"
Data1 . RecordSource = "Titles"
Data1 . Refresh
Print Data1 . Recordset . ("Title")
```

Si tu estas usando la edición profesional, tu puedes usar la variable **Dynaset** :

```
Dim Ds As Dynaset, TitleName$, PubYear%, Author_ID%, BookNumber$
Set Ds = Data1.Recordset
TitleName$ = Ds( 0 )
PubYear% = Ds("Year Published")
Author_ID = Ds( 2 )
BookNumber$ = Ds("ISBN")
```

Nota que Visual Basic hace todas las conversiones necesarias cuando mueves valores **Recordset** dentro de otras variables. Por ejemplo, esto no es necesario para forzar una conversión cuando la variable destino es un dato tipo numérico.

Cambiando registros

Una vez que tu has encontrado un registro, tu puedes cambiar los contenidos. Cambiar el valor de un registro con código es un proceso de tres pasos.

1.- Usar el método **Edit** para preparar el registro grabado para edición.
2.- Asigna nuevos valores a los campos que tu quieres cambiar en el registro grabado.
3.- Usa el método **Update** para salvar los cambios a el registro grabado.

El siguiente ejemplo ilustra como es hecho esto:

```
Sub ReplaceCity (OldValue, NewValue)
Dim Db As Database, T As Table
        Set Db = OpenDatabase ("Biblio.mdb")
        Set T = Db . OpenTable ("Publishers")
        Do Until T.EOF
          If T("City") = OldValue Then
                T. Edit
                T("City") = NewValue
                T.Update
          End If
          T . MoveNext
        Loop
        T. Close
End Sub
```

Un probable error ocurre si tu tratas de cambiar los valores en el registro grabado sin primero usar el método **Edit**. Después de que cambies los valores en algunos campos, debes usar el método **Update**. Si tu te mueves a un registro diferente sin primero usar el método **Update**, los cambios son perdidos. Por ejemplo, omitiendo el método **Update** del código previo, podría resultar el no tener cambios en la tabla.

Tu puedes completar esta tarea usando una pregunta con una sentencia **Update**. Para mayor información sobre preguntas, ver "Usando variables **QueryDef**".

Manejando errores con el Método Edit

En situaciones multiusuarios, muchos usuarios pueden tratar simultáneamente de agregar, cambiar o borrar los mismos registros. Debes estar preparado para probables errores que resultan cuando el **Dynaset** sobre el que estas trabajando, debido a los cambios, deja de existir en la base de datos.

El método **Edit** responde a errores dependiendo de cual tipo de mecanismo estes usando para cerrar:

• Con **Pessimistic Locking**, Visual Basic automáticamente cierra la página conteniendo el registro grabado, cuando tu ejecutas el método **Edit**. Otros usuarios están fuera de la página hasta que ejecutes el método **Update**.

• Con **Optimistic Locking**, Visual Basic retrasa el cerrando de la página hasta que el **Update** es ejecutado. Esto da la oportunidad a otros usuarios para editar y actualizar el registro sobre el cual estas trabajando.

Errores con Pessimistic Locking

Cuando tu estas usando el método **Pessimistic Locking**, el método **Edit** genera un posible error si el registro grabado ha sido cambiado o borrado en la tabla base desde que tu creaste tu **Dynaset**. Por ejemplo, supón que creas un **Dynaset** sobre un grupo de registros, y unos pocos segundos después, otro usuario crea un **Dynaset** que incluye algunos de aquellos mismos registros. Si tu o cualquier otro usuario subsecuentemente borra o cambia cualquiera de los registros compartidos, se genera un posible error. Tu tienes errores cuando estas tratando de editar un registro que no contiene valores o han sido borrados.

Si tu te encuentras en esta situación, puedes intentar el método **Edit** otra vez, después Visual Basic carga los valores de los registros grabados dentro del **Dynaset**. Si un registro fue borrado, tu deberás agregarlo con los nuevos valores.

Errores con Optimistic Locking

Cuando tu estas usando el método **Optimistic Locking**, Visual Basic aplaza el cerrado de la página hasta que el método **Updated** es ejecutado y llegará a ser necesario para verificar que los datos no han cambiado desde que el método **Edit** fue usado por última vez. Esto significa que otros usuarios pueden editar o actualizar el registro sobre el cual estas trabajando. Si otros usuarios cambian el registro que estas editando, el método **Update** generará un posible error.

Actualizando tablas múltiples con Dynasets

Tu puedes crear variables **Dynaset** que refieran a más de una tabla base ó una combinación de tablas base y otros **Recordsets**. Para construir un **Recordset** que refiera a más de una tabla sencilla, Visual Basic ejecuta una "unión relacional".

Esta operación conecta un registro de una tabla a uno o muchos registros en otra tabla, atravez de un campo común, usualmente un campo llave, o muchos registros en una (o más) tabla (s). En Visual Basic, hay tres tipos de uniones:

- Uno a uno; une donde hay un (y solo un) registro en una tabla que esta unida a un registro en otra tabla.

- Uno a Varios; une donde hay un registro en una tabla que esta unida a más de un registro en otra tabla.

- Varios a Varios; une donde hay un producto cruz creado, usando múltiples filas de tablas múltiples. Generalmente, esto no es útil.

Cuando tu haces cambios a un **Dynaset** que refiere a tablas múltiples, los cambios son reflejados en todas las tablas relacionadas. Normalmente hay una restricción. En una a muchas y de muchas a muchas uniones, solo el dato de el lado "muchos" de la unión puede ser cambiado. Intentar cambiar los campos sobre el lado contrario de la unión causa un error de corrida.

Tu puedes escoger ignorar esta restricción creando un **Dynaset** inconsistente, esto te permite modificar libremente los valores sobre el lado "uno" de uno a muchas uniones. Crea un **Dynaset** inconsistente para suministrar la bandera inconsistente en el segundo argumento del método **CreateDynaset**:

```
Dim Db As Database, dsProduct As Dynaset
Set Db = OpenDatabase("Biblio.mdb")
Set dsProduct = Db . CreateDynaset ("New Titles", DB_INCONSISTENT)
```

Aun cuando tu creas un **Dynaset** inconsistente, algunos de estos campos pueden no ser actualizables. Por ejemplo, el contador de los campos no pueden ser actualizado. También, un **Dynaset** creado sobre alguna clase de tablas unidas puede no ser actualizable, dependiendo de la fuente de la tabla unida.
Si cualquiera de los campos en un **Dynaset** no son actualizables, la propiedad **Updatable** de el campo es falsa. Tu puedes examinar la propiedad **Attributes** de cada campo en el objeto **Dynaset** de la colección de campos para determinar cuales campos en el **Recordset** son actualizables.

Nota : Cuando tu estas trabajando con datos localizados en múltiples tablas, necesitas mantener la referencia entre todas las tablas. Integridad referencial, se refiere a las reglas que sigues para preservar las relaciones definidas entre las tablas cuando tu ingresas o actualizas registros.

Para bases de datos creadas con Visual Basic, este código será responsable de mantener la integridad referencial en la base de datos. Cuando tu actualizas un **Dynaset** inconsistente, tu puedes fácilmente destruir la integridad relacional de los datos en el **Dynaset**.
Debes tener cuidado para entender como el dato es relacionado atravez de una a muchas uniones y actualizar los valores sobre ambos lados de la unión en una forma que preserve la integridad referencial.

Para bases de datos creadas con Microsoft Access que tiene establecidas relaciones, la base de datos habrá que cumplir las reglas de integridad referencial por ti. Para mayor información a cerca de como establecer y manejar una base de datos relacional, consulta las referencias bibliográficas o ve la documentación de Microsoft Access.

Borrando registros

Para borrar un registro entero, usa el método **Delete**. Por ejemplo, podrías usar el siguiente código para borrar todos los registros en una tabla:

```
Sub EmptyTable ( tAny As Table )
        tAny . MoveFirst
        Do Until tAny . EOF
          tAny . Delete
          tAny . MoveNext
        Loop
End Sub
```

Cuando tu usas el método **Delete** contra un **Dynaset** o **Table**, **Delete** remueve el registro de la base de datos y hace el registro grabado invalido. El registro no es removido del **Dynaset** hasta que tu te muevas a un nuevo registro. Tu te puedes mover a un nuevo registro usando los métodos **MoveFirst**, **MoveNext**, **MoveLast**, **MovePrevious**, o **Seek**.

Si tu intentas obtener o ajustar los valores de cualquier campo en el registro grabado, después que el registro ha sido borrado, Visual Basic genera un probable error de corrida. Para evitar este error, simplemente muévete a otro registro después de usar **Delete**.

Cuando un registro es borrado de un **Recordset** y el registro esta siendo compartido por uno o más **Recordsets** abiertos, los usuarios de otros **Recordsets** no serán notificados a menos que ellos traten de acceder a aquel registro específico. Cuando los métodos **Edit** o **Delete** son usados sobre un registro que ha sido marcado para borrar, un probable error ocurrirá avisando que el registro ya no esta en la base de datos.

Otra forma rápida para borrar registros es usar el método **Execute**. Para mayor información sobre **Execute**, ver las secciones "Usando Métodos SQL" o "Manipulando **Recordsets** con variables **QueryDef**".

Agregando nuevos registros

Tu puedes agregar o insertar nuevos registros en una **Table** o **Dynaset**. Agregar un nuevo registro es un proceso de tres pasos:

1.- Usa el método **AddNew** para crear un nuevo registro. El puntero del registro grabado esta salvado.
2.- Asigna nuevos valores a los campos en el nuevo registro.
3.- Usa el método **Update** para salvar el nuevo registro. Cuando la actualización esta completada, el puntero del registro grabado es restaurado para salvar el valor.

El siguiente ejemplo ilustra como esto es hecho:

```
Dim Db As Database, T As Table
Set Db = OpenDatabase("Biblio.mdb")
Set T = Db.OpenTable("Publishers")
T.AddNew
T("Name") = "Nuevo libro"
T("PubID") = 31
T.Update
T.Close
Db.Close
```

Cuando uses el método **AddNew**, Visual Basic salva el puntero del registro grabado y crea uno nuevo. Tu puedes entonces hacer cualquier cambio que tu quieras para salvar propiedades de los campos, para ajustar los valores de los nuevos registros. Cuando todos los campos requeridos han sido llenados, usa el método **Update** para agregar el nuevo registro a el final del **Recordset**. El puntero del registro grabado es entonces automáticamente reseteado al salvar el valor. Así, cuando la secuencia **Edit/Update** esta completa, el puntero del registro grabado que fue salvado será restaurado.

Si tu estas agregando nuevos registros a la variable **Table**, el nuevo registro es posicionado de acuerdo al índice. Si no hay índice, el nuevo registro es posicionado al final de la tabla. Para forzar un nuevo registro a aparecer apropiadamente ordenado en el **Dynaset**, debes reordenar y reconstruir el **Dynaset**, o cerrar el **Dynaset** y correr la pregunta que lo produjo.

Si tu usas **AddNew** y te mueves a otro registro sin primero usar **Update**, un nuevo registro no es agregado y todos los valores del nuevo campo son perdidos sin aviso. Por ejemplo, refiriendose al ejemplo del código anterior, dejando fuera la sentencia **T.Update**.

Tu programa generará un probable error si tratas de agregar un registro a un **Recordset**, cuando el registro impone una de las siguientes condiciones:

- Violar la única llave primaria.
- Tiene un valor Nulo en la llave primaria.
- Viola la integridad relacional establecida por la base de datos.

Visualizando los datos

Cuando múltiples usuarios están intentando cambiar el mismo dato, hay un conflicto obvio. La solución a este conflicto común a todos los usuarios de la base de datos, es cerrarla. Para cerrar un dato que esta siendo cambiado por otro usuario, la base de datos debe estar segura de que otros usuarios simultáneamente no cambien el mismo dato.

Debes correr múltiples ejemplos o casos de una aplicación donde cada caso accese la misma base de datos. En esta situación, cada caso es en efecto un usuario separado de la base de datos.

Además, aun en un caso sencillo de una aplicación, múltiples procesos pueden acceder el mismo dato. Por ejemplo, si tu código abre múltiples **Tables** y variables **Dynasets** sobre el mismo dato, el procedimiento que usas con estas variables puede originar conflictos sobre la tabla. Aun cuando tu abras un **Recordset** en modo exclusivo, otros procedimientos en tu propia aplicación podrán ser abiertos.

Visual Basic automáticamente cuida del cerrado en los procedimientos. Esto también ejecuta el checado de los datos automáticamente cuando tu código esta actualizando datos con variables **Table** y **Dynaset**.

Nota : Cuando accesas datos externos (tales como Microsoft SQL server o Oracle ODBC), la metodología de visualizado de los datos es una responsabilidad de la base de datos remota. Visual Basic actua como una entrada a estos servidores y no tiene el control de los datos internos. En algunos casos, tu puedes controlar como el servidor remoto cierre los datos, a través de usar sentencias especificas SQL u opciones administrativas.

Visual Basic proporciona tres niveles de visualizado de datos :

- Visualizando Base de datos.
- Visualizando **Tables** y **Dynasets**.
- Visualizando páginas.

Visualizando bases de datos

Una de las formas más simples, pero también más restrictivas para hacer que solo tu código pueda cambiar datos. Tu puedes hacer esto ajustando la parte **Exclusive** de la función **OpenDatabase** a verdadero:

Set Db = **OpenDatabase** ("Biblio.mdb",True)

Esta función causa un error de corrida si otro usuario u otros usuarios de Visual Basic ya tiene abierta la base de datos (a menos que haya sido abierta como solo lectura).

Más de una persona puede abrir una base de datos cuando la parte **ReadOnly** de la función **OpenDatabase** es verdadera, o cuando las partes **Exclusive** y **ReadOnly** son verdaderas. Si alguien abre una base de datos con una o ambas opciones ajustadas a verdadero, otros intentos para abrirla fallarán.

Nota que cuando cierras una base de datos usando código, tu código (del mismo programa) puede aun abrir variables adicionales **Database** sobre esta misma. Porque cerrando una base de datos entera es también restrictivo, debes usarla solo en situaciones especiales, tales como actualizaciones masivas o reestructuración de una base de datos.

Usando el Data Control para ganar acceso exclusivo a la base de datos es también fácil:
Solo ajusta la propiedad **Exclusive** a verdadero antes de **Refresh** :

Data1 . **Exclusive** = True
Data1 . **DatabaseName** = "Biblio.mdb"
Data1 . **RecordSource** = "Titles"
Data1 . **Refresh**

Visualizando Tables y Dynasets

Un camino menos restrictivo y simple para acceder a los datos, es visualizando un **Table** o **Dynaset**. Cuando tu abres una variable **Table** o **Dynaset**, puedes ajustar partes de la función abierta exclusivamente:

Option% = **DB_DENYWRITE** + **DB_DENYREAD**
Set DataRecs = Db . **CreateDynaset** ("Titles", Option%)

En este caso, especificando **DB_DENYWRITE**, tu prohibes a otros usuarios escribir en los registros de tu **Dynaset**, y especificando **DB_DENYREAD**, también previenes que otros usuarios los puedan leer.

De la misma manera, visualizar el **Dynaset** del Data Control, puede también ser realizado usando la propiedad **Options** y el mismo **DB_DENYWRITE** y **DB_DENYREAD**:

Data1 . **Options** = **DB_DENYWRITE** + **DB_DENYREAD**
Data1 . **DatabaseName** = "Biblio.mdb"
Data1 . **RecordSource** = "Titles"
Data1 . **Refresh**

Cuando tu abres una variable **Table** o **Dynaset**, asegúrate que solo tu código puede cambiar, borrar o unir cualquier registro dentro del **Recordset**. Otros usuarios pueden leer los contenidos de los registros en las tablas de tu **Recordset** exclusivo, pero no pueden escribir.

Visualizando páginas

Cada base de datos que abres con Visual Basic puede usar su propio esquema de visualizado. Si tu no visualizas la base de datos entera, **Table** o **Dynaset**, Visual Basic automáticamente visualiza los datos por páginas. Cuando accesas a bases de datos Microsoft Access/Visual Basic, las páginas son visualizadas en registros de 2048 bytes (2k) de tamaño. Visual Basic almacena muchos registros ajustados sobre cada página. Cuando Visual Basic visuliza la página que contiene un registro que estas editando.

Método Pessimistic Locking

Por default Visual Basic usa **Pesimistic**, cuando tu agregas o actualizas registros en cualquier variable **Dynaset** o **Table**. Con esta estrategia Visual Basic visualiza la página conteniendo un registro tan pronto como tu uses el método **Edit** y no visualiza la página hasta que uses **Update**. Cancela la edición usando el método **Rollback** para deshacer la edición o mover el registro.

La ventaja de **Pessimistic Locking** es que tu conoces un método **Edit**, esto te garantiza que tu programa lea todos los datos. La desventaja es que la página que contiene el registro esta visualizada a la vez que ejecuta el código entre **Edit** y el **Update**. La página podria estar visualizada para un inaceptable periodo de tiempo. Durante este periodo de tiempo, otros usuarios podrian ver los datos anteriores contenidos en aquellas páginas, pero serian incapaz de editarlos.

El siguiente código usa los ajustes por default (**Pessimist Locking**) para actualizar un registro:

```
Dim Db As Database, DataRecs As Dynaset
Set Db = OpenDatabase("Biblio.mdb")
Set DataRecs = Db.CreateDynaset ("Titles")
DataRecs.Edit
. . .
DataRecs.Update
```

Método Optimistic Locking

Si tu quieres abandonar el registro sin cerrar hasta que uses el método **Update**, ajusta la propiedad del registro **LockEdits** a falso. Este causará que Visual Basic use la opción **Optimistic Locking**.

Con **Optimistic Locking**, Visual Basic no cierra la página que contiene el registro cuando uses el método **Edit**. La página es cerrada solo cuando tu realmente actualizas el registro con el método **Update**. La ventaja de esta estrategia es que las páginas están cerradas brevemente durante el tiempo en que los registros son realmente actualizados. La desventaja es que tu no puedes estar seguro cuando inicia la edición de un registro que tu actualizaste realmente.

La estrategia **Optimistic Locking** varia considerablemente, dependiendo en como planeas usar la base de datos. Si tu esperas actualizaciones conflictivas, entonces posiblemente **Pesimistic Locking** es la mejor opción. Si tu esperas atualizaciones y borrados raramente conflictivos, entonces **Optimistic Locking** es la adecuada.

IV.4 Usando variables QueryDef

Una variable **QueryDef** es un objeto que contiene una sentencia SQL que describe una pregunta. Tu puedes usar una **QueryDef** para salvar una pregunta predefinida en la base de datos.
QueryDef no almacena datos, almacena la definición de una pregunta usada para recobrar datos.

Tu declaras una **QueryDef** de la misma forma que declaras otras variables objeto:

Dim Q As QueryDef.

Para ajustar las propiedades de una variable **QueryDef**, puedes redefinir una pregunta existente o definir completamente una nueva pregunta.

Crear una **QueryDef** es usualmente una operación que lleva tiempo, que tu haces mientras desarrollas una aplicación. En el siguiente ejemplo, notarás que en algunos casos donde variables **QueryDef** son creadas, son desechadas al final del procedimiento. Esto no es una práctica común y esta hecha solo para propósitos de ilustración solamente.

Creando una nueva pregunta

Para crear una nueva **QueryDef**, tu debes de usas el método **CreateQueryDef** para crear una pregunta dentro de la base de datos especificada, entonces asigna a esta la variable declarada **QueryDef**:

```
Dim Db As Database, Q As QueryDef
Set Db = OpenDatabase ("Biblio.mdb")
Set Q = Db . CreateQueryDef ("New Query")
```

Nota : Las **QueryDefs** pueden ser creadas solo en Visual Basic y Microsoft Access.

Siempre que creas o modificas una pregunta en Visual Basic, es inmediatamente salvada. Desde Visual Basic no puedes salvar algo sin darle un nombre, debes proporcionarle un nombre a la pregunta cuando la creas.

Para definir la pregunta, asigna una cadena SQL a la propiedad de la variable **QueryDef**:

```
Q.SQL = "SELEC Subtotal FROM Orders; "
```

También puedes crear y definir la pregunta en un solo paso, proporcionando la cadena SQL cuando creas la pregunta:

```
Set Q = Db. CreateQueryDef ( "New Query", " SELEC Subtotal FROM Orders;" )
```

Ejecutando una pregunta

Una vez que ha sido definida una pregunta con una cadena valida SQL, puedes correr la pregunta creando una variable **Dynaset** sobre la **QueryDef**:

```
Set DataRecs = Q.CreateDynaset ( )
```

Cerrando una pregunta

Cuando tu estas finalizando con una variable **QueryDef**, tu debes cerrarla:

```
Q. Close
```

Borrando una pregunta

Visual Basic salva tus variables **QueryDef** en la base de datos cuando tu las creas, puedes querer borrar preguntas que son solo temporales. Para borrar una pregunta, usa el siguiente código:

```
DB. DeleteQueryDef "New Query"
```

No puedes borrar una pregunta que esta en uso. Si tu tienes variables **QueryDef** que refieren a una pregunta, debes cerrarlas usando el método **Close** antes de intentar borrar una pregunta, o un probable error de corrida ocurrirá.

Usando preguntas existentes

Si tu quieres usar una pregunta **SELECT** existente, para retornar un grupo de registros, tu puedes abrir la pregunta en código, del mismo modo que tu abres una tabla. La principal diferencia es que cuando tu abres una variable **Table** sobre una tabla existente, tu puedes manipular inmediatamente los datos en la tabla, pero cuando tu abres una variable **QueryDef**, sobre una pregunta existente, tu puedes manipular solo la pregunta y no los datos. Para obtener el dato recobrado por la pregunta, tu debes abrir un **Dynaset** o **Snapshot** sobre la pregunta:

```
Dim Db As Database, Q As QueryDef, dataRecs As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set DataRecs = Q.CreateDynaset ( )
Q . Close
```

Tu puedes abrir un **Dynaset** o un **Snapshot** directamente sobre una pregunta sin usar una variable **QueryDef**. Sin embargo, variables **QueryDef** te permiten ejecutar acciones de preguntas como con las preguntas **SELECT**. Las acciones de las preguntas no regresan registros, no abras un **Dynaset** o un **Snapshot** sobre ellas. En lugar de ello, usa el método **Execute** con la variable **QueryDef**:

```
Dim Db As Database, Q As QueryDef
Set Db = OpenDatabase ("Biblio.mdb")
Set Q = Db.OpenQueryDef ("Delete Old Titles)
Q.Execute
Q.Close
```

Intentar abrir un **Dynaset** o un **Snapshot** sobre la acción de una pregunta, producirá en un error, de igual modo que el desempeño del método **Execute** sobre una pregunta **SELECT**. Abrir un **Dynaset** o un **Snapshot** sobre un parámetro de una pregunta sin primero ajustar estos parámetros, también produce un error de corrida, para mayor información ver la sección "Usando parámetros de preguntas", más adelante en este capitulo.

Nota : Cuando tu ejecutas una pregunta con una variable **QueryDef**, tu código suspende la ejecución hasta que la pregunta es completada. Durante este tiempo tu puedes mostrar "un reloj de arena" como cursor si la pregunta es demasiado larga.

Modificando preguntas existentes

Tu puedes modificar la definición de una pregunta existente ajustando la propiedad SQL de una variable **QueryDef**. Por ejemplo, el siguiente código agrega a el campo "Zip" la pregunta SQL salvada en la **QueryDef** "Title Author Publisher" de Biblio.mdb:

```
Dim Db As Database, Q As QueryDef, qTemp As QueryDef, DataRecs As Dynaset
Dim SQL As String, I As Long, qTempName As String
Set Db = OpenDatabase ("Biblio.mdb")
Set Q = Db . OpenQueryDef ("Title Author Publisher")
SQL = Q.SQL
Q.Close
I = InStr (1, SQL, "[State]") + 7
' Encuentra el final del campo y concatena esta cadena "[State]"
SQL = Left (SQL, I) & " & ' ' & [ Zip ] " & Right (SQL, Len(SQL) - I)
qTempName = "Temp" & Now
Set qTemp = Db . CreateQueryDef (qTempName)
qTemp . SQL = SQL
Set DataRecs = qTemp . CreateDynaset ( )
qTemp . Close
Db . DeleteQueryDef QTempName
DataRecs . Close
Db . Close
```

Nota : Cuando ajustas la propiedad SQL de una variable **QueryDef**, Visual Basic inmediatamente salva la pregunta modificada en la base de datos. Si tu quieres cambiar la pregunta solo temporalmente, debes hacer una copia de la propiedad SQL antes de hacer cualquier cambio, y entonces restaura la propiedad con esta copia cuando tu lo hayas hecho. Por ejemplo, crea una nueva variable **QueryDef** y la bórrala cuando la hayas terminado. En un ambiente multiusuario, usualmente querrás evitar cambiar una pregunta que otros usuarios puedan estar usando, crear una nueva variable **QueryDef** es la mejor elección.

Manipulando Recordsets con variables QueryDef

Tu puedes usar las variables **QueryDef** para hacer las mismas cosas que puedes hacer con objetos **Recordset**. Por ejemplo, el siguiente código usa una **QueryDef** para encontrar todos los registros en la tabla "Titles" donde el "Year Publisher" vale 0 y actualiza aquel valor a 1992.

```
Dim Db As Database, Q As QueryDefSQL
On Error Resume Next
Set Db = OpenDatabase ("Biblio.mdb")
SQL = "Update titles set [Year Published] = 1992"
SQL = SQL + "WHERE [Year Publisher] = 0"
Set Q = Db . CreateQueryDef ("Fix Year", SQL)
BeginTrans
        Q . Execute
        If Err Then
          Rollback
          MsgBox "Failed: " & Error
        Else
```

```
        CommitTrans
        MsgBox "Sucedio"
    End If
Q . Close
Db . DeleteQueryDef ("Fix Year")
Db . Close
```

Cuando una variable **QueryDef** es ejecutada dentro de una transacción **Rollback**, la transacción hace los cambios hechos por la variable **QueryDef**. Un error ocurre si tu ejecutas una variable **QueryDef** que no puede completar esta acción.

Tu código puede construir la cadena SQL que define la pregunta, tu puedes ejecutar operaciones que pueden adaptarse ellas mismas a los cambios de la estructura de la bases de datos. Esto es diferente que con parámetros de preguntas. Mientras que los parámetros de las preguntas te permiten ejecutar operaciones basadas en condiciones no definidas hasta que la pregunta es corrida, tu no puedes armar una pregunta con el nombre de una tabla, como uno de sus parámetros. Esto significa que no puedes crear un objeto de una pregunta que borre todos los registros en una tabla que tu especifiques. Una **QueryDef** sin embargo, hace este tipo de operaciones fácilmente.

```
Sub EmptyTable (tableName As String)
        Dim Db As Database, qDelete As QueryDef, SQL As String
        Set Db = OpenDatabase ("Biblio.mdb")
        SQL = "DELETE FROM " & tableName & ";"
        Set qDelete = Db . CreateQueryDef ("EmptyTable", SQL)
        qDelete . Execute
        qDelete . Close
        Db . DeleteQueryDef ("EmptyTable")
        If DbName < > "" Then Db.Close
End Sub
```

Usando **QueryDefs** para construir preguntas dinámicamente, puedes usar técnicas de programación avanzada.

Usando los parámetros de las preguntas

Un parámetro de una pregunta espera uno o más valores para ser pasado en un tiempo de corrida. Si tu estas creando un parámetro de una pregunta, debes incluir la declaración de los **PARAMETERS** y de los parámetro de información. Visual Basic insertará cada parámetro o argumento dentro de la cadena SQL, de como la **QueryDef** debe ser procesada durante la ejecución.

Es responsabilidad de tu código llenar los parámetros con sus propios valores, antes que la pregunta sea ejecutada.

El siguiente código es un ejemplo de parámetros de preguntas de una base de datos. Tus parámetros de preguntas no necesitan ser así de complejos.

```
PARAMETERS [state Wanted] Text;
SELECT DISTINCTROW
Titles . Title, Titles . [Year Published], Authors . Author, Publishers . Name
AS Publisher, [City] & ", " & [State] AS Location
FROM Publishers, Titles, Authors, Publishers
```

```
RIGHT JOIN Titles ON Publishers . PubID = Titles . PubID , Titles
LEFT JOIN Authors ON Titles . Au_ID = Authors . Au_ID
WHERE ( ( Publishers . State = [State Wanted] ) )
ORDER BY Titles . Title
WITH OWNERACESS OPTION ;
```

El siguiente código abre la **QueryDef** que contiene el parámetro **State** mostrado arriba. Recuerda que tu aplicación debe completarse con un valor apropiado para cada parámetro, especificado en la sección parámetros de las preguntas.

```
Dim MyDB As Database, NyQuery As QueryDef, MySet As Snapshot
Set MyDB = OpenDatabase ("Biblio.mdb")
Set MyQuery = MyDB . OpenQueryDef ("Get State")
MyQuery ! [State Wanted] = "NY"
Set MySet = MyQuery . CreateSnapshot ( )
```

Tu puedes obtener los parámetros que una pregunta requiere usando el método **ListParameters** sobre una **QueryDef**. El método **ListParameters** devuelve un **Snapshot** con un registro para cada parámetro usado por la pregunta. Si el **Snapshot** regresado por el **ListParameters** no contiene registros, entonces la pregunta no toma parámetros
De otro modo, el **Snapshot** contiene registros comprometidos a los dos campos, como se muestra en la siguiente tabla.

Campo	Contenido	Tipo de dato Visual Basic
Nombre	Nombre de el parametro	String
Tipo	Tipo de el parámetro	Long

El siguiente ejemplo muestra los nombres y tipos de datos de todos los parámetros para una pregunta especificada en la ventana Debug.

```
Sub DumpParams (QueryName)
        Dim Db As Database, qAny As QueryDef, ssDump As Snapshot
        Set Db = Opendatabase ("Biblio.mdb")
        Set qAny = Db . OpenQueryDef(QueryName)
        Set ssDump = qAny . ListParameters ( )
        If ssDump . RecordCount > 0 Then
          Do Until ssDump . EOF
                Debug . Print ssDump . name, ssDump . Type
                ssDump . MoveNext
          Loop
        Else
          Debug . Print " ' " & QueryName & " ' Is not a parameter query"
        End If
        ssDump . Close
        qAny . Close
        Db . Close
End Sub
```

IV.5 Técnicas Avanzadas

La siguiente sección habla acerca de varias técnicas que puedes usar para agregar controles al manejo de tus datos.

Los temas incluidos son :
- Manejo de campos largos.
- Uso de métodos SQL.
- Visualizando variables **Recordset**.
- Uso de transacciones.

Manejando Campos Largos

Variables **String** en Visual Basic no pueden ser mas largas de 64K, o 65,355 bytes. Esto es adecuado para manipular campos texto, campos Memo y Binarios largos, pueden ser mas largos que 64 K. Para manipular campos largos en variables **Recordset**, Visual Basic proporciona tres métodos :

Método	Propósito
AppendChunk	Agrega los contenidos de una variable especificada a el campo.
FieldSize	Devuelve el tamaño de el campo.
GetChunk	Devuelve una porción especificada de el campo.

Estos métodos operan directamente sobre campos Memo y Binarios largos. Los puedes usar para manipular campos largos por rompimiento de campos dentro de trozos pequeños, suficientes para el manejo de Visual Basic. por ejemplo, el siguiente código copia los contenidos de un campo largo a un **Dynaset** dentro de un campo en otro **Dynaset**.

```
Sub CopyLarge ( dsSrc As Dynaset, SrcField, dsDest As Dynaset, DestField)
        Dim ChunkSize, ChunkStart, TotalSize, Chunk
        ChunkSize = 16384
        BeginTrans
          DsDest . Edit
          dsDest (DestField) = ""
          TotalSize = dsSrc (SrcField) . FieldSize ( ) -1
          For ChunkStart = 0 To TotalSize Step ChunkSize
                Chunk = dsSrc( SrcField ) . GetChunk (ChunkStart, ChunkSize)
                dsDest (DestField) . AppendChunk (chunk)
          Next
          dsDest . Update
        CommitTrans
End Sub
```

Es más eficiente para ejecutar operaciones de escritura, tales como **AppendChunk**, dentro de una transacción (**BeginTrans**, **CommitTrans**), porque así habrá menos escritura al disco. En resumen, si ocurren errores durante la aplicación del método **AppendChunk**, tu puedes usar la sentencia **Rollback** para restaurar el estado previo de la tabla afectada. Para mayor información sobre transacciones, ver la sección "Usando transacciones", mas adelante en este capitulo.

Usando el método FieldSize

El método **FieldSize** regresa el total de los datos en un campo Memo. Asegúrate de asignar este valor a una variable **Long**, como el largo de un campo Memo puede exceder 32,768 (La capacidad de un **Integer**). Por ejemplo, el siguiente código regresa el largo de un campo seleccionado para comentarios publisher's :

```
Dim Db As Database
Dim Ds As Dynaset
Dim CommentLength As Long
Set Db = Opendatabase ("Biblio.mdb")
Set Ds = Db . CreateDynaset ("Publisher Comments")
CommentLengh = Ds ("Comments") . FieldSize ( )
```

Usando el método GetChunk

El método **GetChunk** extrae todo o una pieza de un campo Memo basado sobre un **offset** (o byte de inicio). **GetChunk** es aplicado en contra de un campo Memo en un **Dynaset** seleccionado. Por ejemplo, el siguiente código extrae los primeros 4K de datos de un **Dynaset** creado en el ejemplo previo :

```
MyString$ = Ds ("Comments") . GetChunk ( 0, 4096 )
```

Hay métodos para pasar el número de bytes ha saltar (el **offset**) y el número de bytes a extrae. Si tu pasas a 0 **Offset**, el primer byte de el campo Memo será incluido. El tamaño puede ser cualquier valor menor que 64K, pero no mas grande que el tamaño de tu variable **String** o text box. Si preguntas por una cadena mas larga que el método **FieldSize**, el campo entero será extraído. Una vez que el campo es copiado dentro de tu cadena destino, tu puedes usar la función **Len** para determinar el tamaño copiado.

Tu necesitas ejecutar **GetChunk** como muchas veces has necesitado, para extraer el campo Memo entero. Tu puedes usar el **AppendChunk** para reensamblar las piezas.

Si asignas el valor de regreso de **GetChunk** a una variable, y el tamaño es mas grande de 64K, un error ocurrirá.

Usando el método AppendChunk

El Método **AppendChunk** concatena datos de cadena (**string**) a un campo Memo en un **Dynaset** especificado. Puedes usar **AppendChunk** para construir un nuevo campo Memo de componentes de cadenas que cuando son concatenadas son mas largas de 64K.

El Método **AppendChunk** es aplicado en contra de un campo Memo especificado en el registro grabado de un **Dynaset**. Por ejemplo, el siguiente código anexa datos de un control text box a el campo Memo en el **Dynaset** creado en el ejemplo previo.

```
Ds ("Comments") . AppendChunk ( ( Text1 .Text ) )
```

Puedes usar los métodos **AppendChunk** y **GetChunk** para crear un campo Memo que exceda la limitación de Visual Basic de 64K, para el tamaño de las cadenas. Ciertas operaciones (copiar, por ejemplo) involucran cadenas temporales. Si la cadena (**string**) esta limitada, puedes necesitar trabajar con **chunks** de un campo Memo en lugar de un campo entero.

Si tu quieres limpiar los contenidos del registro grabado, ajusta el campo a cero, para el tamaño de la cadena. El siguiente ejemplo muestra como limpiar un campo llamado "Comments".

```
DS . Edit
Ds ("Comments") = ""
DS . Update
```

Si no hay un registro grabado cuando usas el método **AppendChunk**, ocurrirá un error.

Usando métodos SQL

Cualquiera de los métodos **Recordset** creados, excepto **OpenTable** permite el uso de una pregunta SQL para seleccionar, filtrar y ordenar los registros de un **Recordset**. Puedes usar una pregunta SQL con base de datos. Si deseas que una base de datos ODBC ejecute una pregunta, puedes usar la opción **DB_SQLPASSTHROUGH**.

Los métodos **Execute** y **ExecuteSQL** están diseñados para ejecutar sentencias SQL específicas. Estos métodos no retornan registros, estos métodos pueden afectar un número de registros, pero no pueden seleccionar ningún registro para ser regresado como un **Recordset**. Si tu usas una pregunta SQL que retorna líneas con uno de estos métodos, un posible error aparecerá. La siguiente tabla resume como puedes usar las preguntas SQL con Visual Basic.

Métodos	Puede usar ODBC SQLPassThrough	Regresa un Recordset
CreateDynaset	Si	Si
CreateSnapshot	Si	Si
CreateQueryDef	Si	Si (Puede usarse en una pregunta)
ExecuteSQL	Solo ODBC	No
Data1.**Recordset**	Si	Si

El método Execute

Si tu quieres ejecutar un comando SQL que no retorna datos de registros en contra de tu base de datos, esto es simple y rápido usando el método **Execute**. Usar el método **Execute** es el mejor camino para ejecutar muchas operaciones de acciones de preguntas. Unas preguntas usan un comando SQL para hacer cambios a la base de datos o para ejecutar una acción que involucra un gran número de registros. Usando SQL para llevar acabo esas tareas es más eficiente, que pasar datos a través de tu aplicación registro por registro para hacer cambios específicos.

Por ejemplo, para borrar todos los registros en la tabla "Titulos" de la base de datos Biblio.mdb, donde el titulo del libro contiene la cadena "relational", podrías usar el siguiente código:

```
Dim Db As Database
Set Db = OpenDatabase("Biblio.mdb")
Db Execute "Delete FROM Titles WHERE Title LIKE  '*relational*' "
```

El método ExecuteSQL

A diferencia del método **Execute**, el método **ExecuteSQL** siempre se desvía de la base de datos Visual Basic y ejecuta un comando SQL en contra del servidor de la base de datos externa ODBC. Esto no puede ser usado contra ninguna otra base de datos ODBC. Como el método **Execute**, **ExecuteSQL** no puede regresar registros de datos, pero regresa un valor **Long** que indica el número de líneas afectadas por la pregunta (query) SQL. Por ejemplo, tu podrías usar el siguiente código para abrir la base de datos ODBC "Pubs", borrar todos los registros en la tabla "Titles" donde esta el titulo incluyendo la palabra "relational", y entonces determina el número de registros borrados:

```
Dim DB As Database
Dim R As Long
Dim Conn As String
Conn = "ODBC; Uid=Holly; Pwd=Dog; DBQ=Pubs; DSN=Work;")
Set DB = OpenDatabase ("", False, False, Conn$)
R = DB . ExecuteSQL ("DELETE Titles WHERE Title LIKE '%relational%' " )
Debug . Print "There were " & R & "records deleted "
```

Nota : La sintaxis para la pregunta ha sido escrita para conformar la sintaxis del servidor externo ODBC. La sintaxis de la sentencia SQL debe conformarse a los requerimientos del servidor donde se esta ejecutando.

Reglas de código para preguntas SQL

Aunque es posible escribir una pregunta en algunos dialectos SQL que regresan más de un grupo de resultados, solo el primer juego de resultados será regresado a tu aplicación. Si tu invocas un procedimiento SQL almacenado, que pase parámetros anteriores, estos parámetros serán ignorados. Recuerda que debes conformar la sintaxis para el dialecto SQL que tu maquina soporte. La mejor fuente de información sobre la sintaxis es proporcionada por el vendedor del servidor SQL.

Visualizando variables Recordset

Algunas veces desearás mantener más de un registro grabado en un **Recordset**. Por ejemplo, supón que quieres copiar el registro grabado en una variable **Dynaset**. Cuando usas el método **AddNew**, el registro grabado llegará a ser el nuevo registro, así no tendrás un largo camino de referencias para el registro que contiene los valores que quieres copiar. Si tu tratas de usar un **Bookmark** para saltar hacia atrás y adelante los dos registros, el nuevo registro se perderá al moverte atras con el registro previo.

El código es fácil para escribir (y mas eficiente), si tu ejecutas el método **AddNew** sobre la copia del **Dynaset** (**Clone**). De esta forma, el registro grabado del **Dynaset** original contiene los valores que tu quieres copiar, mientras el registro grabado de la copia (**Clone**), es el nuevo registro dentro del cual tu copias los valores.

```
Sub CopyCurrentRec (ds As Dynaset)
        Dim i As integer, dsCopy As Dynaset
        Set dsCopy = ds . Clone ( )
        dsCopy . AddNew
        For i = 0 To ds . Fields . Count - 1
          ds Copy ( i ) = ds ( i )
        Next i
```

```
        dsCopy . Update
        dsCopy . Close
End Sub
```

Hay sutiles pero importantes diferencias entre la copia de un **Dynaset** (clone) y un **Dynaset** original. Esencialmente, la copia puede mantener más de un puntero del registro grabado en el mismo **Dynaset**, mientras que otro **Dynaset** te permite tener dos o más grupos de registros completamente independientes. Considera el siguiente código:

```
Dim Db As Database, ds1 As Dynaset, ds2 As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set ds1 = Db . CreateDynaset ("Titles")
Set ds2 = ds1 . CreateDynaset ( )
```

En este fragmento de código, el segundo **Dynaset** esta creado sobre el primer **Dynaset**. Cada uno tiene su propio grupo de registros, el cual debe ser ordenado diferente, y el cual puede contener diferentes registros. Ellos no pueden compartir **Bookmarks**.

Ahora compara este código:

```
Dim Db As Database, ds1 As Dynaset, ds2 As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set ds1 = Db . CreateDynaset ("Titles")
Set ds2 = ds1 . Clone ( )
```

En este ejemplo, el segundo **Dynaset** es un clone de el primer **Dynaset**. Tal como, ellos comparten el mismo grupo de registros, pero cada uno puede tener una posición diferente en el grupo de registros. Ellos pueden compartir **Bookmarks** y ver los cambios del uno al otro, adiciones y borrados. Objetos **Dynasets** copiados **(Clone)**, no pueden ser filtrados u ordenados diferentemente que su gemelo.

Cerrando Transacciones

Muchas veces por razones de integridad de los datos debes considerar un grupo de operaciones como una unidad independiente. Por ejemplo, la transferencia de fondos de una cuenta de bancos hacia otras de dos operaciones, posicionar un débito en una cuenta y posicionar un crédito en la otra. Ambas operaciones deben tomar lugar, o no se podrá realizar. Si el débito ocurre sin su correspondiente crédito, entonces el dinero "Vanishes", resultará en una desafortunada cuenta para el poseedor. Si el débito ocurre sin el correspondiente débito, entonces el dinero "materializes" , resultará en un desafortunado banco. Las dos operaciones, débito y crédito, son una unidad independiente, o una transacción.

En sistemas manejadores de bases de datos, donde una transacción consiste de una serie de operaciones, la transacción es ejecutada irreversiblemente, si todas las operaciones en la transacción son exitosas. Si alguna de las operaciones falla, la transacción es regresada y el dato es regresado ha el estado en que la transacción empezó.

Visual Basic proporciona tres sentencias para iniciar transacciones.

Sentencia	Propósito
BeginTrans	Inicia una transacción
CommitTrans	Comete todos los cambios hechos desde el mas reciente **BeginTrans**
Rollback	Regresa todos los cambios hechos desde el mas reciente **BeginTrans**

El siguiente código ilustra un ejemplo de como podrías usar sentencias de transacción, para implementar una función **TransferFunds** para pasar dinero entre dos cuentas:

```
Function TransferFunds (FromAccount, ToAccount, CustomerID, Amount)
        Dim Db As Database, Source As Dynaset, Destination As Dynaset
        On Error Resume Next
        Set Db = OpenDatabase ("ACCOUNTS", False, False, "")
        Set Source = Db . CreateDynaset (FromAccount)
        Set Destination = Db . CreateDynaset (ToAccount)
        If Err Then
           TransferFunds = err
           Source . Close : Destination . Close : Db . Close
           Exit Funtion
        End If
        Source . FindFirst " [Cust ID] = " & CustomerID
        Destination . FindFirst " [Cust ID] = " & CustomerID
        If  Not (Source . NoMatch Or Destination . NoMatch) Then

           If Source!Balance > = Amount Then
                 BeginTrans
                 Source . Edit
                 Source ("Balance") =  Source ( "Balance") - Amount
                 Source . Update
                 Destination . Edit
                 Destination ("Balance") = Destination ("Balance") + Amount
                 Destination . Update
                 If Err Then
                    TransferFunds = -1
                    RollBack
                 Else
                    TransferFunds = 0
                    CommitTrans
                 End If
           Else
               TransferFunds = -2
           End If
         Else
           TransferFunds = -3
        End If
        Source . Close
        Destination . Close
        Db . Close
End Function
```

Importante : El uso de las sentencias y métodos de transacciones estan soportados solo para tablas de bases de datos Microsoft Access y Visual Basic. Otros formatos de bases de datos no soportan transacciones. Estas permiten usar sentencias de transacciones en sentencias SQL para enviar a bases de datos externas ODBC que soporten transacciones.

Si creas un **Dynaset** sobre una pregunta que incluya ambas tablas, como tablas foráneas unidas, tu no podrás seguramente ejecutar transacciones sobre el **Dynaset**. Tu puedes regresar (**Rollback**) los cambios hechos por tablas Microsoft Access/Visual Basic, pero no puedes hacer los cambios hechos a tablas foráneas (externas). Cada **Dynaset** tiene una propiedad de transacción que regresa a True, si el **Dynaset** puede regresar (**Rollback**) cambios a todos los campos en el **Dynaset**, y False si no puede.

Usando transacciones múltiples

Mientras una transacción esta pendiente, tu puedes usar otra transacción. La segunda transacción esta anidada dentro de la primera transacción, esta debe ser ejecutada o regresar antes que la primera pueda ser completada. A menos que estés usando tablas ODBC, tu puedes tener cinco transacciones anidadas. Cuando usas tablas ODBC, puedes usar **PassThroughSQL** para implementar una transacción de control, usando la sintaxis SQL definida para la base de datos ODBC.

En el siguiente código, una transacción abarca todos los cambios hechos a ambas tablas, Products y Categories. Dentro de esta transacción, cada cambio hecho a un registro en la tabla Products, llegará a ser otra transacción anidada.
Si alguna de las actualizaciones a la tabla Products falla, la transacción inmediata es regresada a lo largo de la transacción. Esto causa que todas las otras actualizaciones de la tabla Products sean regresadas. Solo si todos los cambios son exitosos algunos de los cambios son hechos.

```
Dim Db As Database, dsCat As Dynaset, dsProduct As Dynaset
Dim criteria, Failure
On Error Resume Next
Set Db = OpenDatabase ("xyz.mdb")
Set dsProduct = Db . CreateDynaset ("Products")
Failure = False
criteria = "[English Name] LIKE '*sauce*'"
BeginTrans                                    ' Fuera de la transacción.
      dsProduct . FindFirst criteria
      Do Until dsProduct . NoMatch
         BeginTrans                           ' Dentro de la transacción.
         dsProduct . Edit
         dsProduct ("Category ID") = "SAUC"
         dsProduct . Update
         If Err Then
    Failure = True
    Rollback                                  ' Dentro de la transacción.
  Else
     CommitTrans                              ' Dentro de la transacción.
  End If
  If Failure Then Exit Do
  dsProduct . FindNext criteria
 Loop
```

```
If Failure Then
    Roollback                                    ' Fuera de la transacción.
    MsgBox "Failed : " & Error
Else

        Set dsCat = Db . CreateDynaset("Categories")
        dsCat . AddNew
        dsCat ("Category ID") = "SAUC"
        dsCat("Category Name") = "Sauces"
        dsCat ("Description") = "Sauces"
        dsCat . Update
        If Err Then
            Rollback
            MsgBox "Failed: " & Error
        Else
            CommitTrans
            MsgBox "Succeeded"
        End If
  End If
Close DsCat : Close DSProduct : Close Ds
```

Después de regresar (**Rollback**) una transacción, el registro grabado esta indefinido. Debes ajustar un **Bookmark** antes de usar la sentencia **BeginTrans**, si quieres regresar a tu posición original en el **Recordset** después de regresar (**Rollback**) una transacción.

Importante : Si tu inicias una transacción, debes terminarla o regresar (**Rollback**) a la misma. Si olvidas cualquiera de las dos cosas, un error eventual ocurrirá cuando tengas demasiadas transacciones pendientes. También encontrarás un error si tienes una transacción pendiente en una base de datos y tu intentas cerrar esa base de datos. Las transacciones deben ser lo más cortas posibles y nunca usadas por el usuario cuando sea posible tener el control y no completar la transacción. Por ejemplo, tu aplicación nunca debe iniciar una transacción o cerrar una operación y indicar al usuario que no puede ser respondida inmediatamente.

Nota también que operaciones de transacción son funcionalmente globales para la sección. Una sección simple es creada cuando tu aplicación inicia. Cuando tienes abiertas más de una base de datos, todo los proceso de las transacción funcionarán contra todas las bases de datos abiertas. No puedes regresar (**Rollback**) transacciones en una base de datos sin regresar las transacciones pendientes en todas las otras bases de datos. Por esta razón, debes trabajar cuidadosamente cuando implementes una transacción con múltiples bases de datos.

Capitulo V

"Usando controles de Visual Basic con objetos para acceder a datos".

La edición profesional de Visual Basic agrega características a el Data Control, para hacerlo mas flexible y capaz de tratar con las contingencias que comúnmente ocurren en aplicaciones complejas en una base de datos. Este capítulo se enfoca a estas nuevas características y muestra como usarlas para operaciones de acceso a datos.

También esta incluida información sobre los tres controles que pueden ser ligados, incluidos en la edición profesional (3D Check box, 3D panel, y Masked Edit). Ademas, podrás encontrar información acerca de como usar controles estándar (unbound) con el Data Control, para indirectamente manipular datos extraídos de una base de datos.

Contenidos :

V.1 Usando el Data Control en la edición profesional

Como en la edición estándar, puedes usar el Data Control para llevar acabo muchas de las funciones del manejo de la base de datos para tu aplicación. Con la edición profesional, sin embargo, aprovechas la habilidad para usar la sentencia **Dim**, para crear cualquier objeto para acceder a los datos que necesites. Puedes usar estos objetos para acceder a los datos, junto con el Data Control, para incrementar tu flexibilidad cuando trabajes con una base de datos. Adicionalmente puedes llevar acabo otras tareas con la edición profesional, incluidas las siguientes:

- Crear variables **Recordset** basadas en variables originales **Dynaset** proporcionadas por la propiedad **Recordset**.
- Modificar la estructura de una base de datos, agregando tablas, campos, índices, o removiendo tablas e índices
- Ejecutar otras **Database** (base de datos) y métodos **Recordset**, no disponibles en la edición estandar, incluyendo el **Seek**, **Find**, **CreateDynaset**, **CreateSnapshot** y métodos **OpenTable** .
- Declarar variables objeto y usar estas como notación **"shorthand"** para **Database** y propiedades **Recordset**.

Usando la propiedad Database

Una vez que tengas abierta la base de datos con el Data Control, puedes usar la propiedad **Database** como si hubiera sido creada por la función **OpenDatabase**. Por ejemplo :

```
Dim Db As Database
Data1.DatabaseName = "Biblio.mdb"
Data1.Refresh
Set Db = Data1.Database
```

También para dimensionar una variable **Database**, tu puedes usar el siguiente código equivalente:

```
Dim Db As Database
Set Db = OpenDatabase ("Biblio.mdb")
```

Nótese que la variable **Database** Db, en el primer ejemplo es equivalente a la variable **Database** Db, usada en el segundo ejemplo. Puedes usar cualquier variable para ejecutar cualquiera de los métodos **Database**, examinar cualquiera de las propiedades **Database**, modificar cualquier miembro de la colección **TableDefs** o modificar cualquiera de los **Fields** (campos) o **Indexes** (índices).

Por otra parte, una de las ventajas del acceso a datos en la edición profesional, es que tu puedes dimensionar cualquiera de las variables **Database**, **TableDef**, **Field** e **Index**, también puedes examinar y modificar los miembros de los **TableDef**, **Fields**, o **Indexes**.

Usando la propiedad Recordset

Como la propiedad **Database**, la propiedad **Recordset** tiene este equivalente en código programático, para acceder a los datos. La propiedad **Recordset** proporciona una variable **Dynaset**, la cual puedes usar para manipular los datos grabados. Por ejemplo, para crear un **Dynaset** programáticamente, puedes usar el siguiente código :

```
Dim Db As Database
Dim Ds As Dynaset
Set Db = OpenDatabase ("Biblio.mdb")
Set Ds = Db.CreateDynaset ("Titles")
```

Para crear un **Dynaset** en un **Recordset**, tu puedes usar el siguiente código equivalente:

```
Dim Ds As Dynaset
Set Ds = Data1 . Recordset
```

Usando el Data Control, puedes crear variables adicionales **Dynaset**, **Snapshot**, y **Table** con la sentencia **Dim**. Para mayor información sobre la creación y manipulación de objetos **Recordset**, ver el capítulo 4 "Creando programas que manipulen datos".

Usando SQL en la propiedad RecordSource

Si quieres crear un **Dynaset** con el Data Control, puedes usar una pregunta **SQL** en lugar del nombre de la Tabla (**Table**) de la base de datos en la propiedad **RecordSource**. Esta pregunta **SQL** debe cumplir con las mismas reglas para las preguntas **SQL** usadas para crear otros objetos **Recordset**. Las sentencias **SQL** puede también enviar la forma de ordenamiento con la cláusula **ORDER BY**. Por ejemplo, puedes usar el siguiente código para construir un **Dynaset** que recupere registros de "Titles" para Publishers que tienen oficinas en California :

```
Dim Ds As Dynaset
Data1.DatabaseName = "Biblio.mdb"
Data1.RecordSource = "SELECT * FROM Titles, Publishers WHERE Publishers.PubId = Titles.PubID
AND Publishers.State = ' CA ' "
Data1.Refresh
Set Ds = Data1.Recordset
Do While Not Ds.EOF
        Print Ds ("Title")
        Ds.MoveNext
Loop
```

Cuando uses una pregunta **SQL** para crear un **Recordset**, necesitas usar la sintaxis Visual Basic **SQL** a menos que uses la opción **SQLPassThrough**, o uses el método **Execute** o **ExecuteSQL** para enviar preguntas **SQL** a un servidor ODBC para su procesamiento. En casos donde otro servidor sea quien procesará la pregunta (query), deberás usar la sintaxis **SQL** que entienda ese servidor.

Para mas información sobre el uso de **SQL** en **Recordsets**, ver capítulo IV "Creando programas que manipulen datos".

Tips generales y técnicas

Esta sección discute varias técnicas que puedes usar para crear preguntas fácilmente. Los tópicos incluyen lo siguiente :

- Creando preguntas **SQL**.
- Usando variables como notación **Shorthand**.
- Trabajando con múltiples **Recordsets** .

Creando preguntas SQL

En algunos casos, puedes querer crear una pregunta (query) que retorne un valor que no tiene asociado ningún nombre de un campo. Por ejemplo, si tienes una tabla "Parts" con campos "Price" y "Quantity", puedes calcular el total de todas las partes en la tabla con esta pregunta :

SELECT Quantity * Price **FROM** Parts

Desafortunadamente, no podrás unir todos los valores regresados. Una opción para extraer el dato, sería utilizando el siguiente código :

```
Total = Data1 . Recordset ( 0 )
```

Si cambias la pregunta (query) para incluir un alias llamado "Total", puedes utilizar el siguiente código :

SELECT Quantity * Price **AS** Total **FROM** Parts

Desde que calculaste el valor usando el campo Total, puedes ahora agregar una etiqueta de control a este resultado. Es buena idea usar una etiqueta de control para mostrar el valor de un campo alias.
En el ejemplo anterior, el dialecto para la cláusula **As** es de Microsoft Access **SQL**. Nota que otros dialectos **SQL** tal vez no soporten la cláusula **As**, pero si soporten la habilidad de crear campos alias. Por ejemplo Microsoft **SQL** Server soporta alias, codificándolo después de una expresión. La misma pregunta (query) escrita para Microsoft **SQL** Server es :

SELECT Quantity * Price Total **FROM** Parts

El siguiente ejemplo somete la pregunta (query) **SQL**, mostrado anteriormente, para un servidor ODBC y procesa el resultado regresado en el **Recordset**.

```
Dim Ds As Dynaset
Data1.DatabaseName = "Work"
Data1.Connect = "odbc; pwd = Scholar; dbq = MyDb; uid = Vicky"
Data1.RecordSource = "SELECT Item, Quantity * Price Total FROM Sales"
Data1.Refresh
```

```
Set Ds = Data1.Recordset
Do While Not Ds.EOF
        Print Ds ("Item")
        Print Ds ("Total")
        Ds.MoveNext
Loop
```

Recuerda que puedes también pasar sentencias **SQL** que no retornan registros a través del servidor ODBC **SQL** para su procesamiento. Para hacer esto, usa la opción **SQLPassTrough** con cualquier otra sentencia **ExecuteSQL** o el método **Execute**.

Usando variables con sintaxis compleja (notacion Shorthand)

Como notaste una ventaja de la edición profesional es la capacidad que te proporciona para crear variables tipo objeto de datos, usando la sentencia **Dim**. Con tu pareja de variables objeto la sentencia **Set**, puede referirse a ellos como complejos objetos de datos usando variables de palabra simples como un tipo de notación **shorthand**.

Por ejemplo, cuando trabajas con el Data Control, puede necesitar referirte a la propiedad **Recordset**. Dos ejemplos son ilustrados a continuación.

En la edición estandar, puedes usar la sintaxis completa para referirte a un **Recordset** :

```
Data1 . DatabaseName = "Biblio.mdb"
Data1 . RecordSource = "Publishers"
Data1 . Refresh
Do While Not Data1 . RecordsetEOF
        Print Data1 . Recordset ("Name").
        Print Data1 . Recordset ("City").
        Print Data1 . Recordset ("State"). .
        Print Data1 . Recordset ("Zip")
        Data1 . Recordset . MoveNext
Loop
```

En la edición profesional puedes dimensionar variables de acceso a datos y usar estas como notación **shorthand**.

```
Dim Ds As Dynaset
Data1 . DatabaseName = "Biblio.mdb"
Data1 . RecordSource = "Publishers"

Data1 . Refresh
Set Ds = Data1 . Recordset
Do While Not Ds.EOF
        Print Ds ("Name").
        Print Ds ("City") . Ds("State") . . Ds("Zip")
        Ds . MoveNext
Loop
```

Trabajando con Recordsets

Puedes usar el método **Clone** para crear versiones múltiples del mismo **Recordset**. Esto te permite tener mas de un puntero dentro de un **Recordset** a la vez. Por ejemplo, cuando usas uno de los métodos **Find** sobre una variable **Dynaset**, o el método **Seek** sobre una tabla. Si usas un **Clone** de un **Recordset** para proporcionar un segundo puntero a el **Recordset**, el registro original no es afectado por el método **Seek** o por uno de los métodos **Find** o **Move** ejecutados contra el **Clone**.

También puedes usar el **Bookmarks** de un **Recordset** para posicionar un registro dentro de un **Clone** de este **Recordset**.

Por ejemplo para un **Clone** de un objeto **Dynaset** con un Data Control, y dos posiciones de punteros grabados dentro de aquel objeto, puedes escribir el siguiente código:

```
Sub GetData_Click ( )
Dim i As Integer
Dim Ds As Dynaset
Dim DsClone As Dynaset
Dim BookMarks ( 2000 )
Data1 . DatabaseName = "Biblio.mdb"
Data1 . RecordSource = "Titles"
Data1 . Refresh

Set Ds = Data1 . Recordset
Set DsClone = Ds . Clone ( )
List1 . Clear
List2 . Clear
i = 0
Do While Not Ds . EOF And i < 2001
        List1 . AddItem Ds ( "Title")
        List2 . AddItem DsClone ("ISBN")
        BookMarks ( i ) = Ds . Bookmark
        i = i +1
        ' Muevete al próximo registro en cada Dynaset.
        Ds.MoveNext
        Dsclone.MoveNext
Loop
End Sub
```

En este caso estos dos **Dynaset** son **Clones**, cada uno está ordenado. Para ordenar los **Dynaset** diferentemente, puedes cambiar la propiedad **Sort** de cada lista de **Dynaset**, como se muestra en el ejemplo anterior.

Una vez que fueron llenadas las listas, el código en el evento **Click** para cada List box, permite el uso de una lista a través de cualquiera de las List box y escoger uno de los articulos. Este evento **Click** ajusta la propiedad original **Bookmarks** de un **Recordset** con el valor almacenado cuando la lista fue construida, basado en la propiedad Listn.**ListIndex**. Una vez ajustados los controles que pueden ser ligados (bound) que muestran el registro de entrada, son actualizados en el registro que contiene el valor escogido en el List box. Por ejemplo, si el usuario seleccionó un titulo del primer List box, los controles ligados (bound) mostrarán el registro grabado de entrada para aquel título. Si el usuario selecciona un número ISBN de un segundo List box, los controles ligados (bound) mostraran datos del registro ISBN.

El código del evento List box **Click** es como el siguiente:

```
Sub List2_Click ( )
Ds . Bookmark = BookMarks ( List2 . ListIndex )
End Sub

Sub List1_Click ( )
Ds . Bookmark = BooMarks ( List1 . ListIndex)
End Sub
```

Con registros **Recordset Clone** grabados con idénticos **Bookmarks**, puedes usar una tabla de **Bookmarks** para mover un objeto **Recordset**.

V.2 Usando controles que pueden ser ligados (controles bound)

La edición profesional proporciona tres controles que pueden ser ligados (controles bound), en adición a los cinco controles bound incluidos en la edición estándar. La siguiente tabla lista el total de los controles bound en Visual Basic.

Control	Icono en el Toolbox
Check box	
Image	
Label	
Picture	
Text box	
3D check box (sólo edición profesional)	
3D panel (sólo edición profesional)	
Masked edit (sólo edición profesional)	

Tips generales y técnicas

Visual Basic implementa con cada control bound tres propiedades que son : habilitar ese control para obtener información y escribir información a la base de datos, estas propiedades son mostradas en la siguiente tabla.

Propiedad	Descripción
DataChanged	Indica ya sea que un valor mostrado en un control bound a cambiado. No disponible en tiempo de diseño; la lectura/escritura es en tiempo de ejecución.
DataField	Especifica el nombre de un campo en la base de datos conectado al control bound. El List puede ser generado en tiempo de diseño o enviado manualmente. Lectura/Escritura en tiempo de ejecución.
DataSource	Especifica el nombre del Data Control al cual el control esta unido. Escritura solo en tiempo de diseño; lectura solo en tiempo de ejecución.

Las siguientes secciones describen las caracteristicas de los controles bound incluidos en la edición profesional.

Usando un control que puede ser ligado "3D Check Box"

El Control 3D Check Box puede ser ligado a un Data Control y mostrar los valores del campo para los registros grabados. El 3D Check Box puede ser ligado solo aun campo de un dato tipo **Boolean**. El Control 3D Check Box puede escribir valores para el dato enviado. Para mayor información ver el capitulo "3D Check Box".

Nota : Cuando el valor de un campo referenciado por la propiedad **DataField** es leído, este es convertido a un valor de propiedad válido, si es posible. Si el valor del campo es **Null**, entonces la propiedad **Value** es ajustada a dos, lo cual significa que el Check Box aparecerá en gris.

Usando un control que puede ser ligado "Panel 3D"

El control panel 3D puede ser ligado a un Data Control y mostrar los valores de los campos. El control 3D panel puede también escribir valores fuera del dato enviado.

Nota : Cuando el valor de un campo referenciado por la propiedad **DataField** es leído, este es convertido a una cadena de la propiedad **Caption**, si es posible. Si el dato enviado es actualizado, la cadena es convertida a el tipo de datos de el campo.

Usando un control que puede ser ligado "Masked Edit"

El Control Masked Edit puede ser ligado (control bound) a el Data Control y mostrar valores de los campos. Este control Masked Edit puede escribir valores fuera del dato enviado. Para mayor información, ver el capítulo "Masked Edit".

Nota : Cuando el valor de un campo referenciado por la propiedad **DataField** es leído, este es convertido a una cadena de una propiedad **Text**, si el dato enviado es actualizado, la cadena es convertida a el tipo de datos de el campo.

V.3 Usando un control que no puede ser ligado (control unbound)

Puedes usar también controles unbound para mostrar datos y hacer cambios aun **Recordset**.

Un ejemplo de uso de un control unbound con el Data Control es usando un List box para visualizar un juego de valores de un **Recordset**. Por ejemplo podrías tener una tabla que contenga códigos ID. Si al usuario de tu programa no le agrada memorizar todos los códigos, puedes proporcionar una lista de códigos validos ID para que el usuario pueda escoger. Tu código podría generar una pregunta (query) que extraiga la lista de códigos ID de la base de datos, entonces usa el Data Control, el objeto de acceso a datos , o ambos, para llenar la List box.

El siguiente ejemplo usa el Data Control para llenar un List box .

```
Sub FillPubsList (StateFromForm$)
Dim Ds As Dynaset
Data1 . DatabaseName = "Biblio.mdb"
Data1 . RecordSource = "SELECT PubID, Name FROM Publishers WHERE State = ' " &
StateFromForm$ & " ' "
```

```
Data1 . Refresh                    ' Abre la base de datos.
CustList.Clear                     ' Limpia el List box.
Set  Ds = Data1 . Recordset

Do While Ds . EOF = False
        CustList . AddItem Ds ( 0 ) & "." & Ds( 1 )        ' Envia los reultados.
        Ds . MoveNext
Loop
End Sub
```

Cuando corre este programa, produce una lista de códigos válidos publisher ID y nombres, y agrega estos a la List box. Si la pregunta **SQL** limita el alcance de los **Recordset** en la cláusula **WHERE**, la pregunta puede ser completada rápidamente como el número de renglones o líneas que debe ser mantenido en un mínimo.

Recuerda que la List box puede ordenar su contenido. Si tu **Recordset** esta ya ordenado, puedes querer deshabilitar la propiedad **Sort** de la List box.

Conclusiones

Análisis costo-benefico :

Visual Basic versión profesional 3.0, es un lenguaje de programación que no es muy estricto en cuanto a los requerimientos del sistema para funcionar, ya que sólo requiere de una computadora con procesador 386 con 4Mb en Ram como mínimo, y la versión 4.0 requiere de un procesador 486 con 8 Mb como mínimo. El costo de la versión 3.0 es de aproximadamente $2500.00, mientras que la versión 4.0 tiene un costo de 5047.28, ya que cuenta con nuevos controles diseñados para windows 95.

El desarrollo de aplicaciones en Visual Basic 3.0 tiene un costo de $150.00 diarios, mientras que el desarrollo de aplicaciones en la versión 4.0 sería de $250.00 diarios. Estos costos son por que el ambiente de trabajo que tienen dichas aplicaciones es muy visual. Ya que nos permiten desplazarnos fácilmente por cualquier módulo de la aplicacion.

Visual Basic nos proporciona un lenguaje de programación muy fácil de comprender y para trabajar en sistemas muy sencillos. Nos permite darle mucho mejor presentación a nuestras aplicaciones, además de una gran rapidez, lo que hace ser mas productivo nuestro negocio.

Hay otros lenguajes de programación que nos permiten desarrollar aplicaciones visuales, pero sobre otros estilos de programación, como podria ser Delphi, pero el estilo de programación es mas parecido a la programación en Pascal.

Glosario

- Data Manager : Es un programa que se carga cuando instalamos Visual Basic, y se encuentra en el grupo de trabajo del icono de Visual Basic.
- Software : Conjunto de instrucciones que hacen una tarea en particular.
- Hardware : Dispositivos físicos con los que cuenta una computadora.
- Servidor Local : Dispositivo que trabaja dentro del sistema.
- Servidor Remoto : Dispositivo que trabaja desde el exterior de nuestro sistema.
- Backup : Copia de información, en forma compactada.
- Mapear : Análisis que se hace de la estructura que forma nuestro dispositivo u objeto.
- Brackets : Signo que se utiliza para separar cierta información "[ ]".
- Linkear : Análisis que se hace de nuestra aplicación, para encontrar los componentes de dicha aplicación y relación que hay entre ellos.
- DML : Lenguaje de manipulación de datos.
- DDL : Lenguaje de definición de datos.
- Puntero : Dirección que permite determinar el lugar físico donde se encuentran determinados los datos.
- Inhabilitado : Que no se encuentra activo.

Apendice A

"Usando bases de datos Microsoft Access".

A.1 "Usando Bases de Datos Microsoft Access"

Este apéndice discute el uso de Visual Basic para manipular bases de datos Microsoft Access. Cuando uses Microsoft Access para crear una base de datos, puedes usar cualquiera de las versiones 1.0 o 1.1 de Microsoft Access. Visual Basic y Microsoft Access comparten librerías (**dll's**). A causa de esto, el formato de la base de datos, el lenguaje de definición de datos (**ddl**) y el lenguaje de manipulación (**dml**) son virtualmente idénticos. Sin embargo, Visual Basic y Microsoft Access implementan ciertas características diferentes, como se muestra en la siguiente tabla.

Componente	Microsoft Access	Visual Basic
Gráficos	Usa gráficos OLE. No puede manipular gráficos tipo objeto salvados con Visual Basic, pinturas o controles de imágenes.	Usa gráficos binarios. No puede leer gráficos tipo objeto de OLE Microsoft Access, pinturas o controles de imágenes.
Seguridad	Permite tener asignados datos tipo objeto.	Puede acceder a bases de datos protegidas, pero no puede cambiar la propiedad de seguridad.
Parentesco	Puede asignar la integridad referencial relacionada a los datos tipo objeto. Permite tener relaciones cuando agregas un dato, actualizas, o borras.	Las aplicaciones Visual Basic no pueden asignar o cambiar la relación para que las operaciones de los datos manipulados sean sujetos a algunos requerimientos de relaciones entre ellos.
Funciones definidas por el usuario	Puede agregar funciones definidas por el usuario dentro de preguntas (queries).	No puede soportar agregar funciones definidas por el usuario en preguntas (queries).

A.2 Funciones unidas a preguntas

Cuando uses una base de datos que fue creada por Microsoft Access la base de datos puede contener preguntas (queries), que tienen funciones definidas por el usuario. Por ejemplo, esto es posible al crear una función en Microsoft Access como esta :

SELECT * **FROM** Countries **WHERE** Countries.**Name** = MyFuntion ( )

Este tipo de pregunta no es soportada cuando la base de datos ha sido abierta con Visual Basic. Tu podrás abrir y usar la base de datos, pero no podrás abrir o ejecutar una pregunta. Preguntas que contienen funciones que están construidas dentro de Visual Basic son soportadas y tu aplicación será ejecutada normalmente.

A.3 Entendiendo la seguridad en el acceso y la base de datos System.mda

Para mantener la integridad del sistema de Microsoft Access, incluye la habilidad de acceso a bases de datos protegidas. Microsoft Access crea una base de datos System.mda cuando es instalado por primera vez. El archivo System.mda es usado para manejar nombres de usuarios y grupos de permisos.

Nota : El nombre de la base de datos usado por la base de datos Visual Basic puede ser ajustado cambiando parámetros en el archivo **VB.INI** en la sección **[Options]**. Para ajustar un nuevo nombre, cambia el parámetro SystemDB.

Por default, el archivo System.mda proporciona administrar y acceder a todas las bases de datos sin password (" "). Una vez creado, el archivo System.mda no será movido, alterado, abierto, reemplazado o cambiado de ninguna forma.

La base de datos Visual Basic no requiere el archivo System.mda a menos que esperes manipular bases de datos. Visual Basic no puede romper el sistema de seguridad Microsoft Access. En otras palabras, si un usuario no tiene permiso para leer una tabla en una base de datos con formato Microsoft Access, no podrá leer la tabla ni usando Visual Basic.

Solo Microsoft Access puede implementar o modificar los ajustes de seguridad de una base de datos con formato Microsoft Access. Ver la documentación de Microsoft Access para mayor información sobre el uso del login, passwords y ajustes de seguridad.

Si la seguridad para la base de datos con formato Microsoft Access ha sido activada, necesitarás ejecutar la sentencia SetdefaultWorskpace para que Visual Basic ajuste y inicie tu nombre de usuario (User) ID y password. En este caso, tu aplicación necesitará el archivo System.mda asociado con la base de datos Microsoft Access que quieras manipular. También puedes desear ajustar el **DB-OPTIONINIPATH** con la sentencia SetDataAccessOption para decir a Visual donde encontrar esta base de datos System.mda.

En casos donde necesites ambos, la base de datos Microsoft Access y el archivo System.mda, asegúrate de incluir provisiones para ambos en tu definición de las rutinas de tu aplicación.

Si tu no estas planeado manipular una base de datos con formato Microsoft Access, entonces el sistema de seguridad Microsoft Access no importará. No necesitas un archivo System.mda para usar una base de datos ODBC con SQL server u Oracle, u otras bases de datos ISAM tales como Btrieve, dBase, Microsoft Fox Pro, o Paradox.

A.4 Usando el archivo que soporta la visualización (Locking)

Cuando Visual Basic abre tu archivo base de datos, un archivo de soporte con la extensión .ldb es creado. Este archivo asiste al manejador de el mecanismo de visualización (Locking). Aunque este archivo aparece en el mismo directorio con tu base de datos, este no tiene que ser compactado (backed up ó shipped) con tu base de datos. Este es creado automáticamente siempre que tu base de datos es abierta. No debes usar Move o Delete para el archivo de soporte cuando la base de datos es abierta.

A.5 Compartiendo bases de datos

Si planeas compartir las bases de datos que creaste, o si tu estas compartiendo otras bases de datos, asegúrate que MS-DOS o el sistema operativo de Windows, estén preparados para manejar conflictos que surjan cuando un número de usuarios traten de acceder al mismo tiempo el mismo archivo.

Visual Basic usa Share.exe para proporcionar un mecanismo de visualización de archivos que permite que Windows pueda comunicarse el uno con el otro para clasificar archivos lectura/escritura. Por consiguiente, es importante cargar Share.exe antes de iniciar Microsoft Windows. Sin Share.exe cargado, puedes experimentar descomposición de datos, cuando accedes a bases de datos con Visual Basic.

Usando **Share.exe** con Visual Basic permite a múltiples usuarios acceder a la base de datos al mismo tiempo, mientras cada usuario proteja sus datos al mismo tiempo. Si una base de datos es abierta en modo exclusivo por un usuario, otros usuarios no podrán usar la base de datos hasta que el primer usuario la libere. Sin embargo, si todos los usuarios abren la base de datos en modo compartido (**Share**), **Share.exe** permite a todos los usuarios leer y escribir a la base de datos simultáneamente, con excepción de registros **Locked**. Por ejemplo, si un usuario tiene 100 registros abiertos en una base de datos en particular, aquel registro es considerado locked y otros usuarios no podrán cambiar ningún dato de los que se están ocupados (100 registros), hasta que el primer usuario haya terminado.

Por esta razón, aunque esto es posible correr Visual Basic sin cargar primero **Share.exe**, esto no es recomendado. Si tu escoges no usar **Share.exe** debes asegurarte que solo un usuario escribe a un registro de la base de datos en cualquier momento.

A.6 Usando Vshare.386

La única vez que no debes usar **Share.exe** es cuando corres Microsoft Windows para trabajo en grupo en modo **Enhanced**. Microsoft Windows para trabajo en grupo en modo **Enhanced** carga y usa un archivo que hace un llamado a **Vshare.386**, el cual es un remplazo de **Share.exe**. Hay una entrada para **Vshare.386** en el archivo **System.ini**.

Si planeas correr sobre Windows para trabajo en grupo exclusivamente en modo **Enhanced**, y no correr otras aplicaciones que requieran **Share.exe**, debes salvar aproximadamente 5K de memoria convencional por no cargar **Share.exe**. Para hacer esto, remueve **Share.exe** de tu archivo **Autoexec.bat** y reinicia tu computadora.

Share.exe es probable que este en tu archivo **Autoexec.bat** porque durante el proceso de instalación, el programa **Setup.exe** automáticamente inserta el siguiente comando MS-DOS en tu archivo **Autoexec.bat**.

<MS-DOS Directorio> \ Share.exe / L : 500

Si **Share.exe** esta presente antes de correr Microsoft Windows para trabajo en grupo en modo **Enhanced**, **Vshare.386** esta localizado en lugar de **Share.exe**, temporalmente **Share.exe** no esta habilitado. **Vshare.386** toma más archivos para trabajar hasta que salgas de Microsoft Windows para trabajo en grupos. Después de que sales de Windows para trabajo en grupo, **Vshare.exe** pasa archivos de trabajo a **Share.386** para usar aplicaciones MS-DOS que lo requieran.

Visual Basic corre correctamente con uno o con otro (**Share.exe** o **Vshare.386**), pero **Share.exe** limita el número de llaves disponibles a el número especificado cuando **Share.exe** fue cargado. El parámetro **/L** especifica el número de llaves compartidas; el valor por default es 20. **Vshare.386**, dinámicamente localiza el número de llaves disponibles basado sobre la demanda de llaves.
El número de llaves disponible es especialmente importante si la computadora esta corriendo en Microsoft Windows para trabajo en grupos.

A.7 Compartiendo System.mda

Un archivo en conflicto System.mda es probable que ocurra solo en sistemas con muchos usuarios (más de 100). El siguiente mensaje aparece cuando dos usuarios simultáneamente tratan de actualizar un objeto similar en una base de datos compartida.

Could't update; currently locked by user '<user name>' on machine '<machine name>'

En diseño, Visual Basic no puede simultáneamente actualizar el archivo **System.mda** para múltiples usuarios. En un sistema con un simple usuario (single-user), este error es disparado cuando no fueron especificadas suficientes llaves en el archivo **Autoexec.bat**.

Para resolver este problema en un sistema multi-usuarios, trata de salvar el objeto otra vez. Si no es actualizado un objeto, al mismo tiempo, la operación deberá trabajar. En un sistema con un simple usuario (single-user), usando **Share.exe**, necesitarás incrementar el número de llaves disponibles. Para hacer esto, sigue los siguientes pasos:

1.- Salir de Windows
2.- Cambiarte al directorio de trabajo Visual Basic y borrar todos los archivos ***.ldb**.
3.- Cambiarte al subdirectorio Windows\Temp o al subdirectorio donde tu envias las variables o archivos Temporales (Temp).
4.- Borrar todos los archivos ***.tmp**
5.- Editar el archivo **Autoexec.bat** y ver si existe esta línea.
\DOS\Share.exe.
6.- Cambiar esta línea a :
\DOS\Share.exe / L : 500.
7.- No podrás encontrar esta línea si no ha sido instalado Vshare.exe o si estas corriendo en una

Apendice B

"Microsoft Access SQL".

B.1 "Microsoft Access SQL"

El SQL ó structured query language (Lenguaje estructurado de preguntas) es usado para preguntar, actualizar y manejar bases de datos relacionales, tales como Microsoft Access. Cuando haces una pregunta (query) usando el data control o cuando creas una variable **Recordset**, Visual Basic construye las sentencias necesarias SQL. Porque Visual Basic usa el proceso de la base de datos de Microsoft Access para estas operaciones, programas que incluyen sentencias SQL en sus preguntas (queries), deben conformar los estándares Microsoft Access SQL. Puedes requerir que las sentencias SQL sean pasadas a través de Visual Basic a el servidor remoto SQL, en este caso debes conformar la sintaxis SQL esperada por el servidor remoto.

Puedes usar sentencias SQL en muchos lugares en Visual Basic donde puedes ingresar el nombre de una tabla, pregunta (query) o campo. Por ejemplo, puedes ingresar una sentencia SQL para ajustar la propiedad **RecordSource** de un Data Control, para desplegar una lista de artículos, etc. . Puedes también usar una sentencia SQL cuando creas un **Dynaset**, **Snapshot** o **QueryDef**.

El formato general de una sentencia SQL incluye tres partes:

[parámetro-declaración;] manipulación - sentencia [opcion - declaración];

Las tablas siguientes proporcionan información sobre la sintaxis de estos elementos:

Para mayor información acerca de como Visual Basic y Microsoft Access implementan SQL, buscar Help para SQL. Ver también la guía de referencia rápida para SQL, para otras fuentes de información acerca de SQL.

B.2 Diferencias entre Microsoft Access SQL y Ansi SQL

Microsoft Access SQL es derivado de estándares Ansi (Ansi-86, Ansi-89 y Ansi-92) y más extensiones. Las tablas en este apéndice, listan las palabras reservadas y características que son directamente soportadas por Microsoft Access SQL. Ciertas características Ansi SQL no están implementadas en Microsoft Access SQL. Contrariamente, Microsoft Access incluye palabras reservadas y características no implementadas en Ansi SQL.

B.3 Mayores Diferencias

- Las dos versiones tienen diferentes tipos de datos. Las tablas lista los tipos de datos equivalentes entre Microsoft Access SQL y Ansi SQL.
- Diferentes reglas se aplican para las palabras reservadas **BETWEEN...AND**, las cuales tienen la siguiente sintaxis.

expresión1 **[NOT] BETWEEN** expresión2 **AND** expresión3

En **Microsoft Access SQL**, la expresión2 puede ser mas grande que la expresión3. En **Ansi SQL**, la expresión2 puede ser igual o menor que la expresión3.

- Diferentes caracteres son usados con la palabra reservada **LIKE**:

Caracteres semejantes	Microsoft Access SQL	Ansi SQL
Algún carácter simple	?	_ (un derscore)
Cero o más caracteres	*	%

• Microsoft Access SQL es generalmente menos restrictivo. Por ejemplo, te permite agrupar y ordenar expresiones.
• Microsoft Access SQL soporta expresiones más poderosas. Por ejemplo, puedes incluir cualquier acceso básico ó definición y uso de una función en una sentencia Microsoft Access SQL.

B.4 Características mas importantes de Microsoft Access SQL

• Unir (interior y exterior).
• La sentencia **TRANSFORM**, la cual proporciona soporte para preguntas cruzadas.
• Agrega funciones adicionales, tales como **StDev** y **VarP**.
• La palabra reservada **PARAMETERS** para definir preguntas (queries) parametrizadas.

B.5 Características Ansi SQL no soportadas en Microsoft Access SQL

• Subpreguntas (subqueries); en lugar de ello, puedes enlazar preguntas múltiples.
• En lugar de sentencias **ddl**, tales como **ALTER**, **CREATE** y **DROP**, puedes usar la interface de usuario.
• La palabra reservada **UNION**.
• Agregar distintas funciones.

B.6 Palabras reservadas Microsoft Access SQL

La siguiente tabla lista las palabras reservadas soportadas por Microsoft Access SQL. Por otra parte, Microsoft Access SQL soporta todas las funciones definidas y de acceso básico.

Tabla 1.1 Palabras Reservadas Microsoft Access SQL

PALABRAS	RESERVADAS		
ALL	DISTINCTROW	LEFT	RIGHT
AS	FLOAT	LEVEL	SELECT
ASC	FROM	LONG	SET
BINARY	GROUP	LONGBINARY	SHORT
BIT	HAVING	LONGTEXT	SMALLINT
BY	IEEESINGLE	OPTION	TABLEID
CHAR [ACTER]	IN	ORDER	TEXT
CURRENCY	INNER	OWNERACCESS	TRANSFORM
DATABASE	INSERT	PARAMETERS	UPDATE
DATETIME	INT [EGER]	PIVOT	VALUE
DELETE	INTO	PROCEDURE	WHERE
DESC	JOIN	REAL	WITH
DISTINCT			

B.7 Tipos de datos equivalentes SQL

La siguiente tabla lista los tipos de datos equivalentes Microsoft Access SQL Y Ansi SQL.

Tabla 1.2 Tipos de Datos Equivalente SQL.

Microsoft Access SQL	ANSI SQL
BINARY	No soportado
BYTE	No soportado
CURRENCY	No soportado
DATETIME	TIMESTAMP
DOUBLE	FLOAT
LONG	INT[EGER]
LONGBINARY	No soportado
LONGTEXT	VARCHAR
SHORT	SMALLINT
SINGLE	REAL
TEXT	[CHAR]ACTER]
No soportado	NUMBER
No soportado	NUMERIC
No soportado	DECIMAL
No soportado	DOUBLE PRECISION
No soportado	PICTURE

B.8 Sintaxis Microsoft Access SQL

Generalmente, una sentencia Microsoft Access SQL consta de tres partes: una declaración del parámetro, una sentencia manipuladora y una opción de declaración.

La declaración del parámetro tiene la siguiente sintaxis:
PARAMETERS definición de la lista de parámetros.

Tabla 1.3 muestra la sintaxis de cada elemento.

Elemento	Sintaxis
Definición de la lista de parámetros	Parámetros de definición [, parámetros de definición]...]
Definición de parámetros	Nombre del parámetro Tipo de dato del parámetro
Parámetros de tipos de datos	Nombre del parámetro de acceso al tipo de dato [Nombre del parámetro Ansi del tipo de dato
Parámetros para acceder a los diferentes tipos de datos	**BINARY | BOOLEAN | BYTE | CURRENCY | DATETIME | DOUBLE | LONG | LONGBINARY | LONGTEXT | SHORT | SINGLE | TEXT**
Parámetros ANSI para los diferentes tipos de datos	**CHAR[ACTER] | FLOAT | INT[EGER] | REAL | SMALLINT | VARCHAR**

La sentencia manipuladora tiene la siguiente sintaxis.

delete-declaración | insert-declaración | select-declaración | select-dentro de la declaración | transform-declaración | update-declaración.

Tabla 1.4 Muestra la sintaxis de cada elemento.

Elemento	Sintaxis
borra una declaración	**DELETE FROM** nombre de la tabla [**WHERE** condición buscada]
Inserta una declaración	**INSERT INTO** nombre de la tabla (nombre de la columna listada) [dentro de la declaración] [selección de la declaración]
Selecciona una declaración	**SELECT [ALL | DISTINCT | DISTINCTROW]** Seleccion de la expresión de la tabla a listar
Selecciona dentro de una declaración	**SELECT [ALL | DISTINCT | DISTINCTROW]** selección de la lista **INTO** nombre de la tabla [dentro de la declaración] de la cláusula
Transforma una declaración	**TRANSFORM** referencia de la función selección de la declaración **PIVOT** nombre de la columna
Actualiza una declaración	**UPDATE** referencia de la tabla a listar **SET** envía la cláusula de la lista [**WHERE** condicion a buscar]

La opción de declaración tiene la siguiente sintaxis:

WITH OWNERACCESS OPTION

Apendice C

"Accediendo bases de datos externas".

C.1 "Accediendo a bases de datos externas"

Una de las ventajas del uso de Visual Basic para manejar bases de datos, es la habilidad para dibujar datos de fuentes externas. Por ejemplo, puedes usar información de otras bases de datos ya existentes en el lugar, o puedes conectarte al manejador de bases de datos para otras aplicaciones de manejo de bases de datos. Para usar datos externos, puedes abrir tablas de bases de datos externas directamente, o unir (pegar) tus bases de datos a estas tablas, haciendo la liga a la parte fija de la tabla externa de tu base de datos. Visual Basic puede acceder a cualquiera de las siguientes bases de datos:

- Microsoft Access / Visual Basic.
- Btrieve (con archivos File.ddf y Field.ddf)
- dBase III y dBase IV
- Microsoft FoxPro y Microsoft Foxpro para windows.
- Bases de datos **ODBC**, tales como Microsoft **SQL** server y Oracle
- Paradox

Nota : Visual Basic puede crear, leer y escribir bases de datos Microsoft Access, como también compartir una base de datos común con Microsoft Access. Las referencias a bases de datos Visual Basic en este apéndice, incluyen aquellas bases de datos creadas o manipuladas por Microsoft Access.

Generalmente puede elegir tres caminos cuando usas datos de fuentes externas:

- Puedes dejar los datos en la base de datos fuente y unir la tabla fuente a tu base de datos Visual Basic.
- Puedes acceder directamente a la tabla de la base de datos externa.
- Puedes importar datos dentro de tu base de datos Visual Basic.

Esto tiene sentido al unir una tabla o acceder directamente a la tabla si esta ya es leída de un servidor ODBC, o en una base de datos externa la cual esta siendo actualizada. En este caso el mecanismo usado para actualizar, manejar y compartir datos puede quedarse en el lugar; Pero tu aplicación tendrá problemas involucrados por la recuperación de datos externos.

Puedes usar tablas externas como cualquier otra tabla en tu base de datos Visual Basic, y puedes usarlas mientras están siendo usadas por otra aplicación, compartiendo el mismo servidor. También puedes combinar operaciones que incluyan datos externos de tablas unidas con datos almacenados en la base de datos local.

Si escoges importar datos provenientes de tablas externas dentro de tablas Visual Basic, esto puede ser realizado a través de Microsoft Access o también puedes utilizar una aplicación Visual Basic. Visual Basic es especialmente adecuado para leer archivos ASCII, usando la sentencia **INPUT #**.

Las siguientes secciones discuten como acceder directamente a tablas externas, o como unirlas a una base de datos Visual Basic existente. Si tienes una base de datos Microsoft Access o Visual Basic existente que ya tiene tablas unidas a esta, Visual Basic puede extraer datos de estas tablas unidas sin ningún trabajo de tu parte.

C.2 Tips generales para el uso de tablas externas

Cuando uses una tabla de bases de datos externa, debes mantener en mente los siguientes tips y consideraciones.

C.3 Parámetros de inicialización

Antes de que tu aplicación use cualquier objeto de acceso a datos, necesitarás proporcionar a Visual Basic la localización del archivo que contiene los parámetros de inicialización para cada base de datos externas que esperes usar. Tu puedes hacer esto usando la sentencia **SetDataAccessOption**. Para mayor información, ver el capítulo 3 "Manejo de bases de datos usando Visual Basic".

C.4 Archivos de inicialización

El archivo de inicialización debe contener una sección que incluya información de configuración de bases de datos externas. Ver la sección "Detalles del archivo de inicialización" mas adelante en este apéndice, para listar los archivos **.INI** que tu necesitas conectar para soportar diferentes bases de datos.

Nota : Si tu no tienes las correctas líneas (como las descritas anteriormente) en tu archivo **VB.INI** o **APPNAME.INI**, podría generarse un error **"Cannot find installable ISAM"**.

C.5 Protección de bases de datos Microsoft Access con un Password

Si accedes a una tabla externa de una base de datos Microsoft Access, puedes necesitar proporcionar un password. Puedes hacer esto usando la sentencia **SetDefaultWorkspace**. Por ejemplo, el siguiente código indica que el nombre del usuario es "Chrissy" y el password es "HighIQ".

```
SetDefaultWorkspace "Chrissy", "HighIQ".
```

Para mayor información, ver el capítulo 3 "Manejo de la base de datos usando Visual Basic".

C.6 Password con bases de datos externas

Si accedes a una base de datos externa proveniente de Btrieve o Paradox, o de una base de datos **ODBC**, puedes necesitar suministrar un password. Nota que este password es diferente al password de usuario de Microsoft Access. Este password esta colocado en la base de datos externa. El password de la base de datos esta suministrado en el argumento de la cadena **"Connect"** de la función **OpenDatabase** usando el identificador **PWD**. Por ejemplo, la siguiente cadena **"Connect"** incluye un password:

```
" Paradox; PWD = mypword; "
```

C.7 Encapsulando una base de datos

Si accedes a una tabla externa y especificas un password, Visual Basic puede almacenar este password en la tabla externa ligándolo a la información en tu base de datos o en la cadena **Connect**. Por consiguiente, cualquier usuario de tu base de datos puede abrir la tabla después. Por esta razón, puedes querer encapsular una base de datos que contiene tablas unidas con passwords. Para mayor información sobre como encapsular una base de datos, ver el capítulo 3 "Manejo de una base de datos usando Visual Basic".

C.8 Salvando el passwords y ligándolo a tablas de información

Para salvar el password con el que ligaste la tabla de información, ajusta o pon el bit **DB_ATTACHSAVEPWD** a **True** en la propiedad **TableDef**.

C.9 Accediendo a tablas externas en una red

Para acceder a una tabla externa en una red, debes conectarte a la red y tener acceso al archivo de la base de datos y directorio, o para una base de datos **ODBC** el nombre del servidor. Si quieres que Visual Basic te conecte automáticamente al archivo apropiado del servidor, cada vez que abres una tabla externa, puedes especificar la ruta completa de la red para el archivo en el **Connect** de la propiedad **TableDef**. No hay un mecanismo para proporcionar un password en una red. En estos casos donde un password es requerido para entrar a una red compartida, deberás conectarte mediante un password, y pre-asignarle un drive (letra del drive) en alguna parte antes de iniciar tu programa.

Por ejemplo, si usas Microsoft Lan (Maneger network), debes proporcionar la siguiente ruta para conectarte a un archivo remoto dBase:

```
\\server\share\datadir\author.dbf
```

Para proporcionar la ruta en la unión de tablas de información, tu debes usar el siguiente código:

```
Dim Db As Database
Dim Td As New TableDef
Set Db = OpenDatabase ("Mydb.mdb")
' Abre una base de datos Visual Basic.
' Asume que tu quieres unir una tabla dBase III llamada Author
' en el servidor \\SERVER\SHARE\DATADIR . Tu quieres el nombre
' de el linkeado "dBase Author Table" en tu base de datos .mdb.

Td . Connect = "dBase III; DATABASE = \\SERVER\SHARE\DATADIR"
Td . SourceTableName = "AUTHOR"              ' El nombre de tu archivo .dbf.
Td . Name = "dBase Author Table"             ' Como tu puedes referirte a este.

Db . TableDefs . Append Td                   ' Crea el linkeo.
```

En este punto, la base de datos Visual Basic **Mydb.mdb** tiene la tabla dBase unida a ella. El archivo dBase no es movido, los datos estarán disponibles para tu aplicación. Nota que el código tiene un alias para el archivo dBase, usando el nombre de la propiedad **TableDef**. Generalmente, la sintaxis para acceso a otros tipos de archivos externos es similar a la técnica descrita anteriormente.

C.10 Definiciones de llaves primarias con tablas externas

Cuando defines tablas externas, solo tablas ODBC y Paradox soportaran definiciones de llaves primarias (Primary Keys). Tablas Paradox requieren llaves primarias (Primary Keys). Para Btrieve, dBase y Tablas FoxPro, no puedes definir una llave primaria (Primary Key), pero si puedes ajustar la propiedad **Unique** igual a **True** (verdadero), para indicar que las tablas no deben de contener líneas duplicadas.

C.11 Trabajando con tablas unidas

Puedes usar una tabla unida como si esta fuera una tabla Microsoft Access, hay consideraciones especiales. Para mayor información acerca del trabajo con tablas unidas, ver "Usando tablas externas" mas adelante en este apéndice.

C.12 Trabajando con múltiples tablas de bases de datos externas

Cuando trabajas con tablas múltiples de bases de datos externas, ocasionalmente puedes encontrar que la propiedad **Updatable** esta en **FALSE**. Generalmente, esto es debido a la complejidad de la pregunta (query). Para poder actualizar consistentemente las tablas externas, puede ser mas fácil acceder con simples preguntas (queries).

C.13 Espacio de alojamiento temporal para preguntas

Cuando Visual Basic manipula bases de datos externas, esto crea índices temporales para las preguntas que están siendo ejecutadas en el disco duro de la WorkStation, aun si la base de datos es una base externa (Networked). Espacio temporal es localizado para el directorio indicado por la variable tipo cadena TEMP, la cual usualmente apunta o señala a el directorio **\Windows\Temp**. Debe de preveer que tu sistema no haya establecido una variable Temp, señale la ruta invalida, o que tu Workstation no tiene espacio suficiente para el archivo temporal. El espacio necesario esta en función del tamaño de la tabla unida; esta puede variar de unos pocos miles de bytes a muchos Mbytes.

C.14 Borrando registros de tablas externas

Cuando borras un registro de una tabla externa, el registro desaparece. Sin embargo, cuando borras una tabla unida de la coleccion **TableDefs**, solo la tabla ligada es borrada. La tabla de datos permanece sin cambios.

C.15 Borrando registros de dBase o FoxPro

Cuando borras registros provenientes de bases de datos como dBase o FoxPro, los registros pueden reaparecer cuando la tabla es cerrada o reabierta. Para decirle a Visual Basic que no vaya a buscar registros borrados, ajustar el parametro **Deleted** en el archivo .INI en "on" (por default).

C.16 Abriendo tablas externas

Tu puedes abrir una tabla externa, usando cualquiera de las dos funciones como es el Data Control o el **OpenDatabase**. Generalmente, el método usado por cada una de las bases de datos externas es aproximadamente el mismo. En secciones subsecuentes en este apéndice, se tratan las características individuales de cada uno de los formatos de bases de datos externas.

C.17 Usando el Data Control

Cuando usas el Data Control para abrir directamente tablas externas, necesitas ajustar las propiedades para el Data Control en la ventana propiedades en tiempo de diseño, o escribir el código que haga los ajustes en tiempo de diseño o en tiempo de ejecución (corrida). El siguiente ejemplo muestra como hacerlo.

```
Data1 . Connect = "Paradox 3.x;"          ' Especificando el tipo de la base de datos.
Data1 . DatabaseName = "C:\PARADOX"' Directorio de la base de datos.
Data1 . RecordSource = "ParaTab1"
```

```
Data1 . Refresh                                    ' Abre la base de datos y crea un Dynaset.
While Not Data1 . Recordset . EOF

        Print Data1 . Recordset ( 0 )              ' Para todos los registros.
        Data1 . Recordset . MoveNext
Wend
```

Usando la función OpenDatabase

Hay varios métodos que puedes usar para abrir una tabla externa directamente con código. En el siguiente ejemplo, el argumento **Connect** para la función **OpenDatabase** es usado solo para contener el tipo de base de datos ("Paradox;" en este caso). La cadena **Connet** puede también ser usada para proporcionar nombre a la base de datos, password (si este es requerido), u otros parámetros necesarios. Cualquier técnica es aceptable, cuando usas **OpenDatabase**, Visual Basic espera encontrar un nombre de directorio (no un nombre de archivo), como primer argumento de la función. El ejemplo abre una base de datos Paradox, el argumento **Connect** esta ajustado para "Paradox;".

Este ejemplo también usa la función **OpenTable**. Nota que el nombre de la tabla especificada en la función **OpenTable** especifica el nombre del archivo Paradox (PARATABL) sin la extensión .db. Puedes usar una de las otras funciones **Recordset** creadas en lugar de **OpenTable**.

El siguiente código muestra un ejemplo de como abrir una tabla directamente. (Si estas trabajando con otro tipo de tabla externa, nota la diferencia para tu tipo de base de datos y realiza los cambios apropiados a tu código).

```
Dim db As Database
Dim tb As Table
Dim conn As String
conn$ = "Paradox;"              ' Especifica el tipo de base de datos, de la base de datos abierta.

Set db = OpenDatabase ("C:\PARADOX", False, False, conn$)
Set tb = db . OpenTable ("ParaTab1")      ' Abre la tabla de Paradox.
While Not tb . EOF
        Print  tb (0)
        tb . MoveNext
When
```

Accediendo a tablas Paradox

Visual Basic puede acceder a tablas externas de Paradox 3.0 y 3.5. Si proporcionas un password correcto, Visual Basic puede abrir tablas encapsuladas Paradox. Si abres una tabla externa Paradox o unes esta a tu base de datos Visual Basic, puedes extraer y actualizar datos, a un si otro usuario esta usando los datos en Paradox. Para mayor información ver la sección "Uniendo Tablas (Un ejemplo)", mas adelante en este apéndice.

Cuando accedes a tablas Paradox, necesitarás especificar el nombre del directorio (no el nombre de un archivo), con o la base de datos en la propiedad **Connect** y el nombre del archivo de la tabla en la propiedad **SourceTable Name**. Por ejemplo, para unir un archivo Paradox llamado Author.db y usa el nombre "Paradox Au hor" para referenciarlo, podrías usar los cambios hechos en el siguiente código a la propiedad **TableDef**.

```
Td . Connect = "Paradox; DATABASE\PARADOX\PUBS;"
Td . SourceTableName = "Author"
Td . Name = "Paradox Author"
```

Cuando abres tablas de base de datos Paradox directamente, podrás necesitar también especificar el nombre del directorio (no un nombre de archivo) a el **DatabaseName** en la función **OpenDatabase** y el nombre de la Tabla, en el argumento del método **OpenTable**.

Por ejemplo, para abrir un archivo Paradox Author.db y usar el nombre "ParaAuthor" para referenciar esto como una tabla tipo objeto, usa el siguiente código:

```
Dim db As Database
Dim ParaAuthor As Table
Dim conn As String
conn$ = "Paradox;"
Set db = OpenDatabase ("C:\PARADOX", False, False, conn$)
Set ParaAuthor = db . OpenTable ("Author")

While Not ParaAuthor . EOF
        Print ParaAuthor ( 0 )
        ParaAuthor . MoveNext
Wend
```

Llaves Primarias e Indices en Paradox

Paradox almacena información importante acerca de la llave primaria de la tabla en un índice (index), especificando en un archivo **.px**. Si entras a una tabla Paradox que tiene una llave primaria (primary keys), Visual Basic necesita el archivo **.px** para abrir la tabla externa. Si borras o mueves este archivo, no podrás abrir la tabla externa.

Cuando creas índices Paradox, el primer índice debe ser marcado como primario. El primer índice debe ser definido sobre la primer columna en la tabla sobre la cual los registros grabados serán ordenados. Todos los demás índices pueden ser definidos solo en una columna sencilla. Solo índices ascendentes son soportados por Paradox. No debes crear índices secundarios porque estos no son soportados.

Si unes una tabla Paradox que no tiene una llave primaria (primary key), no puedes actualizar datos en la tabla usando Visual Basic. Para poder utilizar la tabla, define una llave primaria (primary key) en Paradox.

Conversiones de tipos de datos Paradox a Microsoft Access

Cuando accedes a una tabla externa Paradox, Visual Basic traduce los tipos de datos Paradox dentro de los correspondientes tipos de datos Visual Basic. La siguiente tabla lista las conversiones de tipos de datos.

Tipos de datos Paradox	Tipos de datos Visual Basic
Alphanumeric	Text
Currency	Number (La propiedad del tamaño del campo es enviado como Double)
Date	Date / Time
Number	Number (La propiedad del tamaño del campo es enviado como Double)
Short number	Number (La propiedad del tamaño del campo es enviado como Double)

Acceso a Archivos dBase y FoxPro

Visual Basic puede abrir directamente archivos externos **.dbf** en dBase III, dBase IV o Microsoft FoxPro versión 2.0 o 2.5, o unir los archivos a tu base de datos. Si puedes abrir directamente un archivo dBase o FoxPro o unirlo a tu base de datos Visual Basic, puedes ver y actualizar los datos, aun si otros usuarios están usándolos con dBase o FoxPro.

Si tu accedes a un archivo dBase o FoxPro, también puedes decir a Visual Basic que use uno o mas archivos indices (**.ndx** o **.mdx** para dBase; **.idx** o **.cdx** para FoxPro) para mejorar el desempeño.

Para las bases de datos dBase y FoxPro, Visual Basic lleva la cuenta de los índices en un archivo de información especial (**.inf**). Cuando Usas Visual Basic para actualizar los datos en tu archivo **.dbf**, Visual Basic también actualiza los archivos índice para reflejar los cambios. El archivo **.inf** es creado por ti cuando usas Visual Basic, para crear un nuevo índice para una tabla dBase o una tabla FoxPro, o puedes crearlos por ti mismo con un editor de textos. El formato para los archivos **.inf** es el siguiente :

El archivo **TableName.inf** contiene:

NDX1 = < Index 1 Filename > **.NDX**
NDX2 = < Index 2 Filename > **.NDX**
NDX3 = < Index 3 Filename > **.NDX**

Por ejemplo, un archivo **.inf** para la tabla Authors que será nombrada Authors.inf, contiene las líneas siguientes:

NDX1 = CityIndx **.NDX**
NDX2 = NameIndx.**NDX**

Si estas usando dBase III, debes poner este índice y el archivo **.inf** en el mismo directorio que están los otros archivos dBase III.

Las bases de datos dBase y FoxPro no están mantenidas en un solo archivo, pero si en un mismo directorio, que tiene datos separados, índice, y otros archivos de soporte. Cuando accedes a este tipo de tablas de bases de datos, necesitarás especificar el nombre del directorio, así como el nombre de la base de datos en la propiedad **Connect** y el nombre de la tabla en la propiedad **SourceTableName**. Por ejemplo, para acceder a una tabla dBase IV "Author" (almacenada en Author.dbf) y usar el nombre "dBASE Author" para referirlo, usa la siguiente propiedad **TableDef**.

Td . **Connect** = "dBase IV; DATABASE = C:\DBASE\PUBS\;"
Td . **SourceTableName** = "Author"
Td . **Name** = "dBASE Author"

Cuando abres tablas directamente de bases de datos externas dBase y FoxPro, necesitarás también especificar el nombre del directorio (no nombre de archivo), como el **DatabaseName** en la función **OpenDatabase** y el nombre de la tabla en el argumento **Tablename** del método **OpenTable**.

Por ejemplo, para abrir el archivo Author.dbf de FoxPro versión 2.5 y usar el nombre "FoxAuthor" para definirlo como una tabla objeto, usa el siguiente código:

```
Dim db As Database
Dim FoxAuthors As Table
Dim conn As String
conn$ = "FoxPro 2.5;"
Set db = OpenDatabase ("C:\FOXPRO", False, False, conn$)
Set FoxAuthor = db.OpenTable ( "Author")
While Not FoxAuthors.EOF
        Print FoxAuthor ( 0 )
        FoxAuthor . MoveNext
Wend
```

Los campos dBase y FoxPro grabados están localizados en archivos separados. Estos archivos no pueden ser localizados o movidos fuera del directorio que contienen los archivos de tablas (tables).

Los sistemas de las bases de datos dBase y FoxPro físicamente no borran registros, pero si los marca para borrarlos después. Debes compactar el archivo **.dbf** para remover aquellos registros de los archivos **.dbf**. La función **CompactDatabase** no afecta a las tablas unidas.

Si usas el archivo **.INI**, ajustando la parte **DELETED=ON** (en la sección **[dBASE ISAM]**), Visual Basic filtra los registros borrados, así que no deben aparecer en los **Recordsets**.
Con **DELETED=OFF**, todos los registros están incluidos en los **Recordsets** que tu creaste, incluyendo registros borrados. Esto permite a los usuarios de dBase y FoxPro no borrar registros. En este caso, cuando accedes a una tabla dBase o FoxPro, Visual Basic crea un **Dynaset** para el registro grabado y lo selecciona. Cuando tu borras un registro Visual Basic, deja de seleccionarlo en el **Dynaset** y marca el registro como borrado en el archivo **.dbf**. Si actualizas el **Dynaset** o reabres la tabla, los registros no estarán presentes.

Indices en dBase y FoxPro

Si entras a un archivo **.dbf** y este une a un archivo índice (**.ndx** ó **.mdx** para dBase ó **.idx** o **.cdx** para Fox Pro), Visual Basic necesita el archivo índice (index) para abrir la tabla externa. Si borras o mueves archivos índice o el archivo de información (.inf) no podrás abrir la tabla externa.

Por otra parte, si usas dBase o FoxPro para actualizar datos en un archivo **.dbf** que estas accediendo de la base de datos de Visual Basic, debes también actualizar cualquier índice dBase o FoxPro asociado con el archivo **.dbf**. Si los archivos índices no están al corriente cuando Visual Basic trata de usarlos, los resultados de tus preguntas (queries) son impredecibles.

Conversiones de datos tipo dBase y FoxPro a Visual Basic

Cuando accedes a un archivo dBase o FoxPro, Visual Basic traduce los tipos de datos dBase y FoxPro a los correspondientes datos tipo Visual Basic. La siguiente tabla lista las conversiones de tipos de datos.

Tipos de datos dBase o FoxPro	Tipo de dato Visual Basic
Character	Text
Date	Date / Time
General (solo FoxPro)	OLE
Logical	Yes / No
Memo	Memo
Numeric, Float	Number (el tamaño de los campos es Double)

Acceso a Tablas Btrieve

Usando Visual Basic, puedes abrir directamente o unir tablas en formato Btrieve 5.1. Para usar tablas Btrieve, debes tener los archivos de datos File.ddf y Field.ddf, los cuales dicen a Visual Basic la estructura de tus tablas. Estos archivos son creados por Xtrieve o por otro programa que te genere archivos .ddf. Si borras o mueves estos archivos o tus archivos de datos, no podrás abrir una tabla externa Btrieve.

Para mayor información sobre el uso de Btrieve con Microsoft Access, ver el archivo texto Btrieve.txt en tu directorio Visual Basic.

Cuando accedes a tablas de bases de datos Btrieve, necesitarás el nombre del archivo Btrieve (**.ddf**), así como el nombre de la base de datos en la propiedad **Connect** y el nombre de la tabla en la propiedad **SorceTableName**. En este caso el archivo Btrieve puede no tener relación sobre el nombre de la tabla. El nombre correcto del archivo esta almacenado en el archivo File.ddf.

Por ejemplo, para unir una tabla Btrieve "Author" y usar el nombre "Btrieve Author Table" para referirlo, puedes usar la siguiente propiedad **TableDef** (Para un ejemplo completo de este código, ver la sección "Uniendo una Tabla (Un ejemplo)", más adelante en este apéndice).

```
Dim Db As Database
Dim Td As New TableDef
Set Db = OpenDatabase ("Mydb.mdb")
' Abriendo una base de datos Visual Basic
'Asume que tu quieres agregar una tabla Btrieve llamada Author
' localizada en el directorio C:\BTRIEVE.
Td . Connect = "BTRIEVE; DATABASE = C:\BTRIEVE\File.ddf"
Td . SourceTableName = "Author"
Td . Name = "Btrieve Author Table"
Db . TableDefs . Append  Td                    Creando el linkeamiento.
```

Cuando abres una tabla de una base de datos externa Btrieve directamente, necesitarás especificar el nombre del archivo de datos Btrieve (**.ddf**), así como el del **DatabaseName** en la función **OpenDatabase** y el nombre de la tabla Btrieve en el argumento **Tablename** del método **OpenTable**.

Por ejemplo, para abrir un archivo Btrieve (File.ddf) y usar el nombre "BAuthor" para referir la tabla "Author" de la base de datos Btrieve, usa el siguiente código:

```
Dim db As Database
Dim BAuthor As Table
Dim conn As String

conn$ = "Btrieve;"                ' Especifica la base de datos que se quiere abrir

Set db = OpenDatabase ("C:\BTRIEVE\File.ddf",False, False, conn$)
Set BAuthor = db.OpenTable ("Author")          ' Abre la tabla objeto
While Not BAuthor.EOF
        Print BAuthor ( 0 )                    ' Imprime todos los registros grabados
        BAuthor.MoveNext
Wend
```

Ajustes para el archivo de Inicialización Win.ini

El manejador Btrieve usa la sección **[Btrieve]** del archivo **Win.ini** (no **VB.INI**), cuando este accede a un archivo Btrieve. Después de que instalas Visual Basic y especificas que quieres acceder a archivos Btrieve, el archivo **Win.ini** contiene los siguientes ajustes por default.

[BTRIEVE]

Options= /m:64 /p:4096 /b:16 /f:20 /l:40 /n:12 /t:c:/ \VB3\BTRIEVE.TRN

La siguiente tabla te proporciona una breve descripción de cada parámetro. Debes consultar tu información Btrieve para una mayor lista de ajustes. Si instalas otra aplicación que modifique estos ajustes de valores mostrados, Visual Basic quizá no trabajará correctamente con tus archivos Btrieve.

Switch	Definición
/m	Tamaño de memoria
/p	Tamaño de la pagina
/b	Tamaño de buffer para la pre-imagen
/f	Archivos abiertos
/l	Múltiples Locks
/n	Archivos en una transacción
/t	Nombre del archivo de transacción

Nota : Para usar datos Btrieve, debes tener la librería **"WBTRCALL.DLL"**, la cual no esta proporcionada con Visual Basic. Este archivo esta disponible con Novell, Btrieve para Windows, Novell NetWare SQL, y otros productos basados en Windows que usen Btrieve.

Si esperas compartir una base de datos Btrieve, necesitarás asegurarte que la ruta dada para el archivo de transacción este visible en la red, para todos los usuarios de la base de datos. Generalmente, este archivo se encuentra en un servidor común al que todos los usuarios pueden acceder.

Conversiones de tipos de datos Btrieve a Visual Basic

Cuando tu accedes a una tabla Btrieve, los datos tipo Btrieve son traducidos a los correspondientes datos tipo Visual Basic. La siguiente tabla lista las conversiones de tipos de datos.

Tipo de datos Btrieve	Tipo de datos Visual Basic
Date, time	**Date/Time**
Float o bfloat (4-byte)	**Number** (El tamaño del campo es enviado como **Single**)
Float o bfloat (8-byte, decimal, numeric	**Number** (El tamaño del campo es enviado como **Double**)
Integer (1-,2-, o 4-byte)	**Number** (El tamaño del campo es enviado como **Byte, Integer, o Long Integer**)
Logical	Yes/No
Lvar	Objeto **OLE**
Money	**Currency**
Note	**Memo**
String, lstri g, zstring	**Text**

Uniendo una tabla (Un ejemplo)

El siguiente ejemplo ilustra como unir una tabla a una base de datos con formato Visual Basic. La única diferencia entre los diferentes tipos de bases de datos externas, es el valor de el **Connect** y las propiedades **SourceTableName**.

```
Dim Db As Database
Dim Td As New TableDef
Set Db = OpenDatabase ("C:\MYDB\Pubs.mdb")
' Abriendo una base de datos Visual Basic.
' Se asume que quieres agregar una tabla Paradox llamada Authors (con el
' password mypword) en la base de datos C:\PDX\PUBS.
Td . Connect = "PARADOX 3.X; DATABASE = C:\PDX\PUBS\; PWD = MYPWORD;"
Td . SourceTableName = "Author"
Td . Name = "PDX_Author"
Td . Attributes = DB_ATTACHEXCLUSIVE
Td . Attributes = DB_ATTACHSAVEPWD
' Salvar la cadena Connect.
' Tu puedes configurar el TableDef, y agregarla a la colección.
Db . TableDefs . Append Td                    ' Crear el linkeo.
```

En este punto la base de datos Visual Basic Pubs.mdb se le agregó una tabla Paradox. Se le agregó un password y el archivo Paradox fue puesto en el directorio correcto.
Si tu quieres mover la base de datos Paradox del directorio, tu necesitas especificar la localización correcta de este archivo de bases de datos. El ejemplo siguiente ilustra esto.

```
' El archivo Paradox fué movido a N:\
Db . TableDefs . Delete "PDX_Author"    ' Borrando el linkeo existente.
Set Td = New TableDef

Td . SourceTableName = Db . TableDefs ("PDX_Author") .SourceTableName
Td . Name = d . TableDefs ("PDX_Authors") . Name
Td . Connect = "Paradox 3.X; DSN= N:\; PWD=MYPWORD;"

' Tu puedes configurar el TableDef, y agregarlo a la colección.
Db . TableDefs . Append Td                    ' Creando el nuevo linkeamiento.
```

Uniendo tablas de bases de datos Microsoft Access

Puedes ver y editar datos en otras bases de datos Microsoft Access para acceder sus tablas. Por ejemplo, si tu quieres usar una tabla de otra base de datos almacenada sobre una red. Puedes entrar a tablas de otras bases de datos y usarlas como si estuvieran en la base de datos abierta.

Usando Tablas Externas

Despues de que unes una tabla de otra base de datos, puedes usarla como cualquier otra tabla Microsoft Access. Visual Basic almacena toda la información suministrada cuando unes la tabla, así puede encontrar los datos externos siempre que los necesites. Cuando abres una tabla externa, Visual Basic abre el archivo de la base de datos apropiado o conectado y recupera los datos. Usando una pregunta (query), puedes unir los datos en la tabla externa, con datos de tu tabla Visual Basic. Aun cuando los datos residan en computadoras separadas, Visual Basic combina los datos para responder a tu pregunta (query).

Usando nombres Alias para tablas unidas

Puedes querer usar nombres alternados para las tablas externas que accedes desde Visual Basic. Las base de datos Visual Basic pueden tener nombres de tablas que pueden contener hasta 64 caracteres, y pueden contener espacios, porque puedes querer dar a una tabla un nombre más descriptivo. Por ejemplo, si unes una tabla dBase llamada SLSDATA, podrías renombrarla como sigue "Sales Data 1991 (de dBase)". Para hacer esto, usa la propiedad **SourceTableName** cuando tu creas el linkeamiento para tu tabla externa.

Optimo desempeño con tablas externas

Puedes usar tablas externas como si fueran tablas regulares Microsoft Access. Cada vez que vez un dato en una tabla externa, Visual Basic tiene que recobrar registros de otro archivo. Esto puede tomar tiempo, especialmente si la tabla externa esta en una red o en una base de datos ODBC.

Si estas usando una tabla externa en una red o en una base de datos ODBC, sigue las siguientes recomendaciones para mejores resultados:

- Examina solo los datos que necesites. Evita saltar al último registro en una tabla larga, a menos que quieras agregar nuevos registros.

- Usa preguntas (queries), para limitar el número de registros que recuperas. De esta manera, Visual Basic puede transferir menos datos sobre la red.

- En preguntas que involucran tablas externas, evita usar funciones en preguntas. En particular, evita usar funciones agregadas, tales como **DSum**, en tus preguntas. Cuando usas una función agregada, Visual Basic recobra todos los datos en la tabla externa en orden para ejecutar la pregunta.

- Si agregas muchos registros a una tabla externa, agrega los registros a la tabla de la base de datos Visual Basic y usa una pregunta para unir los registros agregados en una operación. Esto ahorra tiempo, porque Visual Basic no tendrá que recuperar todos los registros en una tabla externa.

- Recuerda que otros usuarios tratarán de usar una tabla externa al mismo tiempo que tu. Cuando una base de datos Visual Basic esta sobre una red, debes evitar cerrar registros largos.

Nota que si cambias la información almacenada en las propiedades **Link**, de una tabla unida (por ejemplo, el archivo base de datos es movido o el pasword cambiado), no podrás abrir la tabla externa. Para especificar información, borra el linkeado existente y une la tabla otra vez.

Nota : Es más rápido acceder a tablas unidas ODBC que acceder directamente.

Detaĥes del archivo de Inicialización

Cuando instalas Visual Basic, puedes instalar muchos de los manejadores de las bases de datos externas. Para estos manejadores que están instalados, hay un archivo asociado con el archivo **.INI**. Abajo se muestran los ajustes por default para los manejadores de bases de datos externas.

Nota : Para determinar el número que deben tener los "Locks", Visual Basic usa la siguiente fórmula: **Count = LockRetry * CommitLockRetry**

Ajustes por default del archivo VB.INI ó APPNAME.INI

```
[Options]
SystemDB = C:\MYPATH\SYSTEM.MDA

[ISAM]
PageTimeout = 5
MaxBufferSize = 128
LockRetry = 20
CommitLockRetry = 20
ReadAheadPages = 16
[Installable ISAMs]

Paradox 3.X = C:\VB\pdx110.DLL
FoxPro 2.0 = C:\VB\xbs110.DLL
FoxPro 2.5 = C:\VB\xbs110.DLL
dBase III = C:\VB\xbs110.DLL
dBase IV = C:\VB\xbs110.DLL

Btrieve = C:\VB\btrv110.DLL
[Paradox ISAM]
PageTimeout = 600
ParadoxUserName = Joe User
ParadoxNetPath = P:\PDOXDB\
CollatingSequence = Ascii
[Btrieve ISAM]
PageTimeout = 600
[dBase ISAM]
PageTimeout = 600
CollatingSequence = Ascii
Century = Off
Date = American
Mark = 47
Deleted = ON                ; Muestra y opera sobre registros grabados
                            ; Deleted=ON : No opera sobre registros grabados
```

Win.ini

La siguiente línea debe de aparece en **Win.ini** (localizado en tu directorio \Windows) si tu intenias usar tablas externas Btrieve. Los detalles de este registro son discutidos en la sección "Accediendo a tablas Btrieve", mas adelante en este apéndice.

[Btrieve]

Options = /m:64 /P:4096 /b:16 /f:20 /1:40 /n:12 /t: c :\VB\BTRIEVE.TRN

Acceso a fuentes de datos ODBC

Usando Visual Basic y abriendo bases de datos conectadas al manejador ODBC, puedes unir tu base de datos a tablas, en una base de datos ODBC, tales como Microsoft **SQL** Server o Oracle. También puedes crear objetos **Dynaset** o **Snapshot** para usarlos en la base de datos, pero no puedes usar la sentencia **OpenTable** directamente para abrir tablas ODBC.

Para editar una tabla externa ODBC, la tabla debe contener un índice único. También puedes acceder a bases de datos ODBC **SQL** view. **SQL** views son siempre de sólo lectura en Visual Basic.

Si estas accediendo a una tabla ODBC, puedes escoger, ya sea que quieras Visual Basic como prompt para tu login ID y password, cada vez que te conectes a la base de datos ODBC conteniendo a la tabla. Si quieres que Visual Basic almacene la información de la conexión en tu base de datos, para que no tengas que escribirla cada vez que incluyas el valor en la cadena **Connect** , especificar **DB_ATTACHSAVEPWD** en la propiedad **TableDefs**.

Nota : Antes de que te conectes a una base de datos ODBC, debes instalar el manejador ODBC apropiado. Puedes instalar un manejador para una base de datos ODBC corriendo el programa Setup de Visual Basic.

Importante : Si te tropiezas con un error mientras accedes a una tabla ODBC, quizá se deba a un problema con la base de datos ODBC Server, en el servidor de la red, o con la misma base de datos. Si no puedes acceder a una tabla ODBC, contacta con la persona que administra la base de datos ODBC. Si el nombre de la tabla o del campo cambia después de que las unas, o si cambias tu password en el servidor de la base de datos, debes borrar tu linkeamiento de la tabla y entonces reunirlas o cambiar el valor aceptado en la cadena **Connect** del **OpenDatabase**.

El siguiente código ilustra un ejemplo de como conectar una tabla a Microsoft SQL Server "Authors" en la base de datos PUBS. Nota que el login ID y password están incluidos en la propiedad **Connect**. Para obtener el manejador ODBC para proporcionar un dialogo para el password, o uno de los otros parámetros, remueve el parámetro (**PWD** o **UID**) de la propiedad **Connect**. Cuando te conectas a un servidor Oracle, necesitarás seleccionar un nombre de una fuente de datos Oracle.

```
Dim Db As Database
Dim Td As New TableDef
Set Db = OpenDatabase ("Mydb.mdb")
' Abriendo una base de datos Visual Basic
'Asume que tu quieres unir una tabla ODBC llamada Pubs.Authors
Td . Connect = "ODBC; UID = SA; PWD = MYPWORD; DSN = WORK; DATABASE = PUBS"
Td . SourceTableName = "Authors"
Td . Name = "ODBC Authors Table"
Td . Attributes = DB_ATTACHSAVEPWD
' Salvando el password en el linkeamiento.
Db . TableDefs . Append Td                    ' Creando el linkeamiento.
```

El siguiente código ilustra como crear un **Dynaset** de una tabla "Authors" en la base de datos Pubs, en un servidor Microsoft SQL. Recuerda, que no puedes usar la función **OpenTable** para una base de datos ODBC. Otra vez, nota que el login ID y el password están incluidos en la propiedad **Connect**. Cuando te conectas a un servidor Oracle, necesitarás seleccionar un nombre de una fuente de datos Oracle.

```
' Asumiendo que tu quieres abrir una tabla ODBC llamada Pubs.Authors
' en el Microsoft SQL server
Dim Db As Database
Dim Ds Dynaset
Dim Conn As String

Conn$ = "ODBC; UID = Marilyn; PWD = 25Years; DSN = Work; DATABASE = PUBS"
' Abriendo la base de datos ODBC
Set Db = OpenDatabase ("", False, False, Conn$)
Set Ds = Db . CreateDynaset ("Authors")
```

```
While Not Ds.EOF
        Print Ds ( 0 ), Ds ( 1 )
        Ds . MoveNext
Wend
```

Usando RegisterDatabase

Cuando trabajas con bases de datos ODBC, Visual Basic espera que le proporciones un nombre de una fuente de datos, que identifique la base de datos externa, localización, manejador, opciones e información del login por default. Usa el nombre de la fuente de datos para especificar el manejador ODBC que tu quieres usar con tu código de la función **OpenDatabase**, o enviando el **DatabaseName** y propiedades de el **Connect** a el Data Control. Generalmente, el nombre de la fuente de datos es reconocida como el **DatabaseName** y ningún parámetro adicional proporcionado en el archivo **ODBC.INI** son proporcionados en la propiedad **Connect** del Data Control o argumento **Connect** de la funcion **OpenDatabase**.

La sentencia **RegisterDatabase** es un camino para agregar o modificar el nombre de la fuente de datos existentes en el archivo **ODBC.INI**, así que tu aplicación puede usar el nombre de la fuente de datos especificada en esta sentencia para acceder una base de datos ODBC. También puedes usar el panel de control ODBC para crear nuevos nombres de fuentes de datos o modificar los existentes. La sintaxis para la sentencia **RegisterDatabase** es como sigue:

RegisterDatabase dsn,driver,silent, atributos.

La sentencia **RegisterDatabase** usa los siguientes argumentos.

Argumento	Descripción
Dsn	Expresa la fuente de datos, frecuentemente es usado el nombre del servidor que se esta usando.
Driver	Expresa el nombre del driver ODBC usado
Silent	Se especifica un valor Boolean. Si es True no visualizaran el dialogo del driver, y si es False tu visualizaras el dialogo del driver que estas usando.
Atributos	Es una cadena donde se especifica los cambios agregados en el ODBC.INI.

Para mayor información acerca de los drivers ODBC tales como SQL Server y Oracle, ver el archivo de ayuda proporcionado con el manejador.

El siguiente ejemplo usa la sentencia **RegisterDatabase** para crear un nuevo nombre de fuente de datos que usa el manejador Microsoft **SQL** Server y proporciona los diálogos ODBC para agregar información.

El dialogo permite al usuario ingresar la "Descripción".

RegisterDatabase "Working", "SQL Server", **False**, "**DBQ**=Pubs"

Una vez que la sentencia **RegisterDatabase** esta completa, el nuevo archivo **ODBC.INI** es como sigue:

```
[ODBC Data Sources]
Working = SQL Server

[Working]
Driver = C:\WIN31\SYSTEM\SQLSRVR.DLL
Description = Testing
```

```
OemToAnsi = No
Network = dbnmp3
Address = \\WORKING\PIPE\SQL\QUERY
```

Una vez que hayas usado **RegisterDatabase** o el programa administrador del ODBC para establecer una entrada ODBC, puedes usar el nuevo nombre de la fuente de datos con el Data Control, la función **OpenDatabase**, o en la propiedad **Connect** del **TableDef** para unir tablas.

Ajustando valores Timeout

Cuando trabajas con un servidor remoto ODBC, puedes limitar el tiempo que Visual Basic espera antes de asumir que algo estuvo mal y retornar un posible error. Visual Basic proporciona dos parámetros **Timeout**.

- **LoginTimeout** ajusta el tiempo de respuesta cuando te conectas a servidor externo ODBC.
- **QueryTimeout** ajusta el tiempo de respuesta cuando ejecutas una pregunta que accede a un servidor externo ODBC.

LoginTimeout

Para ajustar el número de segundos que Visual Basic espera para responder a un servidor externo ODBC, puedes usar el argumento **LoginTimeout** en la propiedad **Connect** de el Data Control, el argumento **Connect** de la función **OpenDatabase**, o en la propiedad **Connect** del **TableDef**, cuando accedes a una tabla ODBC. Esta característica es especialmente útil cuando el tiempo por default de 20 segundos es demasiado corto. Por ejemplo, ajustar un valor de **Timeout** mayor es esencial, cuando usas una red de área local de largas distancias, o en situaciones donde la red ó el servidor tenga mucha carga de trabajo. Se ajusta a 0 para indicar que el **Timeout** no ocurrirá.

Por ejemplo, para indicar que una conexión ODBC debe ser completada en 10 segundos o menos, debes ajustar el argumento **Connect** como sigue:

```
ConnectArg $ = "ODBC; LOGINTIMEOUT = 10; UID = HOLLY; PWD = RAGS;"
```

QueryTimeout

Para ajustar el número de segundos que Visual Basic debe esperar para que un servidor externo ODBC complete una pregunta (query), puedes usar la propiedad **QueryTimeout** del objeto **Database**. Si tu servidor ODBC soporta esta función, podrás usar esta propiedad para averiguar cuando fueron bloqueadas estas preguntas por carga de trabajo o por problemas en el servidor externo. Una vez que una pregunta entra en el **Timeout**, el servidor externo es llamado para detener el proceso y tu aplicación recibirá un posible error. Un ajuste de valor a 0 indica que no ocurrió ningún **Timeout**.

Acceso a bases de datos Oracle

El manejador ODBC Oracle incluye una librería llamada **Ora6win.dll**. Esta librería es instalada automáticamente y debe estar presente en cualquier estación de trabajo (Workstation) que necesite acceder a una base de datos Oracle. Asegúrate de consultar el archivo **Oracle.txt** para detalles sobre la instalación y configuración de tu sistema, para permitirle conectarse a un servidor Oracle. También necesitarás el software Oracle SQL NetWork. Información sobre este archivo y otros archivos la puedes encontrar en el archivo **Drvoracl.hlp** incluida con Visual Basic.

Generalmente, debes contactar con el soporte de productos Oracle, para preguntas específicas sobre como conectarte a servidores Oracle.

Oracle requiere ambiente MS-DOS y los archivos de configuración Config_Files para validar el archivo **Cinfig.ora**. Cuando tu estés usando otro medio ambiente que también se configure, como es el caso del MS-DOS versión 6, los manejadores de Oracle encuentran los archivos de configuración y los lee. Para resolver este problema, tu necesitas resolver el conflicto que hay entre el archivo de configuración Config y el Autoexec.bat para ejecutar Windows. Por ejemplo, para limpiar el ambiente que genera Config_env, coloca la siguiente línea en el Autoexec.bat para mejorar la entrada a Windows :

CONFIG_ENV =

Accediendo a bases de datos Microsoft SQL Server

Es incluido con Visual Basic el archivo de ayuda **Drvsrvr.hlp** y sirve para cuando quieras acceder tablas de **SQL** Server. Visual Basic puede acceder tablas de bases de datos de algunas versiones de **SQL** Server, incluidas para soportar **Sybase**. Debemos estar seguros de que contamos con los archivos **Instcat.sql** de Microsoft o **Sybase SQL** Server. Para poder acceder a un servidor Microsoft **SQL**, todas las siguientes condiciones deben cumplirse :

1) Tu debes tener correctamente instalados los archivos del ODBC en tu sistema. El Sistema de configuración de Visual Basic instala correctamente estos archivos.

2) Tu puedes usar la sentencia **RegisterDatabase** o la caja de dialogo ODBC para crear un nombre de fuente de bases de datos valido. Todos los nombres de fuentes de datos deben ser instalados en el **ODBC.INI** y contener los servidores **SQL** localizados en la Red, bases de datos por default, login ID, y otra información necesaria para ayudar a Visual Basic cuando necesite conectarse a servidores **SQL**. También tu puedes crear un nombre de fuente de datos, para que tu puedas referirte con este nombre a la función **OpenDatabase** (en el argumento **Databasename**) o en el argumento **Connect** usando el parámetro **DSN=**. Cuando tu agregas una tabla, tu puedes también proporcionar un nombre de fuente de datos en la propiedad **Connect** de el **TableDef**.

3) Si tu estas usando el manejador Microsoft Lan, tu necesitas ejecutar Net Start Workstation y agregarle un password válido. Si tu tienes problemas con el login, consulta tu administrador de la red local.

4) Tu debes tener permiso para acceder al servidor de la Lan en donde esta corriendo Microsoft **SQL** Server. Tu administrador de la red puede configurar lo necesario para que tu puedas acceder.

5) Tu debes de tener una buena conexión al servidor **SQL**. El servidor **SQL** puede limitar el número de simultáneas conexiones. Una vez que sea cubierto el límite del alcance, no se podrá conectar ningún usuario más. Tu administrador del servidor **SQL** puede agregar mas conexiones si tu las necesitas.

6) Tu debes tener permiso para acceder al Servidor SQL, las bases de datos y tablas que tu intentas acceder. El servidor **SQL** tiene sus propios sistemas de seguridad. El login ID y el password te proporcionan acceso a el servidor; Una vez que tu estés conectado, ellos te proporcionan el acceso, solo a limitados recursos del servidor **SQL**.

Marco Teórico

Alfredo Careaga; Miguel Angel Medina; Rubén Adad
Fundamentos de las estructuras de datos relaciónales
Editorial Grupo Noriega Editores
Primera Edición
México D.F. 1992
pp. 225

CODASYL
Systems Committee, Feature Analysis of Generalized Data Base Management Systems, Informe técnico.
ACM (Association for Computing Machinery).
Nueva York, Londres y Amsterdam, mayo 1971.

CODASYL
National Bureau of Standards Handbook 113
Data Description Language Journal of Development, U.S. Department of Commerce, National Bureau of Standards.
Washington, 1974.

CODASYL
Programming Languages Committee, Report, ACM (Association for Computing Machinery).
Data Base Task Group (DBTG)
Nueva York , Londres y Amsterdam, abril 1971.

E. F. Codd,
A Relational Model of Data for Large Shared Data Banks
Association for Computing Machinery
Nueva York, Junio 1970
pp. 680

E. F. Codd
Access Control for Relational Data Base System
British Computer Society
Londres, 1973
pp. 844

Microsoft Corporation
Programing System for Windows Versión 3.0 (Profesional Features Book 2)
Microsoft Visual Basic
Professional Edition
USA 1993
pp. 320

Microsoft Corporation
Programing System for Windows Versión 3.0 (Programmer's Guide)
Microsoft Visual Basic
Professional Edition
USA 1993
pp. 711

Microsoft Corporation
Programing System for Windows Versión 3.0 (User Manual)
Microsoft Visual Basic
Professional Edition
USA 1993
pp. 6304

Visual Basic
Editorial Grupo Noriega Editores
Primera Edición
México D.F. 1992
pp. 225