Introducción a los objetos de acceso a datos

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dahowDataAccessOverviewC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dahowDataAccessOverviewX":1} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dahowDataAccessOverviewS"}

Los objetos de acceso a datos (DAO) le habilitan para utilizar un lenguaje de programación para tener acceso y manipular datos en bases de datos locales o remotas y administrar bases de datos, los objetos y la estructura.

Modelos de objeto

El DAO admite dos entornos diferentes de bases de datos o "espacios de trabajo."

- Los espacios de trabajo Microsoft Jet permiten tener acceso a datos en bases de datos Microsoft Jet, bases de datos Microsoft conectadas a ODBC y orígenes de datos ISAM instalable en otros formatos, como Paradox o Lotus 1-2-3.
- Los espacios de trabajo ODBCDirect permiten tener acceso servidores de bases de datos mediante ODBC, sin cargar el motor de base de datos Microsoft Jet.

Utilice el espacio de trabajo Microsoft Jet cuando abra una base de datos Microsoft Jet (archivo .mdb) u otra base de datos de escritorio ISAM o cuando necesite aprovecharse de la características particulares de Microsoft Jet, como la posibilidad de unir datos de diferentes formatos de base de datos.

El espacio de trabajo ODBCDirect proporciona una alternativa cuando sólo necesita ejecutar consultas o procedimientos almacenados en un servidor, como Microsoft SQL Server o cuando la aplicación cliente necesita las capacidades específicas de ODBC, como las actualizaciones por lotes o la ejecución de una consulta asíncrona.

Objetos DAO

Hay 17 tipos diferentes de objetos DAO. Puede declarar nuevas variables de objeto DAO para cualquier tipo de objeto.

Por ejemplo, el siguiente código Visual Basic for Applications (VBA) crea variables de objeto para un objeto **Database**, un objeto **Recordset** de tipo Dynaset y un objeto **Field**:

```
Dim dbsEjemplo As Database
Dim rstEjemplo As Recordset
Dim fldEjemplo As Field

Set dbsEjemplo = OpenDatabase("Editores.mdb")
Set rstEjemplo = dbsEjemplo.OpenRecordset("Autores", _dbOpenDynaset)
Set fldEjemplo = rstEjemplo.Fields("IdAutor")
```

Colecciones DAO

Cada tipo de objeto DAO distinto de **DBEngine** también tiene una colección correspondiente. Una colección incluye todos los objetos existentes de ese tipo. Por ejemplo, la colección **Recordsets** contiene todos los objetos **Recordset** abiertos. Cada colección es "propiedad" de otro objeto en el siguiente nivel superior de la jerarquía. Un objeto **Database** "es propiedad" de una colección **Recordsets**. Excepto para los objetos **Connection** y **Error**, cada objeto DAO tiene una colección **Properties**.

La mayoría de los objetos DAO tienen colecciones y propiedades predeterminadas. Por ejemplo, la colección predeterminada de un objeto **Recordset** es la colección **Fields** y la propiedad predeterminada de un objeto **Field** es la propiedad **Value**. Puede simplificar el código aprovechándose de estas predeterminaciones. Por ejemplo, el siguiente código de ejemplo establece el valor del campo IdEditor en el registro activo:

```
rstEjemplo!IdEditor=99
```

Objetos DBEngine y Workspace

Todos los objetos DAO se derivan del objeto **DBEngine**. Puede establecer la propiedad **DefaultType** en el objeto **DBEngine** para determinar el tipo de espacio de trabajo (Microsoft Jet u ODBCDirect) para crear en las llamadas al método **CreateWorkspace** siguiente o puede sobrescribir esta propiedad con el argumento *tipo* en el mismo método **CreateWorkspace**. Cuando la aplicación crea un espacio de trabajo, la biblioteca adecuada - el motor de base de datos Microsoft Jet u ODBC - se carga en memoria en este momento.

Puede abrir objetos **Workspace** adicionales si los necesita. Cada objeto **Workspace** tiene un Id de usuario y una contraseña asociada.

Uso del espacio de trabajo Microsoft Jet

Abrir una base de datos

Para abrir una base de dato, simplemente abra un objeto **Database** existente o cree uno nuevo. Este objeto puede representar una base de datos Microsoft Jet (archivo .mdb), una base de datos ISAM (por ejemplo, Paradox) o una base de datos ODBC conectada mediante el motor de base de datos Microsoft Jet (también conocido como una "base de datos ODBC conectada a Microsoft Jet").

Lenguaje de definición de datos (DDL)

Puede utilizar variables de objeto y otras características DDL para modificar la estructura de la base de datos. Por ejemplo, puede agregar un nuevo objeto **Field** a una tabla existente con el siguiente código:

```
Dim dbs As Database, tdf As TableDef, fld As Field
' Abre una base de datos.
Set dbs = OpenDatabase("Editores.mdb")
' Open a TableDef.
Set tdf = dbs.TableDefs("Autores")
' Crea un nuevo campo.
Set fld = tdf.CreateField("Dirección", dbText, 20)
' Anexa el campo a la colección Fields del objeto TableDef.
tdf.Fields.Append fld
```

Este código crea un nueva variable de objeto para un objeto **Field** y la agrega a un objeto **TableDef** con el método **Append**. Ya que un objeto **TableDef** contiene la definición de una tabla, la tabla tiene ahora un campo llamado Dirección para escribir datos. Del mismo modo, puede crear nuevas tablas e índices.

Manipulación de datos

El DAO proporciona un excelente conjunto de herramientas de manipulación de datos. Puede crear un objeto **Recordset** para consultar convenientemente una base de datos y manipular el conjunto de registros resultante. El método **OpenRecordset** acepta una cadena SQL o un nombre de un objeto **QueryDef** (consulta almacenada) como un argumento de origen de datos, o se puede abrir desde un objeto **QueryDef** o un objeto **TableDef**, utilizando este objeto como el origen de datos. El objeto **Recordset** resultante presenta un conjunto extremadamente rico de propiedades y métodos con el que examinar y modificar datos.

El objeto **Recordset** está disponible en cuatro tipos diferentes: Table, Dynaset, Forward-only y Snapshot.

Transacciones

Todos los objetos **Database** abiertos a través de un objeto **Workspace** comparten un alcance de

transacción común. Esto quiere decir que cuando utiliza el método **BeginTrans** en un objeto **Workspace**, se aplica a todas las bases de datos abiertas con ese objeto **Workspace**. Del mismo modo, cuando utiliza el método **CommitTrans** a través del objeto **Workspace**, se aplica a todas las bases de datos abiertas en el objeto **Workspace**.

Réplicas

Puede utilizar la réplica de base de datos para crear y mantener réplicas de una base de datos modelo de Microsoft Jet utilizando el método **Synchronize** para actualizar periódicamente todas o parte de las réplicas o para copiar datos nuevos de una réplica a otra. También pude restringir la actualización a sólo los registros seleccionados, utilizando la propiedad **ReplicaFilter** y después sincronizar esos registros con el método **PopulatePartial**.

Seguridad

Puede restringir el acceso a una o más bases de datos .mdb o a las tablas utilizando los valores de seguridad establecidos y administrados por el motor de base de datos Microsoft Jet. En el código, puede establecer los objetos **Group** y **User** para definir el alcance y nivel de permisos disponibles a usuarios individuales en una base objeto a objeto. Por ejemplo, puede establecer permisos para un usuario específico para proporcionar acceso de sólo lectura a una tabla y acceso total a otra.

Uso del Modelo de objetos ODBCDirect

Conexión a una base de datos

Un objeto **Connection** es parecido a un objeto **Database**. De hecho, un objeto **Connection** y un objeto **Database** representan referencias diferentes al mismo objeto y propiedades en cada uno de los dos tipos de objeto que le permiten obtener una referencia al otro objeto correspondiente, que simplifica la tarea de convertir las aplicaciones cliente ODBC que utilizan Microsoft Jet para utilizar en vez de ello ODBCDirect. Utilice el método **OpenConnection** para conectarse a un origen de datos ODBC. El objeto **Connection** resultante contiene información acerca de la conexión, como el nombre del servidor y el nombre del origen de datos.

Consultas

Aunque el DAO no admite consulta almacenadas en un espacio de trabajo ODBCDirect, se puede crear una consulta compilada como un objeto **QueryDef** y se puede utilizar para ejecutar consultas de acción y también para ejecutar procedimientos almacenados en el servidor. La propiedad **Prepare** le permite decidir si crear un procedimiento almacenado temporal y privado en el servidor de un **QueryDef** antes de ejecutar la consulta actual.

Las consultas de parámetros también se pueden transferir al servidor, utilizando objetos **Parameter** en el **QueryDef**. La propiedad **Direction** le permite especificar un **Parameter** como entrada, salida o ambos o aceptar un valor de un procedimiento almacenado.

Manipulación de datos

Crear un objeto **Recordset** es una forma oportuna de consultar una base de datos y manipular el conjunto de registros resultante. El método **OpenRecordset** acepta una cadena SQL o un objeto **QueryDef** (consulta almacenada) como un argumento de origen de datos. El objeto **Recordset** resultante presenta un conjunto extremadamente rico de propiedades y métodos con el que examinar y modificar datos.

El objeto **Recordset** está disponible en cuatro tipos diferentes: Table, Dynaset, Forward-only y Snapshot - que corresponden a los tipos de cursores ODBC: Dynamic, Keyset, Forward-only y Static.

Está disponible una biblioteca de cursor de actualización por lotes para aplicaciones cliente que necesitan trabajar con un cursor sin mantener bloqueos en el servidor o sin emitir las peticiones de actualización de un registro al mismo tiempo. En cambio, el cliente almacena la información de actualización de muchos registros en un búfer local (o "por lotes") y después envía una actualización

por lotes.

Ejecución de método asícrono

Los métodos **Execute**, **MoveLast**, **OpenConnection** y **OpenRecordset** presentan la opción **dbRunAsync**. Esto permite a la aplicación cliente realizar otras tareas (como, por ejemplo, cargar formularios) mientras se está ejecutando el método. Puede comprobar la propiedad **StillExecuting** para ver si la tarea se completó y para terminar una tarea asíncrona con el método **Cancel**.

Métodos de acceso a datos por objeto

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dahowDataAccessOverviewC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dahowDataAccessOverviewS"}

Esta referencia agrupa todos los métodos DAO por objeto. Para ver si un método concreto está disponible para Microsoft Jet o para los espacios de trabajo ODBC, consulte el tema de Ayuda de ese método.

Connection
Contenedor - sin métodos
Database
DBEngine
Document
Error - sin métodos
Field
Group
Index
Parámetro - sin métodos
Propiedad - sin métodos
QueryDef
Recordset
Relation
TableDef
User
Workspace

Métodos de acceso a datos para espacios de trabajo Microsoft Jet

{ewc HLP95EN.dll, DYNALINK, "Vea tambiï¿½n":"daidxMethodsReferenceJetC "} {ewc HLP95EN.dll, DYNALINK, "Detalles":"daidxMethodsReferenceJetS "}

Esta referencia enumera alfabéticamente todos los métodos DAO disponibles para espacios de trabajo Microsoft Jet (archivos de base de datos ISAM).

A-C

AddNew
Append
AppendChunk
BeginTrans
CancelUpdate
Clone
Close
CommitTrans
CompactDatabase
CopyQueryDef
CreateDatabase
CreateField
CreateGroup
CreateIndex
CreateProperty
CreateQueryDef
CreateRelation
CreateTableDef
CreateUser
CreateWorkspace

D-M

Delete
Edit
Execute
FillCache
FindFirst
FindLast
FindNext
FindPrevious
GetChunk
GetRows
Idle
MakeReplica
Move
MoveFirst
MoveLast
MoveNext
MovePrevious

N-Z

NewPassword
OpenDatabase
OpenRecordset
PopulatePartial
Refresh
RefreshLink
RegisterDatabase
RepairDatabase
Requery
Rollback
Seek
SetOption
Synchronize
Update

Métodos de acceso a datos para espacios de trabajo ODBCDirect

{ewc HLP95EN.dll, DYNALINK, "Vea tambiï¿½n":"daidxMethodsReferenceODBCC "} {ewc HLP95EN.dll, DYNALINK, "Detalles":"daidxMethodsReferenceODBCS "}

Esta referencia enumera alfabéticamente todos los métodos DAO disponibles para espacios de trabajo ODBCDirect.

A-C

AddNew
Append
AppendChunk
BeginTrans
Cancel
CancelUpdate
Clone
Close
CommitTrans
CreateQueryDef
CreateWorkspace

D-M

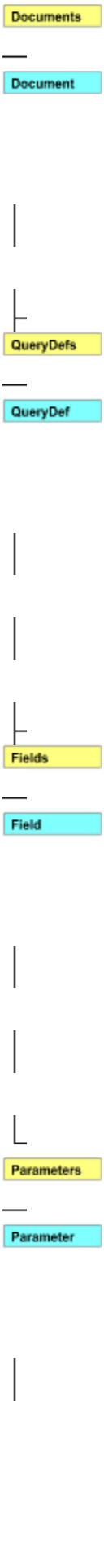
Delete
Edit
Execute
GetChunk
GetRows
Move
MoveFirst
MoveLast
MoveNext
MovePrevious

N-Z

NextRecordset
OpenConnection
OpenDatabase
OpenRecordset
Refresh
RegisterDatabase
Requery
Rollback
Update

Modelo de objeto de acceso a datos para espacios de trabajo Microsoft Jet

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daconmsjetdatabaseengine25c"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daconMSJetDatabaseEngine25S"}


Documents
Document
QueryDefs
QueryDef
Fields
Field
Parameters
Parameter


Recordsets
Recordset
Fields
Field
Relations
Relation
Fields
Field


TableDefs
TableDef
Fields
Field
Indexes
Index


Fields
Field
Groups
Group
Users
User
Users
User
Groups
Group

Modelo de objeto de acceso a datos para espacios de trabajo ODBCDirect

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daconMSJetDatabaseEngine35C"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daconMSJetDatabaseEngine35S"}

Parameters

Recordsets

Referencia de colecciones y objetos de acceso a datos

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daidxObjectsandCollectionsC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daidxObjectsandCollectionsS"}

Los objetos y las colecciones de acceso a datos proporcionan el marco para utilizar códigos de instrucciones para crear y manipular los componentes del sistema de base de datos. Los objetos y las colecciones tienen propiedades que describen las características de los componentes de la base de datos y los métodos que se emplean para manipularlas. Estos objetos y colecciones forman conjuntamente un modelo jerárquico de la estructura de la base de datos, la cual podrá controlar por medio de programas.

Los objetos y las colecciones ofrecen diferentes tipos de relaciones de contención. Los objetos contienen cero o más colecciones, todas ellas de tipos distintos y las colecciones contienen cero o más objetos, siendo todos del mismo tipo. Aunque los objetos y colecciones son entidades similares, la distinción diferencia los dos tipos de relaciones.

En la siguiente tabla, el tipo de colección de la primera columna contiene el tipo de objeto de la segunda columna. La tercera columna describe lo que representa cada tipo de objeto.

Colección	Objeto	Descripción
Connections	Connection	Información acerca de una conexión a un origen de datos ODBC (sólo espacios de trabajo ODBCDirect)
Containers	Container	Almacenamiento de información sobre un objeto predefinido (sólo espacios de trabajo Microsoft Jet)
Databases	Database	Una base de datos abierta
Ninguna	DBEngine	El motor de base de datos Microsoft Jet
Documents	Document	Información sobre un objeto guardado predefinido (sólo espacios de trabajo Microsoft Jet)
Errors	Error	Información sobre errores asociados a este objeto
Fields	Field	Una columna que forma parte de una tabla, consulta, índice, relación o conjunto de registros
Groups	Group	Un grupo de cuentas de usuario (sólo espacios de trabajo Microsoft Jet)
Indexes	Index	Orden y exclusividad predefinidos de los valores de una tabla (sólo espacios de trabajo Microsoft Jet)
Parameters	Parameter	Un parámetro para una consulta con parámetros
Properties	Property	Una propiedad incorporada o definida por el usuario
QueryDefs	QueryDef	Una definición de consulta guardada

Recordsets	Recordset	Los registros de una tabla base o consulta
Relations	Relation	Una relación entre campos de tablas y consultas (sólo espacios de trabajo Microsoft Jet)
TableDefs	TableDef	Una definición de tabla guardada (sólo espacios de trabajo Microsoft Jet)
Users	User	Una cuenta de usuario (sólo espacios de trabajo Microsoft Jet)
Workspaces	Workspace	Una sesión del motor de base de datos Microsoft Jet

Propiedades del acceso a datos por objeto

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daidxPropertiesC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daidxPropertiesS"}

Esta referencia agrupa todos las propiedades DAO por objeto o colección. Para determinar si una propiedad concreta está disponible para Microsoft Jet o para los espacios de trabajo ODBC, consulte el tema de Ayuda de esa propiedad.

Connection
Container
Database
DBEngine
Document
Error
Field
Group
Index
Parameter
Property
QueryDef
Recordset
Relation
TableDef
User
Workspace

Propiedades del acceso a datos para espacios de trabajo Microsoft Jet

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daidxPropertiesReferenceJetC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daidxPropertiesReferenceJetS"}

Esta referencia enumera alfabéticamente todas las propiedades DAO disponibles para espacios de trabajo Microsoft Jet.

A-C

AbsolutePosition
AllowZeroLength
AllPermissions
Attributes
BOF
Bookmark
Bookmarkable
CacheSize
CacheStart
Clustered
CollatingOrder
ConflictTable
Connect
Container
Count

D-H

DataUpdatable
DateCreated
DefaultUser
DefaultPassword
DefaultValue
Description
DesignMasterID
DistinctCount
EditMode
EOF
FieldSize
Filter
Foreign
ForeignName
ForeignTable
HelpContext
HelpFile

I-O

IgnoreNulls
Index
Inherit
Inherited
IniPath
IsolateODBCTrans
KeepLocal
LastModified
LastUpdated
LockEdits
LoginTimeout
LogMessages
MaxRecords
Name
NoMatch
Number
ODBCTimeout
OrdinalPosition
Owner

P-R

PartialReplica
Password
PercentPosition
RecordsAffected
Replicable
ReplicableBool

Permissions
PID
Primary
QueryTimeout
RecordCount
ReplicaFilter
ReplicaID
Required
Restartable
ReturnsRecords

S-Z

Size
Sort
Source
SourceField
SourceTable
SourceTableName
SQL
SystemDB
Table
Transactions
Type
Unique
Updatable
UserName
V1xNullBehavior
ValidateOnSet
ValidationRule
ValidationText
Value
Version

Propiedades del acceso a datos para espacios de trabajo ODBCDirect

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daidxPropertiesReferenceODBCC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daidxPropertiesReferenceODBCS"}

Esta referencia enumera alfabéticamente todas las propiedades DAO disponibles para espacios de trabajo ODBCDirect.

A-D

AbsolutePosition
Attributes
BatchCollisionCount
BatchCollisions
BatchSize
BOF
Bookmark
Bookmarkable
CacheSize
Connect
Connection
Count
Database
DataUpdatable
DefaultCursorDriver
DefaultType
Description
Direction

E-Q

EditMode
EOF
FieldSize
HelpContext
HelpFile
LastModified
LockEdits
LoginTimeout
LogMessages
MaxRecords
Name
Number
ODBCTimeout
OrdinalPosition
OriginalValue
PercentPosition
Prepare
QueryTimeout

R-Z

RecordCount
RecordsAffected
RecordStatus
Restartable
Size
Source
SourceField
SourceTable
SQL
StillExecuting
Transactions
Type
Updatable
UpdateOptions
UserName
Value
Version
VisibleValue

Lo nuevo en DAO

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daidxWhatsNewInDAOC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daidxWhatsNewInDAOS"}

DAO 3.5 introduce un nuevo modo de conexión cliente/servidor, llamado "ODBCDirect." ODBCDirect establece una conexión directamente a un origen de datos ODBC, sin cargar el motor de base de datos Microsoft Jet en memoria y es más útil en situaciones donde se necesitan características específicas de ODBC.

Para las bases de datos Microsoft Jet, también hay nuevas interfaces para exponer la nueva característica de réplica parcial de Microsoft Jet.

Nota Puede enviar consultas DAO a una gran variedad de servidores de bases de datos diferentes con ODBCDirect y diferentes servidores reconocerán dialectos de SQL con pequeñas diferencias. Por lo tanto, no se proporciona Ayuda interactiva para Microsoft Jet SQL, aunque todavía se incluye Ayuda en pantalla para Microsoft Jet SQL mediante el menú Ayuda. Asegúrese de comprobar la documentación de referencia adecuada al dialecto SQL del servidor de base de datos cuando utilice conexiones ODBCDirect o consulta de paso a través en aplicaciones cliente/servidor conectadas a Microsoft Jet.

Nuevas interfaces DAO 3.5 para ODBCDirect

- Objeto **Connection** - una conexión a una base de datos ODBC.
- Método **Cancel** (en objetos **Connection**, **QueryDef** y **Recordset**) - Cancela la ejecución de una operación asíncrona.
- Método **NextRecordset** (en objetos **Recordset**) - Recupera el siguiente conjunto de registros, si hay, devueltos por una consulta que devuelve múltiples conjuntos de registros en una llamada **OpenRecordset** e indica si se recuperó otro conjunto de registros.
- Método **OpenConnection** (en objetos **Workspace**) - Abre un objeto **Connection** en un origen de datos ODBC.
- Propiedad **BatchCollisionCount** (en objetos **Recordset**) - Devuelve el número de registros que no se completaron durante la última actualización por lotes.
- Propiedad **BatchCollisions** (en objetos **Recordset**) - Devuelve una matriz de marcador de posición que indican las filas que generaron colisiones en la última actualización por lotes.
- Propiedad **BatchSize** (en objetos **Recordset**) - Establece o devuelve el número de instrucciones remitidas al servidor en cada proceso por lotes.
- Propiedad **Connection** (en objetos **Database** y **Recordset**) - Devuelve el objeto **Connection** que corresponde al objeto **Database** o que pertenece el **Recordset**.
- Propiedad **Database** (en objetos **Connection**) - Devuelve el nombre del objeto **Database** que corresponde al objeto **Connection**.
- Propiedad **DefaultCursorDriver** (en objetos **Workspace**) - Establece o devuelve el tipo de controlador de cursor utilizado por los objetos **Recordset** de ODBCDirect.
- Propiedad **DefaultType** (en objetos **DBEngine**) - Indica qué tipo de espacio de trabajo (Microsoft Jet u ODBCDirect) se creará en la llamada al método **CreateWorkspace** siguiente.
- Propiedad **Direction** (en objetos **Parameter**) - Indica si un objeto **Parameter** representa un parámetro de entrada, un parámetro de salida o ambos o si el parámetro es el valor devuelto por un procedimiento almacenado.
- Propiedad **MaxRecords** (en objetos **QueryDef**) - Establece o devuelve el número máximo de registros a devolver en una consulta.
- Propiedad **OriginalValue** (en objetos **Field**) - Devuelve el valor del objeto **Field** en la base de datos que existía cuando comenzó la última actualización por lotes.
- Propiedad **Prepare** (en objetos **QueryDef**) - Devuelve un valor que indica si la consulta se debe preparar en el servidor como un procedimiento almacenado temporal con la función **SQLPrepare**

ODBC antes de la ejecución o cuando se ejecuta utilizando la función **SQLExecDirect** de ODBC.

- Propiedad **RecordStatus** (en objetos **Recordset**) - Devuelve un valor que indica el estado de actualización del registro activo si es una parte de una actualización por lotes.
- Propiedad **StillExecuting** (en objetos **Connection**, **QueryDef** y **Recordset**) - Devuelve un valor que indica si se ha terminado o no de ejecutar una operación asíncrona.
- Propiedad **UpdateOptions** (en objetos **Recordset**) - Devuelve un valor que indica cómo se construye la cláusula WHERE para cada registro durante una actualización por lotes y cómo la actualización se debe ejecutar.
- Propiedad **VisibleValue** (en objetos **Recordset**) - Devuelve un valor activo en la base de datos que es más reciente que el **OriginalValue** como se determina en un conflicto de actualización por lotes.

Nuevas posibilidades con ODBCDirect

Conexiones al servidor

Sólo disponible en el modelo de objeto ODBCDirect, el nuevo modelo **Connection** contiene información acerca de la conexión a un origen de datos origen de datos ODBC, como el nombre del servidor, el nombre del origen de datos y otros. Es similar a un objeto **Database** y le parecerá muy familiar si ya ha abierto un objeto **Database** en un origen de datos ODBC. De hecho, un objeto **Connection** y un objeto **Database** representan referencias diferentes al mismo objeto y nuevas propiedades en cada uno de esos dos tipos de objeto le permiten obtener una referencia al otro objeto correspondiente, lo que simplifica la tarea de convertir aplicaciones cliente ODBC existentes que utilizan Microsoft Jet para utilizar en vez de ello ODBCDirect.

Actualizaciones por lotes

Está disponible un nuevo cursor de actualización por lotes para aplicaciones cliente que necesitan trabajar con un cursor sin mantener bloqueos en el servidor o sin emitir las solicitudes de actualización de un registro al mismo tiempo. En cambio, el cliente almacena la información de actualización de muchos registros en un búfer local (o "por lotes") y después emite una actualización por lotes.

Debido a que hay un tiempo de retraso entre abrir un **Recordset** y enviar un proceso por lotes de actualizaciones de ese **Recordset** de vuelta al servidor, otros usuarios tienen una oportunidad para cambiar los datos originales antes que de sus cambios se hayan enviado al servidor, de este modo sus cambios "colisionan" con cambios de otros usuarios. Están disponibles varias características nuevas para ayudarle a determinar dónde se producen tales colisiones, siguiendo una actualización por lotes y proporcionándole algunas opciones para resolverlas.

Asynchronous Method Execution

Los métodos **Execute**, **MoveLast**, **OpenConnection** y **OpenRecordset** presentan la opción **dbRunAsync**. Esto permite a la aplicación cliente realizar otras tareas (como, por ejemplo, cargar formularios) mientras se está ejecutando el método. También puede realizar un sondeo para ver si la tarea se completó y para terminar una tarea asíncrona.

Soporte de cliente para cursores ODBC

Cuatro tipos de **Recordset** diferentes admiten los siguientes tipos de cursores ODBC:

Cursor ODBC	Tipo de Recordset
Dynamic	dbOpenDynamic (Nuevo en el DAO 3.5)
Dynaset	dbOpenDynaset
Forward-only	dbOpenForwardOnly (Nuevo en el DAO 3.5)
Static	dbOpenSnapshot

Interfaces nuevas en el DAO 3.5 para el motor de la base de datos Microsoft Jet

- Método **PopulatePartial** (en objetos **Database**) - Sincroniza cualquier cambio en una réplica parcial con la réplica completa, borra todos los registros en la réplica parcial y después rellena la réplica parcial basada en los filtros de réplica activos.
- Método **SetOption** (en objetos **DBEngine**) - Sobrescribe los valores del Registro para el motor de base de datos de Microsoft Jet durante la instancia actual del DAO.
- Propiedad **FieldSize** (en objetos **Field**) - Reemplaza el método **FieldSize**. Sintácticamente, su uso es el mismo, pero esto no necesitará cambios en su código existente.
- Propiedad **MaxRecords** (en objetos **QueryDef**) - Establece o devuelve el número máximo de registros a devolver por una consulta.
- Propiedad **ReplicaFilter** (en objetos **TableDef**) - Devuelve un valor que indica qué subconjunto de registros se replica a esa tabla desde una réplica completa.
- Propiedad **PartialReplica** (en objetos **Relation**) - Indica qué objeto **Relation** se debe considerar cuando se llena una réplica parcial desde una réplica completa.

Nuevas posibilidades con el motor de base de datos Microsoft Jet

Parcialización de réplica

La versión 3.5 del motor de base de datos Microsoft Jet permite a los usuarios replicar partes de una tabla en vez de la tabla completa (sólo se permiten restricciones de fila, no de columnas). Hay dos tipos de filtros utilizados en una réplica parcial: booleano y relacional. Los filtros booleanos seleccionan sólo filas que cumplen cierto criterio para limitar las filas en una tabla que están replicadas. El DAO representa este filtro con la propiedad **ReplicaFilter** en un objeto **TableDef**. Los filtros relacionales fuerzan una relación entre tablas replicadas parcialmente para limitar las filas en una tabla que están replicadas. Con DAO, puede establecer la propiedad **PartialReplica** en un objeto **Relation** que permite que ese objeto **Relation** se utilice en una réplica parcial.

Nuevo tipo de Recordset

En el DAO 3.5, **dbOpenForwardOnly** es un nuevo argumento *tipo* para el método **OpenRecordset**. Este nuevo tipo de objeto **Recordset** se comporta de la misma forma que un **Recordset** de tipo Snapshot del DAO 3.0 abierto con la opción **dbForwardOnly**.

Sobreescritura del Registro en tiempo de ejecución

El nuevo método **SetOption** le permite sobrescribir los valores del Registro de Microsoft Jet en tiempo de ejecución. Esto permite ajustar el rendimiento de las consultas Microsoft Jet, tiempos de espera y otros.

Características obsoletas en DAO

{ewc HLP95EN.dll, DYNALINK, "Vea tambiï¿½n":"dahowObsoleteFeaturesC"} {ewc HLP95EN.dll, DYNALINK, "Ejemplo":"dahowObsoleteFeaturesX":1} {ewc HLP95EN.dll, DYNALINK, "Detalles":"dahowObsoleteFeaturesS"}

Versiones 1.x y 2.0 de Microsoft Access y versión 3.0 de Microsoft Visual Basic utilizada con versiones anteriores de DAO. Varios objetos, métodos, propiedades e instrucciones en estas versiones se consideran "obsoletas", pero todavía se admiten para compatibilidad con versiones anteriores con el código de usuario existente.

Esta es una lista de métodos, propiedades, objetos e instrucciones de DAO que se han reemplazado por otras características más potentes, flexibles y fáciles de utilizar. Cada característica obsoleta en la lista tiene su correspondiente reemplazo futuro.

Característica obsoleta	Característica de reemplazo
CreateDynaset (todos los métodos)	**OpenRecordset** (método)
CreateSnapshot (todos los métodos)	**OpenRecordset** (método)
ListFields (todos los métodos)	**Fields** (colección)
ListIndexes (todos los métodos)	**Indexes** (colección)
CompactDatabase (instrucción)	**DBEngine.CompactDatabase** (método)
CreateDatabase (instrucción)	**DBEngine.CreateDatabase** (método)
DBEngine.FreeLocks (método)	**DBEngine.Idle** (método)
DBEngine.SetDefaultWorkspace (método)	**DBEngine.DefaultUser** y **DBEngine.Password** (propiedades)
DBEngine.SetDataAccessOption (método)	**DBEngine.IniPath** (propiedad)
Database.BeginTrans (método)	**Workspace.BeginTrans** (método)
Database.CommitTrans (método)	**Workspace.CommitTrans** (método)
Database.Rollback (método)	**Workspace.Rollback** (método)
Database.DeleteQuerydef (método)	**Delete** (método)
Database.ExecuteSQL (método)	**Execute** (método)
Database.ListTables (método)	**Tabledefs** (colección)
Database.OpenQuerydef (método)	**Querydefs** (colección)
Database.OpenTable (método)	**OpenRecordset** (método)
FieldSize (método)	**FieldSize** (propiedad)
Index.Fields (propiedad)	**Index.Fields** (colección)
OpenDatabase (instrucción)	**DBEngine.OpenDatabase** (método)
Querydef.ListParameters (método)	**Parameters** (colección)
Snapshot (objeto)	**Recordset** (objeto)
Dynaset (objeto)	**Recordset** (objeto)
Table (objeto)	**Recordset** (objeto)

Container (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjContainerX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjContainerP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjContainerM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjContainerS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjContainerU":1}

Un objeto **Container** agrupa tipos similares de objetos **Document**.

Comentarios

Cada objeto **Database** tiene una colección **Containers** que consta de objetos **Container** incorporados. Las aplicaciones pueden definir sus propios tipos de documentos y sus contenedores correspondientes (sólo bases de datos Microsoft Jet); sin embargo, estos objetos no siempre se admiten mediante DAO.

Algunos de estos objetos **Container** los define el motor de base de datos Microsoft Jet y otros los definen otras aplicaciones. La tabla siguiente muestra el nombre de cada objeto **Container**, definido por el motor de base de datos Microsoft Jet y el tipo de información que contiene.

Nombre del contenedor	Contiene información sobre
Databases	La base de datos contenedora
Tables	Las tablas y consultas guardadas
Relations	Las relaciones guardadas

Nota No confunda los tipos de objetos **Container** que aparecen en la tabla anterior con los tipos de colección del mismo nombre. Los objetos **Container** del tipo Databases hacen referencia a todos los objetos guardados del tipo especificado, pero las colecciones **Databases** sólo hacen referencia a los objetos abiertos del tipo especificado.

Cada objeto **Container** tiene una colección **Documents** que contiene un objeto **Document** para cada instancia del tipo de objeto que describe. Normalmente se utiliza un objeto **Container** con uno de los objetos **Document** de la colección **Containers** como un vínculo intermedio con la información que contiene el objeto **Document**.

Con un objeto **Container** existente, puede:

- Utilizar la propiedad **Name** para devolver el nombre predefinido del objeto **Container**.
- Utilizar la propiedad **Owner** para establecer o devolver el propietario del objeto **Container**. Para establecer la propiedad **Owner**, deberá disponer de autorización de escritura para el objeto **Container** y establecer la propiedad como el nombre de un objeto **User** o **Group** existente.
- Establecer autorizaciones de acceso para el objeto **Container** utilizando las propiedades **Permissions** y **UserName**, cualquier objeto **Document** creado en la colección **Documents** de un objeto **Container** hereda estos valores de autorización de acceso.

debido a que los objetos **Container** están incorporados, no puede crear nuevos objetos **Container** o eliminar los que existen.

Puede referirse a un objeto **Container** de una colección por su número ordinal o por el valor de su propiedad **Name**, utilizando uno de estas formas de sintaxis:

Containers(0)
Containers("*nombre*")
Containers![*nombre*]

Containers (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolContainerX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolContainerP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolContainerM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolContainerS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolContainerU":1}

Una colección **Containers** contiene todos los objetos **Container** que están definidos en una base de datos (sólo bases de datos Microsoft Jet).

Comentarios

Cada objeto **Database** tiene una colección **Containers** que consta de objetos **Container** incorporados. Algunos de estos objetos **Container** los define el motor de base de datos Microsoft Jet otros los definen otras aplicaciones.

Container (Objeto), Containers (Colección), Resumen

Container (Objeto)

El objeto **Container** no tiene métodos, pero sí las siguientes colecciones y propiedades.

Colecciones

Documents (predeterminada)
Properties

Propiedades

AllPermissions
Inherit
Name
Owner
Permissions
UserName

Containers (Colección)

Una colección **Containers** aparece en cada objeto **Database** de una base de datos Microsoft Jet y contiene este método o esta propiedad.

Método

Refresh

Propiedad

Count

Database (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjDatabaseP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjDatabaseM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjDatabaseS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjDatabaseU":1}

Un objeto **Database** representa una base de datos abierta.

Comentarios

Para manipular una base de datos abierta se utiliza un objeto **Database** y sus métodos y propiedades. En cualquier tipo de bases de datos, puede:

- Utilizar el método **Execute** para ejecutar una consulta de acción.
- Establecer la propiedad **Connect** para establecer una conexión a un origen de datos ODBC.
- Establecer la propiedad **QueryTimeout** para limitar el tiempo de espera para que una consulta se ejecute a través de un origen de datos ODBC.
- Utilizar la propiedad **RecordsAffected** para determinar cuántos registros se cambiaron en una consulta de acción.
- Utilizar el método **OpenRecordset** para ejecutar una consulta de selección y crear un objeto **Recordset**.
- Utilizar la propiedad **Version** para determinar qué versión de un motor de base de datos creó la base de datos.

Con una base de datos Microsoft Jet (archivo .mdb), también puede utilizar otros métodos, propiedades y colecciones para manipular un objeto **Database**, y para crear, modificar u obtener información acerca de sus tablas, consultas y relaciones. Por ejemplo, puede:

- Utilizar los métodos **CreateTableDef** y **CreateRelation** para crear respectivamente tablas y relaciones.
- Utilizar el método **CreateProperty** para definir nuevas propiedades **Database**.
- Utilizar el método **CreateQueryDef** para crear una definición de consulta permanente o temporal.
- Utilizar los métodos **MakeReplica**, **Synchronize** y **PopulatePartial** para crear y sincronizar réplicas parciales o completas de la base de datos.
- Establecer la propiedad **CollatingOrder** para establecer el orden alfabético para campos de tipo carácter en diferentes idiomas.

En un espacio de trabajo ODBCDirect, puede:

- Utilizar la propiedad **Connection** para obtener una referencia al objeto **Connection** que corresponde al objeto **Database**.

Nota Para obtener una lista completa de todos los métodos, propiedades y colecciones disponibles en un objeto **Database** en un espacio de trabajo Microsoft Jet o en un espacio de trabajo ODBCDirect, vea el tema **Resumen**.

Utilice el método **CreateDatabase** para crear un objeto **Database** permanente que se anexa automáticamente a la colección **Databases** y de ese modo guardarlo en disco.

No necesita especificar el objeto **DBEngine** cuando utiliza el método **OpenDatabase**.

Abrir una base de datos con tablas vinculadas no establece automáticamente vínculos a los archivos externos especificados o a los origen de datos ODBC conectados a Microsoft Jet. Debe hacer referencia a los objetos **TableDef** o **Field** de la tabla o abrir un objeto **Recordset**. Si no puede establecer vínculos a estas tablas, se produce un error interceptable. También puede necesitar permiso para tener acceso a la base de datos u otro usuario podría tener abierta la base de datos de modo exclusivo. En estos casos se producen errores interceptable.

También puede utilizar el método **OpenDatabase** para abrir una base de datos externa (como FoxPro, dBASE y Paradox) directamente en vez de abrir una base de datos Microsoft Jet que tiene vínculos con las tablas.

Nota No se recomienda abrir un objeto **Database** directamente en un origen de datos ODBC conectado a Microsoft Jet, como Microsoft SQL Server, porque el rendimiento de la consulta es mucho más lento que cuando se utilizan tablas vinculadas. Sin embargo, el rendimiento no es un problema al abrir un objeto **Database** directamente en un archivo de base de datos ISAM externo, como FoxPro o Paradox.

Cuando un procedimiento que declara un objeto **Database** termina su ejecución, estos objetos **Database** locales se cierran junto con cualquier objeto **Recordset** abierto. Todas las actualizaciones pendientes se pierden y las transacciones pendientes se reanudan, pero no se produce ningún error. Se pueden terminar explícitamente las transacciones pendientes o editar y cerrar objetos **Recordset** y **Database**, antes de abandonar los procedimientos que declaran localmente esas variables del objeto.

Cuando utiliza uno de los métodos de transacción (**BeginTrans**, **CommitTrans** o **Rollback**) en el objeto **Workspace**, estas transacciones se aplican a todas las bases de datos abiertas en el **Workspace** desde el que se abrió el objeto **Database**. Si desea utilizar transacciones independientes, primero debe abrir un objeto **Workspace** adicional y después abrir otro objeto **Database** en ese objeto **Workspace**.

Nota Puede abrir el mismo origen de datos o base de datos más de una vez, creando nombres duplicados en la colección **Databases**. Debe asignar objetos **Database** a variables de objeto y hacer referencia a ellas con un nombre de variable.

Databases (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolDatabaseP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolDatabaseM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolDatabaseS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolDatabaseU":1}

Una colección **Databases** contiene todos los objetos **Database** abiertos o creados en un objeto **Workspace**.

Comentarios

Cuando abre un objeto **Database** existente o crea uno nuevo desde un objeto **Workspace**, se anexa automáticamente a la colección **Databases**. Cuando cierra un objeto **Database** con el método **Close**, se elimina de la colección **Databases**, pero no se borra del disco. Es conveniente cerrar todos los objetos **Recordset** abierto antes de cerrar un objeto **Database**.

En un espacio de trabajo Microsoft Jet, el valor de propiedad **Name** de una base de datos es una cadena que especifica la ruta de acceso del archivo de base de datos. En un espacio de trabajo ODBCDirect, la propiedad **Name** es el nombre del objeto **Connection** correspondiente.

Para hacer referencia a un objeto **Database** en una colección por su número de orden o por su valor de propiedad **Name**, utilice cualquiera de las siguientes formatos de sintaxis:

Databases(0)
Databases("*nombre*")
Databases![*nombre*]

Nota Puede abrir el mismo origen de datos o base de datos más de una vez, creando nombres duplicados en la colección **Databases**. Debe asignar objetos **Database** a variables de objeto y hacer referencia a ellas con un nombre de variable.

Database (Objeto), Databases (Colección), Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumDatabaseS"}

Database (Objeto)

El objeto **Database** contiene las colecciones, métodos y propiedades siguientes.

Leyenda:

Característica disponible sólo en espacios de trabajo Microsoft Jet.

DAO ODBC Característica disponible sólo en espacios de trabajo ODBCDirect.

Colecciones

Containers
Properties

QueryDefs

Recordsets (Predeterminado para)

Relations

TableDefs (Predeterminado para)

Métodos

Close

CreateProperty
CreateQueryDef

CreateRelation

CreateTableDef
Execute

MakeReplica

NewPassword
OpenRecordset
PopulatePartial

Synchronize

Propiedades

CollatingOrder
Connect

Connection

DesignMasterID
Name

QueryTimeout
RecordsAffected

Replicable (definido por el usuario)

ReplicaID
Updatable

V1xNullBehavior
Version

Databases (Colección)

Una colección **Databases** aparece en cada objeto **Workspace** y contiene este método y esta propiedad.

Método
Refresh

Propiedad
Count

DBEngine (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjDBEngineC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjDBEngineX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjDBEngineP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjDBEngineM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjDBEngineS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjDBEngineU":1}

El objeto **DBEngine** es el objeto de nivel más alto en el modelo de objeto DAO.

Comentarios

El objeto **DBEngine** contiene y controla todos los otros objetos en la jerarquía de los objetos del DAO. No puede crear objetos **DBEngine** adicionales y el objeto **DBEngine** no es un elemento de ninguna colección.

Nota Cuando hace referencia a un origen de datos ODBC directamente mediante DAO, se denomina "espacio de trabajo ODBCDirect." Para distinguirlo de un origen de datos ODBC al que hace referencia indirectamente mediante el motor de base de datos Microsoft Jet, utilizando un "espacio de trabajo Microsoft Jet." Cada método de acceso a datos ODBC necesita dos tipos de objetos **Workspace**; puede establecer la propiedad **DefaultType** para elegir el tipo predeterminado de objeto **Workspace** que creará en el objeto **DBEngine**. El tipo de **Workspace** y el origen de datos asociado determina qué objetos, métodos y propiedades del DAO puede utilizar.

Con cualquier tipo de base de datos o conexión, puede:

- Utilizar la propiedad **Version** para obtener el número de versión del DAO.
- Utilizar la propiedad **LoginTimeout** para obtener o establecer el tiempo de espera de registro de ODBC y el método **RegisterDatabase** para proporcionar información de ODBC al motor de base de datos Microsoft Jet. Puede utilizar estas características del mismo modo, sin tener en cuenta si está conectado a un origen de datos ODBC mediante Microsoft Jet o mediante un espacio de trabajo ODBCDirect.
- Utilizar la propiedad **DefaultType** para establecer el tipo predeterminado de conexión de base de datos que utilizarán a continuación los objetos **Workspace** creados - tanto Microsoft Jet como ODBCDirect.
- Utilizar las propiedades **DefaultPassword** y **DefaultUser** para establecer la identificación de usuario y contraseña para el objeto **Workspace** predeterminado.
- Utilizar el método **CreateWorkspace** para crear un objeto **Workspace** nuevo. Puede utilizar argumentos opcionales para sobrescribir los valores de las propiedades **DefaultType**, **DefaultPassword** y **DefaultUser**.
- Utilizar el método **OpenDatabase** para abrir una base de datos en el **Workspace** predeterminado y utilizar los métodos **BeginTrans**, **Commit** y **Rollback** para controlar transacciones en el **Workspace** predeterminado.
- Utilizar la colección **Workspaces** para hacer referencia a objetos **Workspace** específicos.
- Utilizar la colección **Errors** para examinar detalles sobre los errores de acceso a datos.

Otras propiedades y métodos sólo están disponibles cuando utiliza DAO con el motor de base de datos Microsoft Jet. Puede utilizarlos para controlar el motor de base de datos Microsoft Jet, manipular sus propiedades y ejecutar tareas en objetos temporales que no son elementos de colecciones. Por ejemplo, puede:

- Utilizar el método **CreateDatabase** para crear un nuevo objeto Microsoft Jet **Database**.
- Utilizar el método **Idle** para activar el motor de base de datos Microsoft Jet y completar las tareas pendientes.
- Utilice los métodos **CompactDatabase** y **RepairDatabase** para mantener los archivos de base de datos.
- Utilizar las propiedades **IniPath** y **SystemDB** para especificar la ubicación de la información de Registro de Windows de Microsoft Jet y el archivo de información de grupo de trabajo Microsoft Jet, respectivamente. El método **SetOption** le permite sobrescribir los valores del Registro de Windows para el motor de base de datos Microsoft Jet.

Después de cambiar los valores de la propiedad **DefaultType** y **IniPath**, sólo los objetos **Workspace** subsiguientes reflejarán estos cambios.

Nota Para obtener una lista completa acerca de los métodos, propiedades y colecciones disponibles en el objeto **DBEngine**, vea el tema **Resumen**.

Para hacer referencia a una colección que pertenezca al objeto **DBEngine**, a un método o a una propiedad que se aplique a este objeto, utilice la siguiente sintaxis:

[**DBEngine.**][*colección* | *método* | *propiedad*]

DBEngine (Objeto), Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumDBEngineC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumDBEngineS"}

El objeto **DBEngine** contiene las siguientes colecciones, métodos y propiedades.

Leyenda:

Característica disponible sólo en espacios de trabajo Microsoft Jet.

Característica disponible sólo en espacios de trabajo ODBCDirect.

Colecciones

Errors
Properties
Workspaces (predeterminado)

Métodos

BeginTrans
CommitTrans

CompactDatabase

CreateDatabase
CreateWorkspace

Idle

OpenConnection
OpenDatabase
RegisterDatabase

RepairDatabase
Rollback

SetOption

Propiedades

DefaultPassword
DefaultType
DefaultUser

IniPath
LoginTimeout

SystemDB
Version

Document (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjDocumentC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjDocumentX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjDocumentP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjDocumentM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjDocumentS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjDocumentU":1}

Un objeto **Document** incluye información sobre una instancia de un tipo de objeto. El objeto puede ser una base de datos, o una tabla, consulta o una relación guardadas (sólo bases de datos Microsoft Jet).

Comentarios

Cada objeto **Container** tiene una colección **Documents** que contiene objetos **Document** que describen instancias de tipos **Container** incorporados. La tabla siguiente muestra el tipo de objeto que describe cada **Document**, el nombre de su objeto **Container** y el tipo de información que contiene **Document**.

Document	Container	Contiene información sobre
Base de datos	Databases	Base de datos contenedora
Tabla o consulta	Tables	Tabla o consulta guardada
Relación	Relations	Formulario guardado

Nota No confunda los tipos de objetos **Container** con los tipos de colección del mismo nombre. Por ejemplo, el objeto **Container** Databases hace referencia a todos los objetos guardados del tipo especificado, pero la colección **Databases** sólo hace referencia a objetos abiertos del tipo especificado.

Con un objeto **Document**, puede:

- Utilizar la propiedad **Name** para devolver el nombre que un usuario o el motor de base de datos Microsoft Jet dieron al objeto cuando se creó.
- Utilizar la propiedad **Container** para devolver el nombre del objeto **Container** que contiene el objeto **Document**.
- Utilizar la propiedad **Owner** para establecer o devolver el propietario del objeto. Para establecer la propiedad **Owner**, debe tener permiso de escritura en el objeto **Document** y debe establecer la propiedad al nombre de un objeto **User** o **Group** existente.
- Utilizar las propiedades **UserName** o **Permissions** para establecer o devolver los permisos de acceso de un usuario o un grupo al objeto. Para establecer estas propiedades, debe tener permiso de escritura en el objeto **Document** y debe establecer la propiedad **UserName** al nombre de un objeto **User** o **Group** existente.
- Utilizar las propiedades **DateCreated** y **LastUpdated** para devolver la fecha y la hora de creación

y última modificación del objeto **Document**.

Debido a que un objeto **Document** corresponde a un objeto existente, no puede crear objetos **Document** nuevos o eliminar los existentes. Para hacer referencia a un objeto **Database** en una colección por su número de orden o por su valor de propiedad **Name**, utilice cualquiera de las siguientes formatos de sintaxis:

Documents(0)
Documents("*nombre*")
Documents![*nombre*]

Documents (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolDocumentC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolDocumentX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolDocumentP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolDocumentM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolDocumentS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolDocumentU":1}

Una colección **Documents** contiene todos los objetos **Document** de un tipo de objeto específico (sólo bases de datos Microsoft Jet).

Comentarios

Cada objeto **Container** tiene una colección **Documents** que contiene objetos **Document** que describen instancias de tipos **Container** incorporados.

Puede hacer referencia a un objeto **Document** de una colección por el valor de su propiedad **Name**, utilizando esta sintaxis:

Documents(0)
Documents("*nombre*")
Documents![*nombre*]

Document (Objeto), Documents (Colección), Resumen

Document (Objeto)

El objeto **Document** contiene esta colección y estas propiedades.

Colección

Properties

Método

CreateProperty

Propiedades

AllPermissions
Container
DateCreated
KeepLocal (definido por el usuario)
LastUpdated
Name
Owner
Permissions
Replicable (definido por el usuario)
UserName

Documents (Colección)

Una colección **Documents** aparece en cada objeto **Container** y contiene este método y esta propiedad.

Método

Refresh

Propiedad

Count

Recordset de tipo Dynaset (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjDynasetTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjDynasetTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjDynasetTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjDynasetTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjDynasetTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjDynasetTypeRecordsetU":1}

Un objeto **Recordset** es un conjunto de registros dinámico que puede contener campos de una o más tablas o consultas en una base de datos y se puede actualizar. En una base de datos ODBCDirect, un objeto **Recordset** de tipo Dynaset corresponde a un cursor de tipo keyset ODBC.

Comentarios

Un objeto **Recordset** de tipo Dynaset es un tipo de objeto **Recordset** que puede utilizar para manipular datos en una tabla o unas tablas de base de datos base.

Se diferencia de un objeto **Recordset** de tipo Snapshot ya que el tipo Dynaset sólo almacena la clave principal de cada registro, en vez de los datos actuales. Como resultado, un tipo Dynaset se actualiza con los cambios realizados en el origen de datos, mientras que el Snapshot no. Como el objeto **Recordset** de tipo Table, un tipo Dynaset recupera el registro completo sólo cuando lo necesita para modificarlo o mostrarlo.

Para crear un objeto **Recordset** de tipo Dynaset, utilice el método **OpenRecordset** en una base de datos abierta, en vez de otro objeto **Recordset** de tipo Snapshot o Dynaset, en un objeto **QueryDef** o en un objeto **TableDef**. (Abrir objetos **Recordset** en otro objeto **Recordset** usuario objetos **TableDef** sólo está disponible en espacios de trabajo Microsoft Jet.)

Si solicita un objeto **Recordset** de tipo Dynaset y el motor de base de datos Microsoft Jet no puede aprovechar el acceso de lectura y escritura a los registros, el motor de base de datos Microsoft Jet puede crear un objeto **Recordset** de tipo Dynaset de sólo lectura.

Cuando los usuarios actualizan datos, las tablas base reflejan estos cambios. Por lo tanto, los datos actuales están disponibles en la aplicación cuando se vuelve a posicionar en el registro actual. En una base de datos multiusuario, más de un usuario puede abrir un objeto **Recordset** de tipo Dynaset que hace referencia a los mismos registros. Debido a que un objeto **Recordset** de tipo Dynaset es dinámico, cuando un usuario cambia usuario registro, los otros usuarios tienen acceso inmediato a los datos cambiados. Sin embargo, si un usuario agrega un registro, los otros usuarios no verán el nuevo registro hasta que utilicen el método **Requery** en el objeto **Recordset**. Si un usuario elimina un registro, se informa a los otros usuarios cuando intentan tener acceso al mismo.

Los registros añadidos a la base de datos no llegan a formar parte del objeto **Recordset** de tipo Dynaset, a menos que los añada utilizando los métodos **AddNew** y **Update**. Por ejemplo, si realiza una consulta de acciones que contenga una instrucción SQL INSERT INTO para añadir registros, los nuevos registros no se incluirán en su objeto **Recordset** de tipo Dynaset hasta que utilice el método **Requery** o reconstruya su objeto **Recordset** con el método **OpenRecordset**.

Para mantener la integridad de los datos, el motor de base de datos Microsoft Jet puede bloquear los objetos **Recordset** de tipo Table y Dynaset durante las operaciones **Edit** (bloqueo pesimista) o **Update** (bloqueo optimista) así que sólo un usuario puede actualizar un registro concreto al mismo tiempo. Cuando el motor de base de datos Microsoft Jet bloquea un registro, bloquea la página entera de 2 KB que contiene el registro.

También puede utilizar el bloqueo optimista y pesimista con tablas no ODBC. Cuando tiene acceso a tablas externas utilizando ODBC mediante un espacio de trabajo Microsoft Jet, siempre debe utilizar el bloqueo optimista. La propiedad **LockEdits** y el parámetro *bloquearmodificaciones* del método **OpenRecordset** determinan las condiciones de bloqueo durante la modificación.

No es posible actualizar todos los campos en todos los objetos **Recordset** de tipo Dynaset. Para determinar si puede actualizar un campo determinado, verifique la propiedad **DataUpdatable** del objeto **Field**.

Un objeto **Recordset** de tipo Dynaset no se puede actualizar si:

- No hay un índice en la tabla o las tablas ODBC o Paradox.
- La página de datos está bloqueada por otro usuario.
- El registro ha cambiado después de la última lectura.
- El usuario no tiene autorización.
- Una o más tablas o campos están definidas como sólo lectura.
- La base de datos se ha abierto como sólo lectura.
- El objeto **Recordset** se ha creado con tablas múltiples sin una instrucción JOIN o la consulta era demasiado compleja.

El orden del objeto **Recordset** de tipo Dynaset o de los datos del objeto **Recordset** no sigue necesariamente una secuencia determinada. Para ordenar los datos, puede utilizarse una instrucción SQL con una cláusula ORDER BY para crear el objeto **Recordset**. También puede utilizar una cláusula WHERE para filtrar los registros para que se agreguen sólo ciertos registros al objeto **Recordset**. Utilizar instrucciones SQL de este modo para seleccionar un subconjunto de registros y ordenarlos normalmente resulta un modo de acceso más rápido a los datos que utilizar las propiedades **Filter** y **Sort**.

Error (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjErrorP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjErrorM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjErrorS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjErrorU":1}

El objeto **Error** contiene detalles sobre los errores de acceso a los datos, cada uno de los cuales está relacionado con una única operación en la que participan objetos de acceso a datos DAO.

Comentarios

Cualquier operación que involucra a DAO puede generar uno o más errores. Por ejemplo, una llamada a un servidor ODBC podría tener como resultado un error en el servidor de la base de datos, un error de ODBC y un error DAO. Como sucede con cada error, se coloca un objeto **Error** en la colección **Errors** del objeto **DBEngine**. Por tanto, un evento sencillo puede tener como resultado que aparezcan varios objetos **Error** en la colección **Errors**.

Cuando otra operación DAO genera un error, la colección **Errors** se borra y el nuevo objeto **Error** se sitúa en la colección **Errors**. Las operaciones DAO que no generan errores, no tienen efectos sobre la colección **Errors**.

El conjunto de los objetos **Error** en la colección **Errors** describe un error. El primer objeto **Error** es el error de nivel más bajo, el segundo es del nivel inmediatamente superior, etc. Por ejemplo, si se produce un error ODBC mientras se intenta abrir un objeto **Recordset**, el último objeto **Error**; **Errors**(0) - contendrá el error DAO, que indicará que el objeto no ha podido abrirse. El primer objeto error contiene el nivel más bajo de error ODBC, los siguientes errores contienen los errores enviados por las diferentes capas del ODBC. En este caso, el administrador del controlador, y posiblemente el mismo controlador, devolverán objetos **Error** separados. El último objeto **Error** - **Errors.Count** -1 - contiene el error DAO que indica que no se puede abrir el objeto.

Enumerar los errores específicos en la colección **Errors** habilita las rutinas de manejo de errores para determinar de una forma más exacta la causa y el origen de un error y para así crear los pasos adecuados de recuperación. En los espacios de trabajo Microsoft Jet y ODBCDirect, puede leer las propiedades del objeto **Error** para obtener detalles específicos acerca de cada error, incluyendo:

- La propiedad **Description**, que contiene el texto de la alerta de error que se mostrará en la pantalla si no se captura el error.
- La propiedad **Number**, que contiene una variable de tipo **Long** entera con el valor de la constante de error.
- La propiedad **Source**, que identifica el objeto que activó el error. Esto es particularmente útil cuando tiene varios objetos **Error** en la colección **Errors** que siguen a una petición a un origen de datos ODBC.
- Las propiedades **HelpFile** y **HelpContext**, que indican el archivo de Ayuda de Microsoft Windows y el tema de Ayuda adecuados, respectivamente, (si existen) para el error.

Nota Cuando programa en Microsoft Visual Basic for Applications (VBA), si utiliza la palabra clave **New** para crear un objeto que a continuación produce un error antes de que se haya agregado un objeto a la colección, la colección **Errors** del objeto **DBEngine** no contendrá ninguna entrada para ese error de objeto, porque el nuevo objeto no está asociado con el objeto **DBEngine**. Sin embargo, está disponible la información de error en el objeto **Err** de VBA.

El código de manejo de errores de VBA debe examinar la colección **Errors** cada vez que prevea un error de acceso a datos. Si escribe un controlador de error centralizado, pruebe el objeto **Err** de VBA para determinar si es válida la información de error en la colección **Errors**. Si coinciden la propiedad **Number** del último elemento de la colección **Errors** (`DBEngine.Errors.Count - 1`) y el valor del objeto **Err**, entonces puede utilizar una serie de instrucciones **Select Case** para identificar el error o los errores DAO particulares que se producen. Si no coinciden, utilice el método **Refresh** en la colección **Errors**.

Errors (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolErrorP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolErrorM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolErrorS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolErrorU":1}

La colección **Errors** contiene todos los objetos **Error** almacenados, cada uno de los cuales se refiere a una única operación en la que participan objetos de acceso a datos DAO.

Comentarios

Cualquier operación que involucra a DAO puede generar uno o más errores. Como sucede con cada error, se colocan uno o más objetos **Error** en la colección **Errors** del objeto **DBEngine**. Cuando otra operación DAO genera un error, la colección **Errors** se borra el nuevo objeto **Error** se sitúa en la colección **Errors**. El objeto de número más alto en la colección **Errors** (`DBEngine.Errors.Count - 1`) corresponde al error generado por el objeto **Err** de Microsoft Visual Basic for Applications (VBA).

Las operaciones DAO que no generan un error no tienen efecto en la colección **Errors**.

Los elementos de la colección **Errors** no se añaden del mismo modo que las demás colecciones, así que la colección **Errors** no admite los métodos **Append** y **Delete**.

El conjunto de objetos **Error** de la colección **Errors** describe un error. El primer objeto **Error** es el error de nivel más bajo, el segundo es del nivel inmediatamente superior, etc. Por ejemplo, si se produce un error ODBC mientras se intenta abrir un objeto **Recordset**, el último objeto error contiene el nivel más bajo de error ODBC, los siguientes errores contienen los errores enviados por las diferentes capas del ODBC. En este caso, el administrador del controlador, y posiblemente el mismo controlador, devolverán objetos **Error** separados. El último objeto **Error** contiene el error DAO que indica que no se puede abrir el objeto.

Enumerar los errores específicos en la colección **Errors** habilita las rutinas de manejo de errores para determinar de una forma más exacta la causa y el origen de un error y para así crear los pasos adecuados de recuperación.

Nota Si utiliza la palabra clave **New** para crear un objeto que produzca un error antes que ese objeto se haya añadido a la colección **Errors**, la colección no presentará una entrada para ese error de objeto porque el nuevo objeto no está asociado con el objeto **DBEngine**. La información de error se encuentra disponible en el objeto **Err** de Visual Basic.

Error (Objeto), Errors (Colección) Resumen

Error (Objeto)

Un objeto **Error** no contiene métodos ni colecciones y sí estas propiedades:

Propiedades
Description
HelpContext
HelpFile
Number
Source

Errors (Colección)

La colección **Errors** aparece en el objeto **DBEngine**, y contiene este método y esta propiedad:

Método
Refresh

Propiedad
Count

Field (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjFieldP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjFieldM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjFieldS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjFieldU":1}

Un objeto **Field** representa una columna de datos con un tipo de datos y una serie de propiedades comunes.

Comentarios

Las colecciones **Fields** de objetos **Index**, **QueryDef**, **Relation** y **TableDef** contienen las especificaciones de los campos a los que representan estos objetos. La colección **Fields** de un objeto **Recordset** representa los objetos **Field** en una fila de datos o en un registro. Los objetos **Field** de un objeto **Recordset** se utilizan para leer y establecer valores para los campos del registro activo del objeto **Recordset**.

Para manipular un campo en los espacios de trabajo Microsoft Jet y ODBCDirect, utilice un objeto **Field** y sus métodos y propiedades. Por ejemplo, puede hacer lo siguiente:

- Utilice la propiedad **OrdinalPosition** para establecer o devolver el orden de presentación del objeto **Field** en una colección **Fields**. (Esta propiedad es de sólo lectura para las bases de datos ODBCDirect.)
- Utilice la propiedad **Value** de un campo en un objeto **Recordset** para establecer o devolver datos almacenados.
- Utilice los métodos **AppendChunk** y **GetChunk** y la propiedad **FieldSize** para obtener o establecer un valor en un Objeto OLE o un campo de tipo Memo de un objeto **Recordset**.
- Utilice la propiedades **Type**, **Size** y **Attributes** para determinar el tipo de datos que se puede almacenar en el campo.
- Utilice la propiedades **SourceField** y **SourceTable** para determinar el origen de datos original.

En espacios de trabajo Microsoft Jet, puede:

- Utilice la propiedad **ForeignName** para establecer o devolver información acerca de un campo externo en un objeto **Relation**.
- Utilice las propiedades **AllowZeroLength**, **DefaultValue**, **Required**, **ValidateOnSet**, **ValidationRule** o **ValidationText** para establecer o devolver condiciones de validación.
- Utilice la propiedad **DefaultValue** de un campo en un objeto **TableDef** para establecer el valor predeterminado para este campo cuando se agregan nuevos registros.

En espacios de trabajo ODBCDirect, puede:

- Utilizar las propiedades **Value**, **VisibleValue** y **OriginalValue** para comprobar la finalización con éxito de una actualización por lotes.

Nota Para obtener una lista completa de todos los métodos, propiedades y colecciones disponibles en un objeto **Field** en cualquier base de datos o conexión, vea el tema Resumen.

Para crear un nuevo objeto **Field** en un objeto **Index**, **TableDef** o **Relation** se puede utilizar el método **CreateField**.

Cuando se accede al objeto **Field** como parte de un objeto **Recordset**, los datos del registro activo son visibles en la propiedad **Value** del objeto **Field**. Para manipular datos en el objeto **Recordset** no se suele hacer referencia directa a la colección **Fields**; en su lugar se hace referencia indirectamente a la propiedad **Value** del objeto **Field**, en la colección **Fields** del objeto **Recordset**.

Para hacer referencia a un objeto **Field** en una colección por su número de orden o por su valor de propiedad **Name**, utilice los formatos de sintaxis siguientes:

Fields(0)
Fields("*nombre*")
Fields![*nombre*]

Con los mismos formatos de sintaxis, también puede hacer referencia a la propiedad **Value** de un objeto **Field** que crea y agrega a una colección **Fields**. El contexto de la referencia de campo determinará si hace referencia a un objeto **Field** o a la propiedad **Value** del objeto **Field**.

Fields (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolFieldP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolFieldM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolFieldS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolFieldU":1}

Una colección **Fields** contiene todos los objetos **Field** almacenados de un objeto **Index**, **QueryDef**, **Recordset**, **Relation** o **TableDef**.

Comentarios

Las colecciones **Fields** de objetos **Index**, **QueryDef**, **Relation** y **TableDef** contienen las especificaciones de los campos a los que representan estos objetos. La colección **Fields** de un objeto **Recordset** representa los objetos **Field** en una fila de datos, o en un registro. Los objetos **Field** de un objeto **Recordset** se utilizan para leer y establecer valores para los campos del registro activo del objeto **Recordset**.

Para hacer referencia a un objeto **Field** en una colección por su número de orden o por su valor de propiedad **Name**, utilice los formatos de sintaxis siguientes:

Fields(0)
Fields("*nombre*")
Fields![*nombre*]

Con los mismos formatos de sintaxis, también puede hacer referencia a la propiedad **Value** de un objeto **Field** que crea y agrega a una colección **Fields**. El contexto de la referencia de campo determinará si hace referencia a un objeto **Field** o a la propiedad **Value** del objeto **Field**.

Field (Objeto), Fields (Colección) Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumFieldS"}

Field (Objeto)

Un objeto **Field** contiene estas colecciones, métodos y propiedades.

Leyenda:

Característica disponible sólo en el área de trabajo de Microsoft Jet.

Carácterística disponible sólo en el área de trabajo ODBCDirect.

Colección
Properties

Métodos

La siguiente tabla enumera todos los método del objeto **Field**. El tipo de objeto cuya colección **Fields** contiene el objeto **Field** determina qué métodos están disponibles.

Método	Index	QueryDef	Recordset	Relation	TableDef
AppendChunk			✔		
CreateProperty					
GetChunk					

Propiedades

La siguiente tabla enumera todas las propiedades del objeto **Field**. El tipo de objeto cuya colección **Fields** contiene el objeto **Field** determina qué propiedades están disponibles. Todas las propiedades son de sólo lectura para los objetos **Field** agregados a las colecciones **Fields** de los objetos **Index**, **Relation** y **TableDef**.

Sólo lectura
Lectura/Escritura

Propiedad	Index	QueryDef	Recordset	Relation	TableDef
AllowZeroLength					
Attributes					
CollatingOrder					
DataUpdatable					
DefaultValue					
FieldSize					

ForeignName					
Name					
OrdinalPosition					
OriginalValue					
		*			
Required					
Size					
SourceField					
SourceTable					
Type					
ValidateOnSet					
ValidationRule					
ValidationText					
Value					
VisibleValue					
		*			

* Estas propiedades sólo están disponibles en un espacio de trabajo ODBCDirect cuya propiedad **DefaultCursorDriver** se es **dbUseClientBatchCursor.**

Fields (Colección)

Una colección **Fields** aparece en cada uno de los objetos **TableDef**, **QueryDef**, **Recordset**, **Relation** e **Index**, y contiene estos métodos y propiedades.

Método	Index	QueryDef	Recordset	Relation	TableDef
Append					
Delete					
Refresh					
Propiedad	**Index**	**QueryDef**	**Recordset**	**Relation**	**TableDef**

Count

Group (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjGroupX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjGroupP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjGroupM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjGroupS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjGroupU":1}

Un objeto **Group** representa un grupo de cuentas de usuario que tiene autorizaciones de acceso comunes cuando un objeto **Workspace** actúa como un grupo de trabajo asegurado. (sólo espacios de trabajo Microsoft Jet).

Comentarios

Una vez creados los objetos **Group** se utilizan sus nombres para establecer y reforzar las autorizaciones de acceso a las bases de datos, tablas, y consultas, mediante los objetos **Document** que representan los objetos **Database**, **TableDef** y **QueryDef** con los que está trabajando.

Utilizando las propiedades de un objeto **Group**, puede:

- Utilice la propiedad **Name** de un objeto **Group** existente para devolver su nombre. No puede devolver el valor de la propiedad **PID** de un objeto **Group** existente.
- Utilice las propiedades **Name** y **PID** de un objeto **Group** creado recientemente y sin agregados para establecer la identidad de ese objeto **Group**.

Puede añadir un objeto **Group** existente a la colección **Groups** de un objeto **User** para establecer la pertenencia de la cuenta de usuario a ese objeto **Group**. Como alternativa, también puede añadir el objeto **User** a la colección **Users** del objeto **Group** para dotar a la cuenta de usuario de las autorizaciones globales de este grupo. Si utiliza una colección **Groups** o **Users** distinta de aquella a la que acaba de añadir el objeto, es posible que necesite utilizar el método **Refresh** para actualizar la colección con información actual de la base de datos.

El motor de base de datos Microsoft Jet define de modo predeterminado tres objetos **Group** llamados Administradores, Usuarios, e Invitados. Para crear un objeto **Group** nuevo, utilice el método **CreateGroup** en un objeto **User** o **Workspace**.

Para hacer referencia a un objeto **Group** en una colección por su número de orden o por el valor de su propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

Groups(0)
Groups("*nombre*")
Groups![*nombre*]

Groups (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolGroupX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolGroupP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolGroupM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolGroupS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolGroupU":1}

Una colección **Groups** contiene todos los objetos **Group** de un objeto **Workspace** o de una cuenta de usuario (sólo espacios de trabajo Microsoft Jet).

Comentarios

Se puede añadir un objeto **Group** existente a la colección **Groups** de un objeto **User** para establecer la pertenencia de la de cuenta de usuario a ese objeto **Group**. Como alternativa, puede añadir un objeto **User** a la colección **Users** de un objeto **Group** para dar a la cuenta de usuario la autorización global de ese grupo. En cada caso, el objeto **Group** existente debe ser ya un miembro de la colección **Groups** del objeto **Workspace** actual. Si utiliza una colección **Groups** o **Users** diferente de aquella a la que acaba de añadir el objeto, es posible que sea necesario utilizar el método **Refresh** para actualizar la colección con información actual sobre la base de datos.

Para hacer referencia a un objeto **Group** en una colección por su número de orden o por su valor de propiedad **Name**, utilice los formatos de sintaxis siguientes:

Groups(0)
Groups("*nombre*")
Groups![*nombre*]

Group (Objeto), Groups (Colección) Resumen

Group (Objeto)

El objeto **Group** contiene las colecciones, métodos, y propiedades siguientes.

Colecciones

Properties
Users (predeterminado)

Método

CreateUser

Propiedades

Name
PID

Groups (Colección)

La colección **Groups** aparece en cada objeto **User** y en cada objeto **Workspace** Microsoft Jet y contiene los siguientes métodos y propiedades.

Métodos

Append
Delete
Refresh

Propiedad

Count

Index (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjIndexP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjIndexM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjIndexS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjIndexU":1}

Los objetos **Index** especifican el orden de los registros a los que se accede desde las tablas de la base de datos y determinan si se aceptan o no registros duplicados, permitiendo un acceso eficiente a los datos. En las bases de datos externas, los objetos **Index** describen los índices establecidos para tablas externas (sólo espacios de trabajo Microsoft Jet).

Comentarios

El motor de base de datos Microsoft Jet utiliza índices cuando se une con tablas y crea objetos **Recordset**. Los índice determinan el orden en que los objetos **Recordset** de tipo Table devuelven registros, pero no determinan el orden en el que el motor de base de datos Microsoft Jet los almacena en la tabla base o el orden en que cualquier otro tipo de objeto **Recordset** los devuelve.

Si utiliza un objeto **Index**, puede:

- Utilizar la propiedad **Required** para determinar si los objetos **Field** del índice necesitan valores no **Null** y utilizar la propiedad **IgnoreNulls** para determinar si los valores **Null** tienen entradas de índice.
- Utilizar las propiedades **Primary** y **Unique** para determinar el orden y exclusividad del objeto **Index**.

El motor de base de datos Microsoft Jet mantiene todos los índices de la tabla base automáticamente. Los índices se actualizan siempre que se añadan, cambian, o eliminan registros de la tabla base. Una vez que se ha creado la base de datos, se puede utilizar el método **CompactDatabase** periódicamente para actualizar las estadísticas del índice.

Al acceder a un objeto **Recordset** de tipo tabla, se puede determinar el orden de los registros mediante la propiedad **Index**. Se puede establecer esta propiedad para el valor de la propiedad **Name** de un objeto **Index** existente en la colección **Indexes**. Esta colección está contenida en el objeto **TableDef**, que subyace al objeto **Recordset** que está rellenando.

Nota No es necesario crear índices para una tabla, pero en las tablas grandes y sin índice, el acceso a un registro específico o el proceso de uniones puede tardar mucho tiempo. Por el contrario,

si se crean demasiados índices, se puede hacer más lenta la actualización de la base de datos, ya que hay que modificar todos los índices de la tabla.

La propiedad **Attributes** de cada objeto **Field** del índice establece el orden de los registros y por ello determina las técnicas de acceso que se deben utilizar para ese índice.

Cada objeto **Field** de la colección **Fields** de un objeto **Index** es un componente del índice. Para definir un nuevo objeto **Index** es necesario establecer las propiedades antes de añadirlo a una colección, lo que permite que el objeto **Index** esté disponible para su uso posterior.

Nota Puede modificar el valor de la propiedad **Name** de un objeto existente **Index** sólo si la propiedad **Updatable** del objeto **TableDef** que la contiene es **True**.

Cuando establezca una clave principal para una tabla, el motor de base de datos Microsoft Jet la definirá como el índice principal. Un índice principal consta de uno o más campos que identifican exclusivamente todos los registros de una tabla en un orden predefinido. Como el índice principal del campo debe ser único, el motor de base de datos Microsoft Jet establece automáticamente como **True** la propiedad **Unique** del objeto **Index**. Si el índice principal consta de más de un campo, cada campo puede contener valores duplicados, pero la combinación de valores de todos los campos indexados debe ser única. Un índice principal consta de una clave para la tabla y siempre está formado de los mismos campos que la clave principal.

Importante Compruebe que los datos coincidan con los atributos del nuevo índice. Si el índice requiere valores exclusivos, compruebe que no existan duplicados en los registros de datos existentes. Si existen duplicados, el motor de base de datos Microsoft Jet no podrá crear el índice y se producirá un error cuando intente utilizar el método **Append** con el nuevo índice.

Cuando cree una relación que exija la integridad referencial, el motor de base de datos Microsoft Jet creará automáticamente un índice con la propiedad **Foreign**, establecida como clave externa de la tabla de referencia. Una vez establecida una relación de tabla, el motor de base de datos Microsoft Jet evita adiciones o cambios en la base de datos que violen esa relación. Si establece la propiedad **Attributes** del objeto **Relation** para permitir operaciones de actualización en cascada y eliminación en cascada, el motor de base de datos Microsoft Jet actualiza o elimina automáticamente los registros de las tablas relacionadas.

Para crear un nuevo objeto Index

1. Utilice el método **CreateIndex** con el objeto **TableDef**.
2. Utilice el método **CreateField** con el objeto **Index** para crear un objeto **Field** para cada campo (columna) que se vaya a incluir en el objeto **Index**.
3. Seleccione las propiedades **Index** según sea necesario.
4. Añada el objeto **Field** a la colección **Fields**.
5. Añada el objeto **Index** a la colección **Indexes**.

Nota La propiedad **Clustered** se pasa por alto para las bases de datos que utilizan el motor de base de datos Microsoft Jet, ya que no admite índices agrupados.

Indexes (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolIndexP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolIndexM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolIndexS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolIndexU":1}

Una colección **Indexes** contiene todos los objetos **Index** almacenados de un objeto **TableDef** (sólo espacios de trabajo Microsoft Jet).

Comentarios

Cuando tiene acceso a un objeto **Recordset** de tipo Table, utilice la propiedad **Index** del objeto para especificar el orden de los registros. Establezca esta propiedad al valor de la propiedad **Name** de un objeto **Index** existente en la colección **Indexes** del objeto **TableDef** bajo el objeto **Recordset**.

Nota Puede utilizar el método **Append** o **Delete** en una colección **Indexes**, sólo si el valor de la propiedad **Updatable** del objeto **TableDef** es **True**.

Una vez creado, el nuevo objeto **Index** debe añadirse a la colección **Indexes** del objeto **TableDef**, utilizando el método **Append**.

Importante Asegúrese cumple con los atributos del nuevo índice. Si el índice requiere valores únicos, asegúrese que no existen registros con datos duplicados. Si hay duplicados, el motor de base de datos Microsoft Jet no puede crear el índice; se produce un error interceptable cuando intenta utilizar el método **Append** en el nuevo índice.

Index (Objeto), Indexes (Colección) Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumIndexS"}

Index (Objeto)

El objeto **Index** contiene las siguientes colecciones, métodos y propiedades.

Colecciones

Fields (predeterminada)
Properties

Métodos

CreateField
CreateProperty

Propiedades

Clustered
DistinctCount
Foreign
IgnoreNulls
Name
Primary
Required
Unique

Indexes (Colección)

Una colección **Indexes** aparece en cada objeto **TableDef**, y contiene los siguientes métodos y propiedades.

Métodos

Append
Delete
Refresh

Propiedad

Count

Parameter (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjParameterP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjParameterM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjParameterS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjParameterU":1}

Un objeto **Parameter** representa un valor suministrado a una consulta. El parámetro está asociado a un objeto **QueryDef** creado a partir de una consulta con parámetros.

Parameters

Comentarios

Los objetos **Parameter** le permiten cambiar los argumentos en un objeto **QueryDef** que se ejecuta frecuentemente si tener que volver a compilar la consulta.

Utilizando las propiedades de un objeto **Parameter**, puede establecer un parámetro de consulta que puede cambiarse antes de ejecutar la consulta. Puede hacer lo siguiente:

- Utilice la propiedad **Name** para devolver el nombre de un parámetro.
- Utilice la propiedad **Value** para establecer o devolver los valores del parámetro que se va a utilizar en la consulta.
- Utilice la propiedad **Type** para devolver el tipo de datos del objeto **Parameter**.
- Utilice la propiedad **Direction** para establecer o devolver si el parámetro es un parámetro de entrada, un parámetro de salida o ambos.

En un espacio de trabajo ODBCDirect, también puede:

- cambiar el valor de la propiedad **Type**.
- Utilizar la propiedad **Direction** para establecer o devolver si el parámetro es un parámetro de entrada, , un parámetro de salida o ambos.

Parameters (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolParameterP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolParameterM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolParameterS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolParameterU":1}

Una colección **Parameters** contiene todos los objetos **Parameter** de un objeto **QueryDef**.

Comentarios

Una colección **Parameters** sólo proporciona información sobre los parámetros existentes. No puede añadir ni eliminar objetos de la colección **Parameters**.

Parameter (Objeto), Parameters (Colección) Resumen

Parameter (Objeto)

Un objeto **Parameter** no contiene métodos; contiene esta colección y estas propiedades.

Leyenda:

Característica disponible sólo en áreas de trabajo ODBCDirect.

Colección

Properties

Propiedades

Direction
Name
Type
Value (predeterminada)

Parameters (Colección)

Una colección **Parameters** aparece en cada objeto **QueryDef** y contiene este método y esta propiedad.

Método

Refresh

Propiedad

Count

Property (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjPropertyP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjPropertyM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjPropertyU":1}

Un objeto **Property** representa una característica incorporada o una característica definida por el usuario de un objeto de acceso a datos (DAO).

Comentarios

Todos los objetos de acceso de datos contienen una colección **Properties** que tiene algunos objetos **Property** incorporados. El usuario también puede definir los objetos **Property** y agregarlos a la colección **Properties** de algunos objetos DAO. Estos objetos **Property** (que con frecuencia sólo se llaman propiedades) únicamente caracterizan esa instancia del objeto.

Puede crear y agregar sus propias propiedades definidas por el usuario a estos objetos:

- Objetos **Database**, **Index**, **QueryDef** y **TableDef**
- Objetos **Field** en colecciones **Fields** de objetos **QueryDef** y **TableDef**

Para agregar una propiedad definida por el usuario, utilice el método **CreateProperty** para crear un objeto **Property** con un valor único para la propiedad **Name**. Establezca las propiedades **Type** y **Value** del nuevo objeto **Property** y, a continuación, añádalo a la colección **Properties** del objeto correspondiente. El objeto al que agregue la propiedad definida por el usuario debe estar guardado en el disco (es decir, debe añadirse a una colección). Hacer referencia a un objeto **Property** definido por el usuario que no se ha agregado a la colección **Properties** producirá un error, como agregar un objeto **Property** definido por el usuario a una colección **Properties** que contiene un objeto **Property** con el mismo nombre.

Puede eliminar propiedades definidas por el usuario de la colección **Properties**, pero no puede eliminar las propiedades incorporadas.

Nota Una propiedad definida por el usuario (objeto **Property**) sólo está asociada a la instancia específica del objeto. La propiedad no se define para todas las instancias de objetos del tipo seleccionado.

Se puede utilizar la colección **Properties** de un objeto para enumerar las propiedades incorporadas y definidas por el usuario del objeto. No es necesario que conozca de antemano exactamente qué propiedades existen o cuáles son sus características (propiedades **Name** y **Type**) para manipularlas. Sin embargo, si intenta leer una propiedad de sólo escritura (como la propiedad **Password** de un objeto **Workspace**) o intenta leer o escribir una propiedad en un contexto adecuado, como el valor de la propiedad **Value** de un objeto **Field** en la colección **Fields** de un objeto **TableDef**, se produce un error.

El objeto **Property** también tiene cuatro propiedades incorporadas:

- La propiedad **Name**, un valor de tipo **String** que identifica únicamente la propiedad.
- La propiedad **Type**, un valor de tipo **Integer** que especifica el tipo de datos de la propiedad.
- La propiedad **Value**, un valor de tipo **Variant** que contiene el valor de la propiedad.
- La propiedad **Inherited**, un valor de tipo **Boolean** que indica si la propiedad se hereda de otro objeto. Por ejemplo, un objeto **Field** en una colección **Fields** de un objeto **Recordset** puede heredar propiedades del objeto **TableDef** o **QueryDef** base.

Para hacer referencia a un objeto **Property** incorporado en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

objeto.**Properties**(0)
objeto.**Properties**("*nombre*")
objeto.**Properties**![*nombre*]

Para una propiedad incorporada, también puede utilizar esta sintaxis:

objeto.nombre

Nota Para una propiedad definido por el usuario, debe utilizar la sintaxis *objeto*.**Properties**("*nombre*") completa.

Con los mismos formatos de sintaxis, también puede hacer referencia a la propiedad **Value** de un objeto **Property**. El contexto de la referencia determinará si hace referencia al objeto **Property** por si mismo o a la propiedad **Value** del objeto **Property**.

Properties (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolPropertyP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolPropertyM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolPropertyU":1}

Una colección **Properties** contiene todos los objetos **Property** de una instancia determinada de un objeto.

Comentarios

Todos los objetos de acceso de datos contienen una colección **Properties**, que tiene algunos objetos **Property**. Estos objetos **Property** (que a veces se denominan propiedades) caracterizan de forma exclusiva a esa instancia del objeto.

Además de las propiedades incorporadas, algunos objetos permiten crear y añadir propiedades definidas por el usuario. Para añadir una propiedad definida por el usuario a una instancia existente de un objeto, es necesario definir previamente sus características con el método **CreateProperty** y a continuación añadirla a la colección con el método **Append**. Hacer referencia a un objeto **Property** definido por el usuario que no se ha agregado a la colección **Properties** producirá un error, como agregar un objeto **Property** definido por el usuario a una colección **Properties** que contiene un objeto **Property** con el mismo nombre.

Puede utilizar el método **Delete** para eliminar propiedades definidas por el usuario de la colección **Properties**, pero no puede eliminar las propiedades incorporadas.

Nota Una propiedad definida por el usuario (objeto **Property**) sólo está asociada a la instancia específica del objeto. La propiedad no se define para todas las instancias de objetos del tipo seleccionado.

Se puede utilizar la colección **Properties** de un objeto para enumerar las propiedades incorporadas y definidas por el usuario del objeto. No es necesario que conozca de antemano exactamente qué propiedades existen o cuáles son sus características (propiedades **Name** y **Type**) para manipularlas. Sin embargo, si intenta leer una propiedad de sólo escritura, como la propiedad **Password** de un objeto **Workspace**) o intenta leer o escribir una propiedad en un contexto adecuado, como el valor de la propiedad **Value** de un objeto **Field** en la colección **Fields** de un objeto **TableDef**, se produce un error.

Para hacer referencia a un objeto **Property** incorporado en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

objeto.**Properties**(0)
objeto.**Properties**("*nombre*")
objeto.**Properties**![*nombre*]

Para una propiedad incorporada, también puede utilizar esta sintaxis:

objeto.*nombre*

Nota Para una propiedad definido por el usuario, debe utilizar la sintaxis *objeto*.**Properties**("*nombre*") completa.

Con los mismos formatos de sintaxis, también puede hacer referencia a la propiedad **Value** de un objeto **Property**. El contexto de la referencia determinará si hace referencia al objeto **Property** por si mismo o a la propiedad **Value** del objeto **Property**.

Property (Objeto), Properties (Colección) Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumPropertyS"}

Property (Objeto)

El objeto **Property** no contiene métodos, pero contiene esta colección y estas propiedades:

Colección

Properties

Propiedades

Inherited (Siempre **False** en bases de datos ODBCDirect)

Name

Type

Value

Properties (Colección)

Una colección **Properties** aparece en cada uno de los demás objetos de acceso de datos y contiene los siguientes métodos y propiedades.

Métodos

Append

Delete

Refresh

Propiedad

Count

QueryDef (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjQueryDefP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjQueryDefM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjQueryDefS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjQueryDefU":1}

Un objeto **QueryDef** es una definición almacenada de una consulta en una base de datos Microsoft Jet o una definición temporal de una consulta en un espacio de trabajo ODBCDirect.

Comentarios

Puede utilizar el objeto **QueryDef** para definir una consulta. Por ejemplo, puede:

- Utilice la propiedad **SQL** para establecer o devolver la definición de la consulta.
- Utilice la colección **Parameters** del objeto **QueryDef** para establecer o devolver los parámetros de

la consulta.

- Utilice la propiedad **Type** para devolver un valor que indica si la consulta selecciona registros de una tabla existente, crea una tabla nueva, inserta registros de una tabla en otra tabla, elimina registros o los actualiza.
- Utilice la propiedad **MaxRecords** para limitar el número de registros devueltos por una consulta.
- Utilice la propiedad **ODBCTimeout** para indicar el tiempo de espera antes de que una consulta devuelva registros. La propiedad **ODBCTimeout** se aplica a cualquier consulta que tiene acceso a datos ODBC.

En un espacio de trabajo Microsoft Jet, también puede:

- Utilizar la propiedad **ReturnsRecords** para indicar que la consulta devuelve registros. La propiedad **ReturnsRecords** sólo es válida en las consultas SQL de paso a través.
- Utilizar la propiedad **Connect** para hacer una consulta SQL de paso a través en una base de datos ODBC.

En un espacio de trabajo ODBCDirect, también puede:

- Utilizar la propiedad **Prepare** para determinar si llamar a la API ODBC **SQLPrepare** cuando se ejecuta la consulta.
- Utilizar la propiedad **CacheSize** para almacenar en memoria caché los registros devueltos por una consulta.

También puede crear objetos **QueryDef** temporales. A diferencia de los objetos **QueryDef** permanentes, los objetos **QueryDef** temporales no se guardan en disco ni se agregan a la colección **QueryDefs**. Los objetos **QueryDef** temporales son útiles para consultas que debe ejecutar repetidamente durante el tiempo de ejecución, pero no necesita guardar en disco, particularmente si crea las instrucciones SQL durante el tiempo de ejecución .

Puede pensar en un objeto **QueryDef** permanente en espacios de trabajo Microsoft Jet como una instrucción SQL compilada. Si ejecuta una consulta desde un objeto **QueryDef** permanente, la consulta se ejecutará más rápido que si ejecuta la instrucción SQL equivalente desde el método **OpenRecordset**. Esto es debido a que el motor de base de datos Microsoft Jet no necesita compilar la consulta antes de ejecutarla.

La vía preferente para utilizar el dialecto original SQL de un motor de base de datos externo es a través del motor de base de datos Microsoft Jet y a través de objetos **QueryDef**. Por ejemplo, se puede crear una consulta SQL de transacción y almacenarla en un objeto **QueryDef**. Cuando necesite utilizar una consulta SQL de un motor de base de datos diferente a Microsoft Jet, tendrá que incorporar una cadena de propiedad **Connect** que señale hacia el origen de datos externo. Las consultas con propiedades **Connect** válidas evitan el motor de base de datos Microsoft Jet y pasan la consulta directamente al servidor de la base de datos externa para su procesamiento.

Para crear un nuevo objeto **QueryDef**, utilice el método **CreateQueryDef**. En un espacio de trabajo Microsoft Jet, si proporciona una cadena para el argumento *nombre* o si establece la propiedad **Name** del nuevo objeto **QueryDef** explícitamente no a una cadena de longitud cero, creará un objeto **QueryDef** permanente que se agregará automáticamente a la colección **QueryDefs** y guardado en disco. Proporcionar una cadena de longitud cero como el argumento *nombre* o establecer la propiedad **Name** explícitamente a una cadena de longitud cero dará como resultado un objeto **QueryDef** temporal.

En un espacio de trabajo ODBCDirect, un objeto **QueryDef** siempre es temporal. La colección **QueryDefs** contiene todos los objetos **QueryDef** abiertos. Cuando se cierra un objeto **QueryDef**, se elimina automáticamente de la colección **QueryDefs**.

Para hacer referencia a un objeto **QueryDef** incorporado en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes

QueryDefs(0)
QueryDefs("*nombre*")

QueryDefs![*nombre*]

Puede hacer referencia a objetos **QueryDef** temporales sólo por las variables de objeto que les haya asignado.

QueryDefs (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolQueryDefP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolQueryDefM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolQueryDefS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolQueryDefU":1}

Una colección **QueryDefs** contiene todos los objetos **QueryDef** de un objeto **Database** en una base de datos Microsoft Jet y todos los objetos **QueryDef** de un objeto **Connection** en un espacio de trabajo ODBCDirect.

Comentarios

Para crear un nuevo objeto **QueryDef**, utilice el método **CreateQueryDef**. En un espacio de trabajo Microsoft Jet, si proporciona una cadena para el argumento *nombre* o si establece la propiedad **Name** del nuevo objeto **QueryDef** explícitamente no a una cadena de longitud cero, creará un objeto

QueryDef permanente que se agregará automáticamente a la colección **QueryDefs** y guardado en disco. Proporcionar una cadena de longitud cero como el argumento *nombre* o establecer la propiedad **Name** explícitamente a una cadena de longitud cero dará como resultado un objeto **QueryDef** temporal.

En un espacio de trabajo ODBCDirect, un objeto **QueryDef** siempre es temporal. La colección **QueryDefs** contiene todos los objetos **QueryDef** abiertos. Cuando se cierra un objeto **QueryDef**, se elimina automáticamente de la colección **QueryDefs**.

Para hacer referencia a un objeto **QueryDef** incorporado en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

QueryDefs(0)
QueryDefs("*nombre*")
QueryDefs![*nombre*]

Puede hacer referencia a objetos **QueryDef** temporales sólo por las variables de objeto que les haya asignado.

QueryDef (Objeto), QueryDefs (Colección) Resumen

QueryDef (Objeto)

El objeto **QueryDef** contiene las siguientes colecciones, métodos y propiedades.

Leyenda:

Disponible sólo en un espacio de trabajo Microsoft Jet.

Disponible sólo en un espacio de trabajo ODBCDirect.

Colecciones

Fields
Parameters (predeterminada)
Properties

Métodos

Cancel
Close

CreateProperty
Execute
OpenRecordset

Propiedades

CacheSize
Connect

DateCreated

KeepLocal

LastUpdated

LogMessages
MaxRecords
Name
ODBCTimeout

Prepare
RecordsAffected

Replicable

ReturnsRecords
SQL

StillExecuting

Type

Updatable

QueryDefs (Colección)

Una colección **QueryDefs** que aparece en cada objeto **Connection** en un espacio de trabajo ODBCDirect y en cada objeto **Database** y contiene estos métodos y esta propiedad .

Métodos

Append

Delete

Refresh

Propiedad

Count

Recordset (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjRecordsetU":1}

Un objeto **Recordset** representa los registros de una tabla base o los registros que se generan al ejecutar una consulta.

Comentarios

Utilice los objetos **Recordset** para manipular datos en una base de datos a nivel de registro. Cuando utiliza objetos de acceso de datos, interactúa con los datos prácticamente utilizando objetos **Recordset**. Todos los objetos **Recordset** se construyen utilizando registros (filas) y campos (columnas). Existen tres tipos de objetos **Recordset**:

- **Recordset de tipo Table** - una representación en código de una tabla base que puede utilizarse para añadir, cambiar o eliminar registros desde una única tabla de base de datos (sólo espacios de trabajo Microsoft Jet).
- **Recordset de tipo Dynaset** - el resultado de una consulta cuyos registros pueden actualizarse. Un objeto **Recordset** de tipo Dynaset es un conjunto dinámico de registros que puede utilizarse para añadir, cambiar o eliminar registros desde una tabla o tablas subyacentes de una base de datos. Un objeto **Recordset** de tipo Dynaset puede contener campos de una o más tablas de una base de datos. Este tipo corresponde a un cursor de tipo keyset ODBC.

- **Recordset de tipo Snapshot** - una copia estática de un conjunto de registros que puede utilizar para encontrar datos o generar informes. Un objeto **Recordset** de tipo Snapshot puede contener campos de una o más tablas de una base de datos pero no se puede actualizar. Este tipo corresponde a un cursor de tipo static ODBC.
- **Recordset de tipo Forward-only** - idéntico a un tipo Snapshot excepto que no se proporciona ningún cursor. Sólo puede avanzar en los registros. Esto mejora el rendimiento en situaciones donde sólo necesita hacer una pasada sencilla en el conjunto de resultado. Este tipo corresponde a un cursor de tipo forward-only ODBC.
- **Recordset de tipo Dynamic** - un conjunto de resultado de una consulta de una o más tablas base en las que puede agregar, cambiar o eliminar registros de una consulta que devuelve filas. Además, también aparecen en el objeto **Recordset** los registros que agregan, eliminan o modifican otros usuarios en la tablas base. Este tipo corresponde a un cursor de tipo dynamic ODBC (sólo espacios de trabajo ODBCDirect).

Puede elegir el tipo de objeto **Recordset** que quiere crear usando el argumento *tipo* del método **OpenRecordset**.

En un espacio de trabajo Microsoft Jet, si no especifica un *tipo*, DAO intenta crear el tipo de objeto **Recordset** con la mayor funcionalidad disponible, comenzando con tabla. Si no está disponible este tipo, DAO intenta un Dynaset, después un Snapshot y por último un objeto **Recordset** de tipo Forward-only.

En un espacio de trabajo ODBCDirect, si no especifica un *tipo*, DAO intenta crear el tipo de objeto **Recordset** con la respuesta de consulta más rápida, comenzando con Forward-only. Si no está disponible este tipo, DAO intenta un Snapshot, después un Dynaset y por último un objeto **Recordset** de tipo Dynamic.

Cuando se crea un objeto **Recordset** utilizando un objeto **TableDef** no adjunto, se crean objetos **Recordset** de tipo Table. Sólo pueden crearse **Recordset** de tipo Dynaset o Snapshot con tablas adjuntas o tablas de bases de datos externas ODBC.

Cuando abre el objeto se agrega automáticamente un nuevo objeto **Recordset** a la colección **Recordsets** y se elimina automáticamente cuando lo cierra.

Nota Si utiliza variables para representar un objeto **Recordset** y el objeto **Database** que contiene el conjunto de registros, compruebe que las variables tengan el mismo alcance o duración. Por ejemplo, si establece una variable global que representa un objeto **Recordset**, debe asegurarse de que la variable que represente la base de datos que contiene el conjunto de registros también sea global o se encuentra en un procedimiento **Sub** o **Function** con la palabra clave **Static**.

Su aplicación puede crear tantas variables objeto **Recordset** como se necesiten. Un objeto **Recordset** puede hacer referencia a una o más tablas o consultas y los campos sin conflictos.

Los **Recordset** de tipo Dynaset y Snapshot se almacenan en la memoria local. Si no hay suficiente espacio en la memoria local para almacenar los datos, el motor de base de datos Microsoft Jet guarda los datos adicionales en el disco TEMP. Si este espacio está agotado, se producirá un error.

La colección predeterminada de un objeto **Recordset** es la colección **Fields** y la propiedad predeterminada de un objeto **Field** es la propiedad **Value**. El código puede simplificarse utilizando estos valores predeterminados.

Cuando se crea un objeto **Recordset**, el registro activo se coloca como primer registro si existen varios registros. Si no hay registros, el valor de la propiedad **RecordCount** será 0 y los valores de la propiedad **BOF** y **EOF** serán **True**.

Puede utilizar los métodos **MoveNext**, **MovePrevious**, **MoveFirst** y **MoveLast** para volver a establecer el registro activo. Los objetos **Recordset** de tipo Forward-only sólo admiten el método **MoveNext**. Cuando se utilizan los métodos **Move** para moverse entre los registros (o "andar" a través del objeto **Recordset**), puede utilizar las propiedades **BOF** y **EOF** para comprobar el inicio o el fin del objeto **Recordset**.

Con los objetos **Recordset** de tipo Dynaset y Snapshot en un espacio de trabajo Microsoft Jet, también puede utilizar los métodos **Find**, como **FindFirst**, para localizar un registro específico basado en un criterio. Si no se encuentra el registro, la propiedad **NoMatch** se establece a **True**. Para objetos **Recordset** de tipo Table, puede buscar registros utilizando el método **Seek**.

La propiedad **Type** indica el tipo de objeto **Recordset** creado y la propiedad **Updatable** indica si puede cambiar los registros del objeto.

La información acerca de la estructura de la tabla base, como los nombres y los tipos de datos de cada objeto **Field** y cualquier objeto **Index**, se almacena en un objeto **TableDef**.

Para hacer referencia a un objeto **Recordset** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

Recordsets(0)
Recordsets("*nombre*")
Recordsets![*nombre*]

Nota Puede abrir un objeto **Recordset** del mismo origen de datos o base de datos más de una vez creando nombres duplicados en la colección **Recordsets**. Debe asignar objetos **Recordset** a variables de objeto y hacer referencia a ellos por el nombre de variable.

Recordsets (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolRecordsetU":1}

La colección **Recordsets** contiene todos los objetos **Recordset** abiertos en un objeto **Database**.

Comentarios

Cuando utiliza objetos de acceso de datos, se interactúa con los datos utilizando objetos **Recordset**.

Cuando abre el objeto se agrega automáticamente un nuevo objeto **Recordset** a la colección **Recordsets** y se elimina automáticamente cuando lo cierra.

Puede crear tantas variables de objeto del objeto **Recordset** como necesite. Objetos **Recordset** diferentes pueden tener acceso a las mismas tablas, consultas y campos sin conflictos.

Para hacer referencia a un objeto **Recordset** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

Recordsets(0)
Recordsets("*nombre*")
Recordsets![*nombre*]

Nota Puede abrir un objeto **Recordset** del mismo origen de datos o base de datos más de una vez creando nombres duplicados en la colección **Recordsets**. Debe asignar objetos **Recordset** a

variables de objeto y hacer referencia a ellos por el nombre de variable.

Recordset (Objeto), Recordsets (Colección) Resumen

Recordset (Objeto)

El objeto **Recordset** contiene los siguientes colecciones, métodos y propiedades:

Leyenda:

Característica disponible en espacios de trabajo Microsoft Jet y ODBCDirect.

Característica disponible sólo en espacios de trabajo Microsoft Jet.

Característica disponible sólo en espacios de trabajo ODBCDirect.

Colecciones

Fields (predeterminada)
Properties

Recordset (Métodos)

La siguiente tabla incluye métodos **Recordset** y muestra qué tipo de objeto **Recordset** admite cada método y si el método está disponible en Microsoft Jet o en un espacio de trabajo ODBCDirect o en ambos.

Método	Table	Dynaset	Snapshot	Forward-only	Dynamic
AddNew					
			*		
Cancel					
CancelUpdate					
			*		
Clone					
Close					
CopyQueryDef					
Delete					
			*		
Edit					
			*		
FillCache					
FindFirst					
FindLast					
FindNext					
FindPrevious					
GetRows					

Move				Sólo con movimientos hacia adelante que no utilicen una compensación de marcador	
MoveFirst					
MoveLast					
MoveNext					
MovePrevious					
NextRecordset					
OpenRecordset					
Requery					
Seek					
Update			*		

* En un espacio de trabajo ODBCDirect, un objeto **Recordset** de tipo Snapshot se puede actualizar dependiendo del controlador ODBC. Los métodos **AddNew**, **Edit**, **Delete**, **Update** y **CancelUpdate** sólo están disponibles en objetos **Recordset** ODBCDirect de tipo Snapshot si el controlador ODBC admite snapshots que se pueden actualizar.

Recordset (Propiedades)

La siguiente tabla indica las propiedades que se aplican a cada tipo de objeto **Recordset** y si el valor de la propiedad es de lectura/escritura, de sólo escritura o siempre **False** en bases de datos Microsoft Jet o ODBCDirect.

	Sólo lectura				
	Lectura/escritura				
Propiedad	**Table**	**Dynaset**	**Snapshot**	**Forward-only**	**Dynamic**
AbsolutePosition					
BatchCollisionCount					
BatchCollisions					
BatchSize					

BOF		
Bookmark		
Bookmarkable		
CacheSize		
	para espacios de trabajo Microsoft Jet	
	para espacios de trabajo ODBCDirect	
CacheStart		
Connection		
DateCreated		
EditMode		
EOF		
Filter		
Index		
LastModified		
		*
LastUpdated		
LockEdits		
	para espacios de trabajo	para espacios de trabajo

		Microsoft Jet para espacios de trabajo ODBCDirect	Microsoft Jet para espacios de trabajo ODBCDirect
Name			
NoMatch			
PercentPosition			
RecordCount			
RecordStatus			
Restartable	**False**		
Sort			
StillExecuting			
Transactions			
Type		Siempre **False**	Siempre **False**
Updatable		Siempre **False** en espacios de trabajo Microsoft Jet	Siempre **False** en espacios de trabajo Microsoft Jet
UpdateOptions		en espacios de trabajo ODBCDirect *	en espacios de trabajo ODBCDirect *
ValidationRule			

ValidationTe
xt

---

* En un espacio de trabajo ODBCDirect, un objeto **Recordset** de tipo Snapshot se puede actualizar dependiendo del controlador ODBC. Está disponible la propiedad **LastModified** y la propiedad **Updatable** es **True** sólo en objetos **Recordset** ODBCDirect de tipo Snapshot si el controlador ODBC admite snapshots actualizables.

Recordsets (Colección)

Una colección **Recordsets** aparece en cada objeto **Connection** y **Database** y contiene el siguiente método y la siguiente propiedad.

Método

Refresh

Propiedad

Count

Relation (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjRelationX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjRelationP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjRelationM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjRelationS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjRelationU":1}

Un objeto **Relation** representa una relación entre campos de tablas o consultas (sólo bases de datos Microsoft Jet).

Comentarios

Puede utilizar el objeto **Relation** para crear nuevas relaciones y examinar relaciones existentes en la base de datos.

Si utiliza un objeto **Relation** y sus propiedades, puede:

- Especificar una relación aplicable entre campos de tablas base (pero no una relación que exija una consulta o una tabla adjunta).
- Establecer relaciones no exigibles entre cualquier tipo de tablas o consultas o adjuntas.
- Utilizar la propiedad **Name** para hacer referencia a la relación entre los campos de la tabla principal de referencia y la tabla ajena.
- Utilizar la propiedad **Attributes** para determinar si la relación entre los campos de la tabla es de uno a uno o de uno a varios y si se debe exigir la integridad referencial.
- Utilizar la propiedad **Attributes** para determinar si el motor de base de datos Microsoft Jet puede realizar operaciones de actualización en cascada o de eliminación en cascada, en tablas principales y externas.
- Utilizar la propiedad **Attributes** para determinar si la relación entre los campos de la tabla es de combinación izquierda o de combinación derecha.
- Utilizar la propiedad **Name** de todos los objetos **Field** de la colección **Fields** de un objeto **Relation** para establecer o devolver los nombres de los campos en la clave principal del la tabla de referencia o el valor de la propiedad **ForeignName** de los objetos **Field** para establecer o devolver los nombres de los campos en la clave externa de la tabla de referencia.

Si realiza cambios que no cumplan las relaciones establecidas para la base de datos, se produce un

error interceptable. Si solicita una operación de actualización en cascada o una operación de eliminación en cascada, el motor de base de datos Microsoft Jet también modifica la tablas principal o la clave externa para exigir las relaciones que establece.

Por ejemplo, la base de datos Neptuno contiene una relación entre la tabla Pedidos y la tabla Clientes. El campo Id de cliente de la tabla Clientes es la clave principal y el campo Id de cliente de la tabla Pedidos es la clave externa. Para que Microsoft Jet acepte un registro nuevo en la tabla Pedidos, busca en la tabla Clientes una coincidencia en el campo Id de cliente de la tabla Pedidos. Si Microsoft Jet no encuentra ninguna coincidencia, no acepta el nuevo registro y se produce un error interceptable.

Cuando exige integridad referencial, ya debe existir un índice único para el campo clave en la tabla de referencia. El motor de base de datos Microsoft Jet crea automáticamente un índice con el valor de la propiedad **Foreign** para que funcione como la clave externa en la tabla de referencia.

Para crear un nuevo objeto **Relation**, utilice el método **CreateRelation**. Para hacer referencia a un objeto **Relation** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

Relations(0)
Relations("*nombre*")
Relations![*nombre*]

Relations (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolRelationX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolRelationP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolRelationM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolRelationS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolRelationU":1}

Una colección **Relations** contiene objetos **Relation** almacenados de un objeto **Database** (sólo bases de datos Microsoft Jet).

Comentarios

Puede utilizar el objeto **Relation** para crear nuevas relaciones y examinar relaciones existentes en la base de datos. Puede añadir un objeto **Relation** a la colección **Relations**, primero créela con el método **CreateRelation** y después agréguela a la colección **Relations** con el método **Append**. Esto guardará el objeto **Relation** cuando cierre el objeto **Database**. Para eliminar un objeto **Relation** de una colección, utilice el método **Delete**.

Para hacer referencia a un objeto **Relation** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

Relations(0)
Relations("*nombre*")
Relations![*nombre*]

Relation (Objeto), Relations (Colección) Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumRelationS"}

Relation (Objeto)

Un objeto **Relation** contiene las siguientes colecciones, métodos y propiedades.

Colecciones

Fields (predeterminada)
Properties

Método

CreateField

Propiedades

Attributes
ForeignTable
Name
PartialReplica
Table

Relations (Colección)

Una colección **Relations** está contenida en cada objeto **Database** de una base de datos Microsoft Jet, y contiene estos métodos y esta propiedad.

Métodos

Append
Delete
Refresh

Propiedad

Count

Recordset de tipo Snapshot (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjSnapshotTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjSnapshotTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjSnapshotTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjSnapshotTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjSnapshotTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjSnapshotTypeRecordsetU":1}

Un objeto **Recordset** de tipo Snapshot es un conjunto estático de registros que se puede utilizar para examinar datos en una tabla o tablas subyacentes. En una base de datos ODBCDirect, un objeto **Recordset** de tipo Snapshot corresponde a un cursor de tipo static.

Comentarios

Para crear un objeto **Recordset** de tipo Snapshot, utilice el método **OpenRecordset** en un base de datos abierta, en otro objeto **Recordset** de tipo Dynaset o Snapshot o en un objeto **QueryDef**.

Un objeto **Recordset** de tipo puede contener campos de una o más tablas de la base de datos. En un espacio de trabajo Microsoft Jet, no se puede actualizar un tipo Snapshot. En un espacio de trabajo ODBCDirect, un tipo Snapshot se puede actualizar, dependiendo del controlador ODBC.

Cuando se crea un objeto **Recordset** de tipo Snapshot, los valores de los datos de todos los campos (excepto los tipos de datos de campos Memo y OLE (Long Binary) de los archivos .mdb) se lleva a la memoria. Una vez cargados, los cambios realizados en los datos de la tabla base no se reflejan en los datos del objeto **Recordset** de tipo Snapshot. Para volver a cargar el objeto **Recordset** de tipo Snapshot con datos actuales, puede utilizar el método **Requery** o volver a ejecutar el método **OpenRecordset**.

El orden de los datos del objeto **Recordset** no sigue necesariamente ninguna secuencia específica. Para ordenar los datos se puede utilizar una instrucción SQL con una cláusula ORDER BY para crear el objeto **Recordset**. También puede utilizarse esta técnica para filtrar los registros, de modo que sólo algunos registros se añadan al objeto **Recordset**. Utilizar esta técnica en vez de las propiedades **Filter** o **Sort** o probar cada registro individualmente generalmente da como resultado un acceso más rápido a los datos.

Los objetos **Recordset** de tipo Snapshot suelen ser más rápidos a la hora de crear y acceder que los objetos **Recordset** de tipo Dynaset, porque sus registros se encuentran en la memoria o bien almacenados en el espacio TEMP del disco y el motor de base de datos Microsoft Jet no necesita bloquear páginas ni manejar objetivos multiusuario. Sin embargo, los objetos **Recordset** de tipo Snapshot utilizan más recursos que los objetos **Recordset** de tipo Dynaset porque se carga el registro completo en la memoria de trabajo.

Recordset de tipo Table (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjTableTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjTableTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjTableTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjTableTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjTableTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjTableTypeRecordsetU":1}

Un objeto **Recordset** de tipo Table es una representación en código de una tabla base que puede utilizarse para añadir, cambiar o borrar registros de una tabla. Sólo el registro activo está cargado en la memoria. Se utiliza un índice predefinido para determinar el orden de los registros en el objeto **Recordset** (sólo espacios de trabajo Microsoft Jet).

Comentarios

Para crear un objeto **Recordset**, utilice el método **OpenRecordset** en un objeto **Database** abierto.

Se puede crear un objeto **Recordset** de tipo Table con una tabla base de una base de datos Microsoft Jet, pero no con una ODBC o con una tabla adjunta. El objeto **Recordset** de tipo Table puede utilizarse con bases de datos ISAM (como FoxPro, dBASE o Paradox) cuando se abren directamente.

A diferencia de los objetos **Recordset** de tipo Dynaset o Snapshot, el objeto **Recordset** de tipo Table no puede hacer referencia a más de una tabla base y no puede accederse al mismo utilizando una instrucción SQL que filtre u ordene los datos. Por lo general, cuando se accede a un objeto **Recordset** de tipo Table, se especifica uno de los índices definidos de forma predeterminada para la tabla que ordena los datos devueltos a la aplicación. Si la tabla no tiene índice, no se puede garantizar el orden de los datos devueltos. Si es necesario, la aplicación puede crear un índice que devuelva los registros en un orden específico. Para seleccionar un orden específico para su objeto **Recordset** de tipo Table, puede establecer un índice válido para la propiedad **Index**.

También a diferencia de los objetos **Recordset** de tipo Dynaset o Snapshot, no necesita llenar explícitamente los objetos **Recordset** de tipo Table para obtener un valor exacto para la propiedad **RecordCount**.

Para mantener la integridad de los datos, los objetos **Recordset** de tipo Table se mantienen bloqueados durante los métodos **Edit** y **Update**, de modo que sólo un usuario puede actualizar un registro determinado cada vez. Cuando el motor de base de datos Microsoft Jet bloquea un registro, bloquea la página completa de 2 KB que contiene el registro.

Con las tablas no -ODBC se utilizan dos tipos de bloqueo: pesimista y optimista. Las tablas de acceso ODBC siempre utilizan el bloqueo optimista. La propiedad **LockEdits** determina las condiciones de bloqueo durante la edición.

TableDef (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjTableDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjTableDefP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjTableDefM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjTableDefS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjTableDefU":1}

Un objeto **TableDef** representa la definición almacenada de una tabla base o una tabla adjunta (sólo espacios de trabajo Microsoft Jet).

Comentarios

Para manipular una definición de tabla, utilice un objeto **TableDef** y sus métodos y propiedades. Por ejemplo, puede hacer lo siguiente:

- Examinar el campo y la estructura del índice de cualquier tabla local, adjunta o externa de una base de datos.
- Utilice las propiedades **Connect** y **SourceTableName** para establecer o devolver información acerca de las tablas vinculadas y utilice el método **RefreshLink** para actualizar conexiones a tablas vinculadas.
- Utilice las propiedades **ValidationRule** y **ValidationText** para establecer o devolver condiciones de validación.
- Utilice el método **OpenRecordset** para crear un objeto **Recordset** de tipo Table, Dynaset, Dynamic, Snapshot o Forward-only, basado en la definición de la tabla.

Para tablas base, la propiedad **RecordCount** contiene el número de registros en la tabla base. Para tablas vinculadas, el valor de la propiedad **RecordCount** es siempre -1.

Para crear un nuevo objeto **TableDef**, utilice el método **CreateTableDef**.

Para agregar un campo a una tabla

1. Asegúrese de que todos los objetos **Recordset** basados en la tabla están cerrados.
2. Utilice el método **CreateField** para crear una variable de objeto **Field** y establecer sus propiedades.
3. Utilice el método **Append** para agregar el objeto **Field** a la colección **Fields** del objeto **TableDef**.

Puede eliminar un objeto **Field** de una colección **TableDefs** si no tiene índices asignados, pero perderá los datos del campo.

Para crear una tabla lista para nuevos registros en una base de datos

1. Cree un objeto **CreateTableDef** utilizando el método **TableDef**.
2. Establezca sus propiedades.
3. Para cada campo de la tabla, utilice el método **CreateField** para crear una variable de objeto **Field** y establecer sus propiedades.
4. Utilice el método **Append** para añadir los campos a la colección **Fields** del objeto **TableDef**.
5. Utilice el método **Append** para añadir el nuevo objeto **TableDef** a la colección **TableDefs** del objeto **Database**.

Una tabla adjunta se encuentra vinculada a la base de datos mediante las propiedades **SourceTableName** y **Connect** del objeto **TableDef**.

Para adjuntar una tabla a una base de datos

1. Cree un objeto **TableDef** utilizando el método **CreateTableDef**.
2. Establezca sus propiedades **Connect** y **SourceTableName** (y, opcionalmente, su propiedad **Attributes**).
3. Añádalo a la colección **TableDefs** de un objeto **Database** utilizando el método **Append**.

Para hacer referencia a un objeto **TableDef** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

TableDefs(0)
TableDefs("*nombre*")
TableDefs![*nombre*]

TableDefs (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolTableDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolTableDefP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolTableDefM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolTableDefS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolTableDefU":1}

Una colección **TableDefs** contiene todos los objetos **TableDef** almacenados en una base de datos (sólo espacios de trabajo Microsoft Jet).

Comentarios

Para manipular una definición de tabla, utilice un objeto **TableDef** y sus métodos y propiedades.

La colección predeterminada de un objeto **Database** es la colección **TableDefs**.

Para hacer referencia a un objeto **TableDef** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

TableDefs(0)
TableDefs("*nombre*")
TableDefs![*nombre*]

TableDef (Objeto), TableDefs (Colección) Resumen

TableDef (Objeto)

Un objeto **TableDef** contiene las siguientes colecciones, métodos y propiedades:

Colecciones

Fields (predeterminada)
Indexes
Properties

Métodos

CreateField
CreateIndex
CreateProperty
OpenRecordset
RefreshLink

Propiedades

Attributes
ConflictTable
Connect
DateCreated
KeepLocal (definida por el usuario)
LastUpdated
Name
RecordCount
Replicable (definida por el usuario)
ReplicaFilter
SourceTableName
Updatable
ValidationRule
ValidationText

Un objeto **TableDef** contiene además propiedades definidas por la aplicación. Si desea información sobre la forma de leer y establecer estas propiedades, consulte la Ayuda en línea de la aplicación.

TableDefs (Colección)

Un objeto **Database** contiene una colección **TableDefs** en una base de datos Microsoft Jet y contiene estos métodos y esta propiedad.

Métodos

Append
Delete
Refresh

Propiedad

Count

User (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjUserC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjUserP"} {ewc HLP95EN.DLL,DYNALINK,"Mïï¿½todos":"daobjUserM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjUserS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjUserU":1}

Un objeto **User** representa una cuenta de usuario con autorizaciones de acceso cuando un objeto **Workspace** actúa como un grupo de trabajo asegurado (sólo espacios de trabajo Microsoft Jet).

Comentarios

Los objetos **User** se utilizan para establecer y mantener autorizaciones de acceso para los objetos **Document** que representan bases de datos, tablas, y consultas. Además, si conoce las propiedades de un determinado objeto **User**, puede crear un nuevo objeto **Workspace** para el que dispondrá de las mismas autorizaciones de acceso del objeto **User**.

Puede añadir un objeto **User** existente a la colección **Users** de un objeto **Group** para dotar a la cuenta de usuario de las autorizaciones de acceso de ese objeto **Group**. Como alternativa, también puede añadir el objeto **Group** a la colección **Groups** del objeto **User** para establecer la pertenencia de la cuenta de usuario a ese grupo. Si utiliza una colección **Users** o **Groups** distinta de aquella a la que acaba de añadir el objeto, puede ser necesario utilizar el método **Refresh**.

Las propiedades del objeto **User** le permiten:

- Utilice la propiedad **Name** para devolver el nombre de un usuario existente. No puede devolver las propiedades **PID** y **Password** de un objeto **User** existente.
- Utilice las propiedades **Name**, **PID** y **Password** de un objeto **User** nuevo y sin agregar para

establecer la identidad de ese objeto **User**. Si no establece la propiedad **Password**, su valor será una cadena de longitud cero ("").

El motor de base de datos Microsoft Jet define de modo predeterminado dos objetos **User** llamados Administrador e Invitado. El usuario Administrador es miembro de los objetos **Group** llamados Administradores y Usuarios y el usuario Invitado sólo es miembro del objeto **Group** llamado Invitados.

Para crear un nuevo objeto **User**, utilice el método **CreateUser**.

Para hacer referencia a un objeto **User** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

[*espaciodetrabajo* | *grupo*].**Users**(0)
[*espaciodetrabajo* | *grupo*].**Users**("*nombre*")
[*espaciodetrabajo* | *grupo*].**Users**![*nombre*]

Users (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolUserC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolUserP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolUserM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolUserS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolUserU":1}

Una colección **Users** contiene todos los objetos **User** almacenados de un objeto **Workspace** o de un objeto **Group** (sólo espacios de trabajo Microsoft Jet).

Comentarios

Puede añadir un objeto **User** existente a la colección **Users** de un objeto **Group** para dotar a la cuenta de usuario de las autorizaciones de acceso de ese objeto **Group**. Como alternativa, también puede añadir el objeto **Group** a la colección **Groups** del objeto **User** para establecer la pertenencia de la cuenta de usuario a ese grupo. Si utiliza una colección **Users** o **Groups** distinta de aquella a la que acaba de añadir el objeto, puede ser necesario que utilice el método **Refresh**.

El motor de base de datos Microsoft Jet define de modo predeterminado dos objetos **User** llamados Administrador e Invitado. El usuario Administrador es miembro de los objetos **Group** llamados Administradores y Usuarios; y el usuario Invitado sólo es miembro del objeto **Group** llamado Invitados.

Para hacer referencia a un objeto **User** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

[*espaciodetrabajo* | *grupo*].**Users**(0)
[*espaciodetrabajo* | *grupo*].**Users**("*nombre*")

[*espaciodetrabajo* | *grupo*].**Users**![*nombre*]

User (Objeto), Users (Colección) Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumUserC"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumUserS"}

User (Objeto)

El objeto **User** contiene las siguientes colecciones, métodos y propiedades.

Colecciones

Groups (predeterminada)
Properties

Métodos

CreateGroup
NewPassword

Propiedades

Name
Password
PID

Users (Colección)

Un objeto **Workspace** de Microsoft Jet y cada objeto **Group** contiene una colección **Users**, que contiene los siguientes métodos y propiedades.

Métodos

Append
Delete
Refresh

Propiedad

Count

Workspace (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjWorkspaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjWorkspaceP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjWorkspaceM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjWorkSpaceS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjWorkSpaceU":1}

Un objeto **Workspace** define una sesión para un usuario. Contiene las bases de datos abiertas y proporciona mecanismos para realizar transacciones y, en espacios de trabajo Microsoft Jet admite asegurar grupos de trabajo. También controles si utiliza el motor de base de datos Microsoft Jet o ODBCDirect para tener acceso a datos externos.

Comentarios

Un objeto **Workspace** no es un objeto persistente que define cómo interactúa la aplicación con datos - utilizando el motor de base de datos Microsoft Jet usuario ODBCDirect. Utilice el objeto **Workspace** para administrar la sesión actual o para iniciar una sesión adicional. En una sesión, puede abrir múltiples bases de datos o conexiones y administrar transacciones. Por ejemplo, puede:

- Utilizar las propiedades **Name**, **UserName** y **Type** para establecer una sesión con nombre. La sesión crea un alcance en el que puede abrir múltiples bases de datos y dirigir una instancia de transacciones anidadas.
- Utilice el método **Close** para terminar una sesión.
- Utilizar el método **OpenDatabase** para abrir una o más bases de datos existentes en ese **Workspace**.
- Utilizar los métodos **BeginTrans**, **CommitTrans** y **Rollback** para administrar el proceso de transacciones anidadas en un **Workspace** y utilizar varios objetos **Workspace** para realizar varias transacciones simultáneas y superpuestas.

Además, utilizar una base de datos Microsoft Jet, puede establecer seguridad basada en nombres de usuario y contraseñas:

- Utilice las colecciones **Groups** y **Users** para establecer permisos de acceso a nivel de grupo y usuario a objetos en el objeto **Workspace**.
- Utilice la propiedad **IsolateODBCTrans** para aislar múltiples transacciones que involucren a la misma base de datos ODBC conectada a Microsoft Jet.

Nota Para una lista completa de los métodos, propiedades y colecciones disponibles en un objeto **Workspace** en una base de datos Microsoft Jet o en una base de datos ODBCDirect, vea el tema Resumen.

Cuando hace la primera referencia o utiliza un objeto **Workspace**, automáticamente crea el espacio de trabajo predeterminado, `DBEngine.Workspaces(0)`. Los valores de las propiedades **Name** y **UserName** del espacio de trabajo predeterminado son "#Default Workspace#" y "Admin," respectivamente. Si está establecida la seguridad, el valor de la propiedad **UserName** es el nombre del usuario que inició la sesión.

Para establecer un objeto **Workspace** ODBCDirect y, por tanto , evitar la carga del motor de base de datos Microsoft Jet en memoria, establezca la propiedad **DefaultType** del objeto **DBEngine** a **dbUseODBC** o establezca el argumento *tipo* del método **CreateWorkspace** a **dbUseODBC**.

Cuando utilice transacciones, afecta a todas las bases de datos del objeto **Workspace** especificado - incluso si están abiertos múltiples objetos **Database** en el objeto **Workspace**. Por ejemplo, utilice un método **BeginTrans**, actualice varios registros en una base de datos y después elimine registros en otra base de datos. Si después utiliza el método **Rollback**, las operaciones de actualización y eliminación se cancelan y se deshacen los cambios. Puede crear objetos **Workspace** adicionales para administrar transacciones independientemente a través de los objetos **Database**.

Puede crear objetos **Workspace** con el método **CreateWorkspace**. Después de crear un nuevo objeto **Workspace**, debe agregarlo a la colección **Workspaces** si necesita hacer referencia a él desde la colección **Workspaces**.

Puede utilizar un objeto **Workspace** creado nuevamente sin agregarlo a la colección **Workspaces**. Sin embargo, debe hacer referencia a él mediante la variable de objeto que tiene asignada.

Para hacer referencia a un objeto **Workspace** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

DBEngine.**Workspaces**(0)
DBEngine.**Workspaces**("*nombre*")
DBEngine.**Workspaces**![*nombre*]

Workspaces (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolWorkspaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolWorkspaceP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolWorkspaceM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolWorkspaceS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolWorkspaceU":1}

La colección **Workspaces** contiene todos los objetos **Workspace** activos, no ocultos, del objeto **DBEngine**. (Los objetos **Workspace** ocultos no se agregan a la colección y se hace referencia a los mismos mediante la variable que tienen asignada.)

Comentarios

Utilice el objeto **Workspace** para administrar la sesión activa o comenzar una sesión adicional.

Cuando hace referencia o utiliza por primera vez un objeto **Workspace**, automáticamente crea el espacio de trabajo predeterminado, `DBEngine.Workspaces(0)`. Los valores de las propiedades **Name** y **UserName** del espacio de trabajo predeterminado son "#Default Workspace#" y "Admin," respectivamente. Si está activa la seguridad, el valor de la propiedad **UserName** es el nombre del usuario que inició la sesión.

Puede crear nuevos objetos **Workspace** con el método **CreateWorkspace**. Después de crear un objeto **Workspace** nuevo, debe agregarlo a la colección **Workspaces** si necesita hacer referencia a él desde la colección **Workspaces**. Sin embargo, puede utilizar un objeto **Workspace** creado nuevamente sin agregarlo a la colección **Workspaces**.

Para hacer referencia a un objeto **Workspace** en una colección por su número de orden o por el valor de la propiedad **Name**, utilice cualquiera de los formatos de sintaxis siguientes:

DBEngine.**Workspaces**(0)
DBEngine.**Workspaces**("*nombre*")
DBEngine.**Workspaces**![*nombre*]

Workspace (Objeto), Workspaces (Colección) Resumen

Workspace (Objeto)

El objeto **Workspace** contiene las siguientes colecciones, métodos y propiedades.

Leyenda:

Característica disponible sólo en espacios de trabajo Microsoft Jet.

Característica disponible sólo en espacios de trabajo ODBCDirect.

Colecciones

Connections
Databases (predeterminada)

Groups
Properties

Users

Métodos

BeginTrans
Close
CommitTrans

CreateDatabase

CreateGroup

CreateUser

OpenConnection
OpenDatabase
Rollback

Propiedades

DefaultCursorDriver

IsolateODBCTrans

LoginTimeout
Name
Type
UserName

Workspaces (Colección)

La colección **Workspaces** está incluida en el objeto **DBEngine** y contiene los siguientes métodos y

propiedades.

Métodos

Append
Delete
Refresh

Propiedad

Count

Connection (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjConnectionP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjConnectionM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjConnectionS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjConnectionU":1}

Un objeto **Connection** representa una conexión con una base de datos ODBC (sólo espacios de trabajo ODBCDirect).

Comentarios

Una **Connection** es un objeto no persistente que representa una conexión con una base de datos remota. El objeto **Connection** sólo está disponible en espacios de trabajo ODBCDirect (es decir, un objeto **Workspace** creado con la opción *tipo* establecida a **dbUseODBC**).

Nota El código escrito para versiones anteriores de DAO puede seguir utilizando el objeto **Database** para compatibilidad con versiones anteriores, pero si desea las nuevas funciones de una **Connection**, debería revisar el código para utilizar el objeto **Connection**. Para ayudarle en la conversión del código puede obtener una referencia de un objeto **Connection** de una **Database** leyendo la propiedad **Connection** del objeto **Database**. Y al contrario, puede obtener una referencia de un objeto **Database** desde la propiedad **Database** del objeto **Connection**.

Connections (Colección)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dacolConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"dacolConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"dacolConnectionP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"dacolConnectionM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dacolConnectionS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"dacolConnectionU":1}

Una colección de **Connections** contiene los objetos **Connection** activos de un objeto **Workspace**. (Sólo espacios de trabajo ODBCDirect).

Comentarios

Cuando abre un objeto **Connection**, éste se anexa automáticamente a la colección de **Connections** del **Workspace**. Cuando cierra un objeto **Connection** con el método **Close**, éste se elimina de la colección de **Connection**. Debería cerrar todos los objetos **Recordset** dentro de la **Connection** antes de cerrarla.

Al mismo tiempo que abre un objeto **Connection**, se crea y se anexa a la colección de **Databases** un objeto **Database** correspondiente en el mismo **Workspace** y viceversa. De manera parecida, si cierra la **Connection**, se elimina de la colección de **Databases** la **Database** correspondiente, etc.

El valor de la propiedad **Name** de una **Connection** es una cadena que especifica la ruta de acceso del archivo de la base de datos. Para referir un objeto **Connection** en una colección por su número ordinal o por su valor de propiedad **Name**, use una de las siguientes formas sintácticas

Connections(0)
Connections("*nombre*")
Connections![*nombre*]

Nota Puede abrir el mismo origen de datos más de una vez creando nombres duplicados en la colección de **Connections**. Debería asignar objetos de **Connection** a variables de objetos y referirlos por el nombre de la variable.

Connection (Objetos), Connections (Colección), Resumen

Objeto conexión

El objeto **Connection** contiene las siguientes colecciones, métodos y propiedades.

Colecciones

QueryDefs (predeterminado)
Recordsets

Métodos

Cancel
Close
CreateQueryDef
Execute
OpenRecordset

Propiedades

Connect
Database
Name
QueryTimeout
RecordsAffected
StillExecuting
Transactions
Updatable

Colección de conexiones

En cada objeto **Workspace** de ODBCDirect hay una colección de **Connections** que contiene el siguiente método y propiedad:

Método

Refresh

Propiedad

Count

Recordset de tipo Forward-only (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjForwardOnlyTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjForwardOnlyTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjForwardOnlyTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjForwardOnlyTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjForwardOnlyTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Resumen"."daobjForwardOnlyTypeRecordsetU":1}

Este tipo de objeto **Recordset** es idéntico a una Snapshot excepto que sólo puede desplazarse hacia delante en sus registros. Esto mejora el rendimiento en situaciones donde sólo se necesita hacer una única pasada por un conjunto de resultados.

En un espacio de trabajo ODBCDirect este tipo se corresponde con un cursor de tipo Forward-only de ODBC

Recordset de tipo Forward-only (Objeto), Resumen

El objeto **Recordset** de tipo Forward-only contiene las siguientes colecciones, métodos y propiedades:

Leyenda:

Función disponible sólo espacios de trabajo Microsoft Jet.

Función disponible sólo espacios de trabajo ODBCDirect.

Colecciones	
Fields (predeterminado)	
Properties	
Métodos	**Restricciones**
AddNew	
Cancel	
CancelUpdate	
Close	
CopyQueryDef	
Delete	
Edit	
GetRows	
Move	Sólo con movimientos hacia delante que no utilicen compensación de marcadores.
MoveNext	
NextRecordset	
Requery	
Update	

Propiedades

La siguiente tabla indica si cada valor de propiedad es lectura/escritura, sólo lectura o siempre **False** en espacios de trabajo Microsoft Jet o ODBCDirect.

Sólo lectura

Lectura/escritura

Propiedades	Restricciones
BatchCollisionCount	
BatchCollisions	
BatchSize	

BOF	
Connection	
EOF	
Filter	
Name	
RecordCount	
RecordStatus	
Restartable	
StillExecuting	
Transactions	Siempre **False**
Type	
Updatable	**False**
UpdateOptions	
ValidationRule	
ValidationText	

Recordset de tipo Dynamic (Objeto)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daobjDynamicTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daobjDynamicTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Propiedades":"daobjDynamicTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Mï¿½todos":"daobjDynamicTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daobjDynamicTypeRecordset S"} {ewc HLP95EN.DLL,DYNALINK,"Resumen":"daobjDynamicTypeRecordsetU":1}

Este tipo de **Recordset** representa un conjunto de resultados de una consulta de una o más tablas base en el que puede agregar, cambiar o eliminar registros de una consulta que devuelve filas. También aparecen en su **Recordset** los registros que otros usuarios agregan, eliminan o editan en las tablas base.

Este tipo sólo está disponible en espacios de trabajo ODBCDirect y se corresponde con un cursor de tipo Dynamic ODBC.

Recordset de tipo Dynamic (Objeto), Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumDynamicTypeRecordsetC "} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumDynamicTypeRecordsetS"}

Un objeto **Recordset** de tipo Dynamic contiene la siguientes colecciones, métodos y propiedades. Este tipo de **Recordset** y sus métodos y propiedades sólo están disponibles en un espacio de trabajo ODBCDirect.

Colecciones

Fields (predeterminado)

Properties

Métodos

AddNew
Cancel
CancelUpdate
Close
Delete
Edit
GetRows
Move
MoveFirst
MoveLast
MoveNext
MovePrevious
NextRecordset
Requery
Update

Propiedades

La siguiente tabla indica si cada valor de la propiedad es lectura/escritura, sólo lectura o siempre **False**.

Sólo lectura

Lectura/escritura

Propiedades	Restricciones
AbsolutePosition	
BatchCollisionCount	
BatchCollisions	
BatchSize	
BOF	
Bookmark	
Bookmarkable	
CacheSize	

Connection

EditMode

EOF

LastModified

LockEdits

Name

PercentPosition

RecordCount

RecordStatus

Restartable

StillExecuting

Type

Updatable

UpdateOptions

Recordset de tipo Dynaset (Objeto), Resumen

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"dasumDynasetTypeRecordsetC "} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"dasumDynasetTypeRecordsetS"}

El objeto **Recordset** de tipo Dynaset contiene las siguientes colecciones, métodos y propiedades.

Leyenda:

Función disponible sólo espacios de trabajo Microsoft Jet .

Función disponible sólo espacios de trabajo ODBCDirect.

Colecciones

Fields (predeterminado)
Properties

Métodos	**Restricciones**
AddNew	
Cancel	
CancelUpdate	
Clone	
Close	
CopyQueryDef	
Delete	
Edit	
FillCache	
FindFirst	
FindLast	
FindNext	
FindPrevious	
GetRows	
Move	
MoveFirst	
MoveLast	
MoveNext	
MovePrevious	
NextRecordset	
OpenRecordset	
Requery	
Update	

Propiedades

La siguiente tabla indica si el valor de la propiedad es lectura/escritura, sólo lectura o sólo disponible en espacios de trabajo Microsoft Jet u ODBCDirect.

Sólo lectura

Lectura/escritura

Propiedades	Restricciones
AbsolutePosition	
BatchCollisionCount	
BatchCollisions	
BatchSize	
BOF	
Bookmark	
Bookmarkable	
CacheSize	Microsoft Jet ODBCDirect
CacheStart	
Connection	
EditMode	
EOF	
Filter	
LastModified	
LockEdits	
Name	
NoMatch	
PercentPosition	
RecordCount	
RecordStatus	
Restartable	
Sort	
StillExecuting	
Transactions	
Type	
Updatable	
UpdateOptions	
ValidationRule	

ValidationText

Recordset de tipo Snapshot (Objeto), Resumen

El objeto **Recordset** de tipo Snapshot contiene las siguientes colecciones, métodos y propiedades.

Leyenda:

Función disponible sólo espacios de trabajo Microsoft Jet .

Función disponible sólo espacios de trabajo ODBCDirect.

Colecciones	
Fields (predeterminado)	
Properties	
Métodos	**Restricciones**
AddNew	*
Cancel	
CancelUpdate	*
Clone	
Close	
CopyQueryDef	
Delete	*
Edit	*
FindFirst	
FindLast	
FindNext	
FindPrevious	
GetRows	
Move	
MoveFirst	
MoveLast	
MoveNext	
MovePrevious	
NextRecordset	
OpenRecordset	
Requery	
Update	*

* En un espacio de trabajo ODBCDirect, un **Conjunto de registros** de tipo Snapshot, se puede actualizar, dependiendo del controlador de ODBC. Los métodos **AddNew**, **Edit**, **Delete**, **Update** y **CancelUpdate** sólo están disponibles en objetos **Recordset** de tipo Snapshot de ODBCDirect si el controlador de ODBC admite snapshots que se pueden actualizar.

Propiedades

La siguiente tabla indica si el valor de la propiedad es lectura/escritura, sólo lectura o siempre **False** en espacios de trabajo Microsoft Jet o ODBCDirect.

Sólo lectura

Lectura/escritura

Propiedades	Restricciones
AbsolutePosition	
BatchCollisionCount	
BatchCollisions	
BatchSize	
BOF	
Bookmark	
Bookmarkable	
CacheSize	
Connection	
EditMode	
EOF	
Filter	
LastModified	
LockEdits	*
Name	
NoMatch	
PercentPosition	
RecordCount	
RecordStatus	
Restartable	
Sort	
StillExecuting	
Transactions	Siempre **False**
Type	
Updatable	Siempre **False** en espacios de trabajo Microsoft Jet,
	en espacios de

trabajo ODBCDirect *

UpdateOptions

ValidationRule

ValidationText

---

* En un espacio de trabajo ODBCDirect, un **Recordset** de tipo Snapshot se puede actualizar, dependiendo del controlador de ODBC. La propiedad **LastModified** está disponible y la propiedad **Updatable** es **True** en objetos de **Recordset** de tipo Snapshot de ODBCDirect sólo si el controlador de ODBCDirect admite snapshots que se pueden actualizar.

Recordset de tipo Table (Objeto), Resumen

Un objeto **Recordset** de tipo Table contiene los siguientes colecciones, métodos y propiedades. Este tipo de **Recordset** y sus métodos sólo están disponibles en un espacio de trabajo Microsoft Jet.

Colecciones

Fields (predeterminada)
Properties

Métodos

AddNew
CancelUpdate
Clone
Close
Delete
Edit
GetRows
Move
MoveFirst
MoveLast
MoveNext
MovePrevious
OpenRecordset
Seek
Update

Propiedades

La siguiente tabla indica si el valor de cada propiedad es lectura/escritura, sólo lectura o siempre **False**.

Propiedades	Sólo lectura	Lectura/escritura	Restricciones
BOF			
Bookmark			
Bookmarkable			
DateCreated			
EditMode			
EOF			
Index			
LastModified			

LastUpdated	
LockEdits	
Name	
NoMatch	
PercentPosition	
RecordCount	
Restartable	Siempre **False**
Transactions	
Type	
Updatable	
ValidationRule	
ValidationText	

AddNew (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthAddNewC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthAddNewX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthAddNewA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthAddNewS"}

Crea un nuevo registro para un objeto **Recordset** de tipo Table o Dynaset.

Sintaxis

recordset.**AddNew**

El marcador de posición del *recordset* es una variable de objeto que representa un objeto **Recordset** que se puede actualizar al que puede agregar un registro nuevo.

Comentarios

Utilice el método **AddNew** para crear y agregar un nuevo registro en el objeto **Recordset** llamado por el *recordset*. Este método establece los campos a los valores predeterminados y si no se especifican valores predeterminados, establece los campos a **Null** (los valores predeterminados especificados pare el **Recordset** tipo Table).

Después de modificar el nuevo registro, utilice el método **Update** para guardar los cambios y agregar el registro al **Recordset**. No se producirán cambios en la base de datos hasta que no se utilice el método **Update**.

Precaución Si ejecuta un **AddNew** y a continuación realiza una operación que desplace otro registro sin usar **Update**, los cambios se perderán sin previo aviso. Además, si cierra el **Recordset** o finaliza el procedimiento que declara el **Recordset** o su objeto **Database**, el nuevo registro y los cambios realizados se descartarán sin previo aviso.

Nota Cuando utilice **AddNew** en un espacio de trabajo Microsoft Jet y el motor de base de datos tenga que crear una nueva página para almacenar el registro activo, el bloqueo de páginas será pesimista. Si el nuevo registro cabe en una página existente, el bloqueo de páginas será optimista.

Si no se ha desplazado hasta el último registro de su **Recordset**, los registros agregados a las tablas subyacentes pueden incluirse, si se colocan más allá del registro activo. Sin embargo, si agrega un registro a un **Recordset**, el registro será visible en el **Recordset** y se incluirá en la tabla subyacente donde estará visible para todos los nuevos objetos **Recordset**.

La posición del nuevo registro depende del tipo de **Recordset**:

- En un objeto **Recordset** tipo Dynaset, los registros se insertan al final del conjunto del **Recordset**, independientemente de las reglas de clasificación u orden que estuvieran en vigor cuando se abrió el **Recordset**.
- En un objeto **Recordset** tipo Table en el que su propiedad **Index** se haya establecido, los registros se insertan en el lugar adecuado dentro del orden definido. Si no se ha establecido la propiedad **Index**, los nuevos registros se insertarán al final del **Recordset**.

El registro que estaba activo antes de utilizar **AddNew** permanece activo. Si desea convertir el nuevo registro en el registro activo, puede establecer la propiedad **Bookmark** con marcador identificado por el valor de la propiedad **LastModified**.

Nota Para agregar, modificar o eliminar un registro, debe haber sólo un índice único en el registro en el origen de datos base. Si no, se producirá un error "Permiso denegado " en el método **AddNew**, **Delete** o **Edit** que se llama en un espacio de trabajo Microsoft Jet o se producirá un error "Argumento no válido " en el método **Update** que se llama en un espacio de trabajo ODBCDirect.

Append (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthAppendC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthAppendX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthAppendA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthAppendS"}

Agrega un nuevo objeto DAO a una colección.

Sintaxis

colección.**Append** *objeto*

La sintaxis del método **Append** consta de los siguientes argumentos.

Argumento	Descripción
colección	Una variable de objeto que representa cualquier colección que puede aceptar objetos nuevos (para ver las limitaciones, consulte la tabla al final de este tema).
objeto	Una variable de un tipo de datos objeto que identifica el objeto que se va a agregar, que debe ser del mismo tipo que los elementos de la *colección.*

Comentarios

Se puede utilizar el método **Append** para agregar una nueva tabla a una base de datos, agregar un campo a una tabla y agregar un campo a un índice.

El objeto añadido se convierte en un objeto persistente, almacenado en disco, hasta que se elimine mediante el método **Delete**. Si la *colección* es una colección **Workspaces** (que se almacena sólo en memoria), el objeto permanecerá activo hasta que se elimine mediante el método **Close**.

La adición de un nuevo objeto ocurre inmediatamente, pero debe utilizar el método **Refresh** en cualquier otra colección que pueda verse afectada por cambios en la estructura de la base de datos.

Si el objeto que se añade no está completo (como ocurre cuando no ha añadido ningún objeto **Field** a una colección **Fields** de un objeto **Index** antes de agregarlo a una colección **Indexes**) o si las propiedades establecidas para uno o más objetos subordinados son incorrectas, el método **Append** genera un error. Por ejemplo, si no ha especificado un tipo de campo y a continuación intenta agregar el objeto **Field** a la colección **Fields** de un objeto **TableDef**, el método **Append** genera un error interceptable.

La siguiente tabla muestra algunas limitaciones en el uso del método **Append**. El objeto de la primera columna es un objeto que contiene la colección de la segunda columna. La tercera columna indica cuándo es posible agregar un objeto a esa colección (por ejemplo, nunca se puede agregar un objeto **Container** a la colección **Containers** de un objeto **Database**).

Objeto	Colección	¿Puede anexar nuevos objetos?
DBEngine	**Workspaces**	Sí
DBEngine	**Errors**	No. Los objetos Error nuevo se anexan automáticamente cuando se produce.
Workspace	**Connections**	No. Utilizando el método **OpenConnection** se anexan automáticamente los objetos nuevos.
Workspace	**Databases**	No. Utilizando el método **OpenDatabase** se anexan

		automáticamente los objetos nuevos.
Workspace	**Groups**	Sí
Workspace	**Users**	Sí
Connection	**QueryDefs**	No. Utilizando el método **CreateQueryDef** se anexan automáticamente los objetos nuevos.
Connection	**Recordsets**	No. Utilizando el método **OpenRecordset** se anexan automáticamente los objetos nuevos.
Database	**Containers**	No
Database	**QueryDefs**	Sólo cuando el objeto **QueryDef** es un objeto nuevo no anexado creado sin nombre. Consulte el método **CreateQueryDef** para obtener más detalles.
Database	**Recordsets**	No. Utilizando el método **OpenRecordset** anexan automáticamente los objetos nuevos.
Database	**Relations**	Sí
Database	**TableDefs**	Sí
Group	**Users**	Sí
User	**Groups**	Sí
Container	**Documents**	No
QueryDef	**Fields**	No
QueryDef	**Parameters**	No
Recordset	**Fields**	No
Relation	**Fields**	Sí
TableDef	**Fields**	Sólo cuando la propiedad **Updatable** del objeto **TableDef** se establece a **True** o cuando el objeto **TableDef** no está anexado.
TableDef	**Indexes**	Sólo cuando la propiedad **Updatable** del objeto **TableDef** se establece a **True** o cuando el objeto **TableDef** no está anexado.
Index	**Fields**	Sólo cuando el objeto **Index** es un objeto nuevo no anexado.
Database, **Field, Index**, **QueryDef**, **TableDef**	**Properties**	Sólo cuando el objeto **Database**, **Field**, **Index**, **QueryDef** o **TableDef** está en un espacio de trabajo Microsoft Jet.
DBEngine,	**Properties**	No

Parameter,
Recordset,
Workspace

AppendChunk (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthAppendChunkC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthAppendChunkX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthAppendChunkA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthAppendChunkS"}

Añade datos de una expresión de cadena a un objeto **Field** Memo o Long Binary de un objeto **Recordset**.

Sintaxis

recordset **!** *campo***.AppendChunk** *origen*

La sintaxis del método **AppendChunk** utiliza los siguientes argumentos.

Argumento	Descripción
recordset	Una variable de un tipo de datos objeto que hace referencia al objeto **Recordset** que contiene la colección **Fields**.
campo	El nombre de un objeto **Field** cuya propiedad **Type** se establece como **dbMemo** (de tipo Memo), **dbLongBinary** (de tipo Long Binary) o el equivalente.
origen	Una expresión o variable de tipo **Variant** (subtipo **String**) contiene los datos que se desean agregar al objeto **Field** especificado por *campo.*

Comentarios

Puede utilizar los métodos **AppendChunk** y **GetChunk** para acceder a subconjuntos de datos de un campo de tipo Memo y Long Binary.

También puede utilizar estos métodos para conservar espacio de cadena cuando trabaje con campos de tipo Memo y Long Binary. Algunas operaciones (como copiar) generan cadenas temporales. Si el espacio de cadena es limitado, puede que necesite trabajar con fragmentos de un campo en lugar de hacerlo con el campo completo.

Si no hay ningún registro activo cuando utilice **AppendChunk** se producirá un error.

Nota

- El **AppendChunk** inicial (después del primer método **Edit** o **AddNew**) aunque el registro ya contenga datos, simplemente coloca los datos en el campo como si fueran los únicos datos. Por lo tanto las llamadas **AppendChunk** de una sesión **Edit** o **AddNew** se agregarán a continuación de los datos.
- En un espacio de trabajo ODBCDirect, a menos que primero modifique otro campo en el registro activo, utilizando **AppendChunk** fallará (no se produce un error) mientras está en el modo **Edit**.
- En un espacio de trabajo ODBCDirect, después de que utilice **AppendChunk** en un campo, no puede leer o grabar ese campo en una instrucción de asignación hasta que salga del registro activo y vuelva a él. Puede hacer esto utilizando los método **MoveNext** y **MovePrevious**.

BeginTrans, CommitTrans, Rollback (Métodos)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthBeginTransC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthBeginTransX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthBeginTransA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthBeginTransS"}

Los métodos de transacción administran el proceso de transacción durante una sesión definida por un objeto **Workspace** de la siguiente manera:

- **BeginTrans** inicia una nueva transacción.
- **CommitTrans** termina la transacción actual y guarda los cambios.
- **Rollback** termina la transacción activa y restaura las bases de datos del objeto **Workspace** al estado en que se encontraban cuando se inició la transacción activa.

Sintaxis

espaciodetrabajo.**BeginTrans | CommitTrans | Rollback**

El marcador de posición *espaciodetrabajo* es una variable de objeto que representa **Workspace** que contiene las bases de datos que utilizarán transacciones.

Comentarios

Estos métodos se utilizan con un objeto **Workspace** cuando se desea tratar como una unidad a una serie de cambios realizados en las bases de datos durante una sesión.

Normalmente, las transacciones se utilizan para mantener la integridad de los datos cuando se deben actualizar registros en dos o más tablas y comprobar que los cambios realizados están completos (ejecutados) en todas las tablas o en ninguna (anulados). Por ejemplo, si transfiere dinero de una cuenta a otra, puede substraer una cantidad de una y agregarla a la otra. Si una de las actualizaciones falla, las cuentas ya no cuadrarán. Utilice el método **BeginTrans** antes de actualizar el primer registro y luego, si falla cualquier actualización posterior, podrá usar el método **Rollback** para deshacer todas las actualizaciones. Utilice el método **CommitTrans** después de haber actualizado correctamente el último registro.

Precaución Dentro de un objeto **Workspace**, las transacciones siempre son globales al **Workspace** y no están limitadas a un sólo objeto **Connection** o **Database**. Si ejecuta operaciones en más de una conexión o base de datos dentro de una transacción **Workspace**, que resuelve la transacción (esto es, utilizando el método **CommitTrans** o **Rollback**) afecta a todas las operaciones en todas las conexiones y bases de datos del espacio de trabajo.

Una vez que utilice **CommitTrans**, no podrá deshacer los cambios realizados durante esa transacción, a menos que la transacción esté anidada dentro de otra a la que se deshicieron los cambios. Si anida transacciones, deberá guardar o anular la transacción activa antes de poder guardar o anular una transacción de un nivel de anidamiento superior.

Si desea tener transacciones simultáneas con alcances superpuestos y no anidados, puede crear objetos **Workspace** adicionales para contener las transacciones concurrentes.

Si cierra un objeto **Workspace** sin guardar o anular las transacciones pendientes, las transacciones se anularán automáticamente.

Si utiliza los métodos **CommitTrans** o **Rollback** sin usar primero el método **BeginTrans**, se producirá un error.

Algunas bases de datos ISAM utilizadas en un espacio de trabajo Microsoft Jet pueden no aceptar transacciones, en cuyo caso la propiedad **Transactions** del objeto **Database** o **Recordset** tendrá el valor **False**. Para asegurarse de que la base de datos soporta transacciones, compruebe el valor de la propiedad **Transactions** del objeto **Database** antes de usar el método **BeginTrans**. Si utiliza un objeto **Recordset** basado en más de una base de datos, compruebe la propiedad **Transactions** del objeto **Recordset**. Si un **Recordset** está totalmente basado en las tablas Microsoft Jet, siempre

podrá utilizar transacciones. Los objetos **Recordset** basados en tablas creadas con otros productos de base de datos, puede que no admitan transacciones. Por ejemplo, no puede utilizar transacciones en un **Recordset** basado en una tabla Paradox. En este caso, la propiedad **Transactions** tendrá el valor **False**. Si los objetos **Database** o **Recordset** no admiten transacciones, los métodos se ignorarán y no se producirán errores.

No puede anidar transacciones si está teniendo acceso a orígenes de datos ODBC mediante el motor de base de datos Microsoft Jet.

Notas

- Con frecuencia, puede mejorar el rendimiento de su aplicación interrumpiendo operaciones que necesitan tener acceso a disco en bloques de transacciones. Estos crea un búfer de sus operaciones y puede reducir significativamente las veces a las que se tiene acceso al disco.
- En un espacio de trabajo Microsoft Jet, las transacciones se registran en un archivo que se guarda en el directorio especificado en la variable de entorno TEMP de su equipo. Si el archivo de registro de la transacción agota la capacidad de almacenamiento disponible de su unidad TEMP, el motor de base de datos emitirá un error interceptable. Llegado este punto, si utiliza **CommitTrans**, se ejecutará un número indeterminado de operaciones, pero las operaciones incompletas restantes se perderán y será necesario volver a iniciar la operación. El método **Rollback** libera el registro de transacción y anula todas las operaciones de la transacción.

CancelUpdate (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCancelUpdateC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCancelUpdateX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCancelUpdateA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCancelUpdateS"}

Cancela todas las actualizaciones pendientes del objeto **Recordset**.

Sintaxis

recordset.**CancelUpdate** *tipo*

La sintaxis del método **CancelUpdate** consta de las siguientes partes.

Parte	Descripción
recordset	Una variable de objeto que representa el objeto **Recordset** en el que se cancelan las actualizaciones pendientes.
tipo	Opcional. Una constante que indica el tipo de actualización, como se especifica en Valores (sólo espacios de trabajo ODBCDirect).

Valores

Puede utilizar los siguientes valores para el argumento *tipo* sólo si está habilitada la actualización por lotes.

Constante	Descripción
dbUpdateRegular	Predeterminado. Cancela los cambios pendientes que no están en la memoria caché.
dbUpdateBatch	Cancela los cambios pendientes en la memoria cache actualizada.

Comentarios

El método **CancelUpdate** cancela todas las actualizaciones pendientes a causa de una operación **Edit** o **AddNew**. Por ejemplo, si un usuario llama al método **Edit** o **AddNew** sin haber llamado anteriormente al método **Update**, **CancelUpdate** cancelará todos los cambios efectuados después de llamar a **Edit** o **AddNew**.

Compruebe la propiedad **EditMode** del **Recordset** para determinar si existe alguna operación pendiente que sea necesario cancelar.

Nota La utilización del método **CancelUpdate** tiene el mismo efecto que moverse a otro registro sin utilizar el método **Update**, salvo que el registro activo no cambia y algunas propiedades, como **BOF** y **EOF**, no se actualizan.

Clone (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCloneC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCloneX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCloneA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCloneS"}

Crea un objeto **Recordset** duplicado que hace referencia al objeto **Recordset** original.

Sintaxis

Set *duplicado* = *original*.**Clone**

La sintaxis del método **Clone** utiliza estos argumentos.

Argumento	Descripción
duplicado	Una variable de objeto que identifica el objeto **Recordset** duplicado que ha creado.
original	Una variable de un tipo de datos objeto que identifica al objeto **Recordset** que quiere duplicar.

Comentarios

Utilice el método **Clone** para crear múltiples objetos **Recordset** duplicados. Cada objeto **Recordset** puede tener su propio registro activo. El uso de **Clone** por sí solo no cambia los datos de los objetos o de sus estructuras subyacentes. Utilizando el método **Clone**, puede compartir marcadores entre dos o más objetos **Recordset**, ya que sus marcadores son intercambiables.

Puede utilizar **Clone** cuando desee ejecutar una operación en un **Recordset** que requiera múltiples registros activos. Esto es más rápido y eficaz que crear un segundo objeto **Recordset**. Cuando se crea un **Recordset** mediante el método **Clone**, carece inicialmente de un registro activo. Para hacer activo un registro antes de utilizar el objeto **Recordset** especificado por *duplicado*, debe establecer la propiedad **Bookmark** o utilizar uno de los métodos Move, uno de los métodos Find o el método **Seek**.

El uso del método **Close** en el objeto original o en el duplicado no afecta al otro objeto. Por ejemplo, si utiliza **Close** en el **Recordset** original se cerrará el duplicado.

Notas

- Cerrando una copia del objeto **Recordset** en una transacción pendiente provocará una operación **Rollback** implícita.
- Cuando se utiliza en un objeto **Recordset** tipo Table en un espacio de trabajo Microsoft Jet, el valor de la propiedad **Index** no se duplica en la nueva copia del **Recordset**. Tiene que copiar manualmente el valor de la propiedad **Index**.
- Puede utilizar el método **Clone** con objetos **Recordset** de tipo Forward-only sólo en un espacio de trabajo ODBCDirect.

Close (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCloseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCloseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCloseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCloseS"}

Cierra un objeto DAO.

Sintaxis

objeto.**Close**

El marcador de posición *objeto* es una variable de objeto que representa un objeto **Connection**, **Database**, **Recordset** o **Workspace** abierto.

Comentarios

Cerrar un objeto abierto lo elimina de la colección a la que está añadido. Los intentos de cerrar el área de trabajo predeterminada se ignorarán.

Si el objeto **Connection**, **Database**, **Recordset** o **Workspace** llamado por *objeto* está cerrado cuando utiliza **Close** se produce un error en tiempo de ejecución.

Precaución Si sale de un procedimiento que declara objetos **Connection**, **Database** o **Recordset** y la base de datos esté cerrada, los cambios no guardados se perderán, todas las transacciones pendientes se anularán y se anularán todas las modificaciones pendientes de los datos.

Si intenta cerrar un objeto **Connection** o **Database** mientras hay algún objeto **Recordset** abierto, estos objetos **Recordset** se cerrarán y las actualizaciones o modificaciones pendientes quedarán anuladas. Si intenta cerrar un objeto **Workspace** mientras hay algún objeto **Connection** o **Database** abierto, los objetos **Connection** y **Database** se cerrarán, el cual cerrará sus objetos **Recordset**.

La utilización del método **Close** en un objeto **Recordset** original o duplicado no afecta al otro objeto **Recordset**.

Para eliminar objetos de colecciones que se pueden actualizar distintas de las colecciones **Connections**, **Databases**, **Recordsets** y **Workspaces**, utilice el método **Delete** en estas colecciones. No puede agregar un miembro nuevo a las colecciones **Containers**, **Documents** y **Errors**.

Una alternativa al método **Close** es establecer el valor de una variable de objeto a **Nothing** (`Set dbsTemp = Nothing`).

CompactDatabase (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCompactDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCompactDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCompactDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCompactDatabaseS"}

Copia, compacta y ofrece la posibilidad de cambiar la versión, la secuencia de ordenación y la codificación. (sólo espacio de trabajo Microsoft Jet).

Sintaxis

DBEngine.CompactDatabase *antiguabasededatos*, *nuevabasededatos*, *escenario*, *opciones*, *contraseña*

La sintaxis del método **CompactDatabase** tiene los siguientes argumentos.

Argumento	Descripción
antiguabasededatos	Una **String** que identifica una base de datos existente y cerrada. Puede ser una ruta completa y un nombre de archivo, como `"C:\db1.mdb"`. Si el nombre de archivo tiene una extensión, deberá especificarla. Si su red lo admite, también puede especificar una ruta de red, como `"\\server1\share1\dir1\db1.mdb"`.
nuevabasededatos	Un tipo de datos **String** que es la ruta completa de la base de datos compactada que va a crear. También puede especificar una ruta de acceso de red al igual que con *antiguabasededatos*. No puede usar el argumento *nuevabasededatos* para especificar el mismo archivo de base de datos que *antiguabasededatos.*

escenario	Opcional. Un tipo de datos **Variant** que es una expresión de cadena que se utiliza para especificar la secuencia de ordenación para crear *nuevabasededatos*, como se especifica en Opciones. Si omite este argumento, el escenario de la *nuevabasededatos* será el mismo que el de la *antiguabasededatos*.
	También puede crear una contraseña para *nuevabasededatos* concatenando la cadena de la contraseña (que comienza con `";pwd="`) con una constante del argumento *escenario*, como este:
	`dbLangSpanish & ";pwd=NuevaContraseña"`
	Si desea utilizar el mismo *escenario* como *antiguabasededatos* (el valor predeterminado), pero especificar una contraseña nueva, simplemente escriba una contraseña en *escenario*:
	`";pwd=NuevaContraseña"`

opciones	Opcional. Un valor entero que indica una o más opciones, según se especifica en Opciones. Puede combinar opciones sumando las correspondientes constantes.
contraseña	Opcional. Un tipo de datos **Variant** que es una expresión de cadena que contiene una contraseña, si la base de datos está protegida con contraseña.

La cadena ";pwd=" debe preceder a la propia contraseña. Si incluye una valor de contraseña en *escenario*, este valor se ignora.

Valores

Puede utilizar una de las siguientes constantes para el argumento *escenario* para especificar la propiedad **CollatingOrder** del texto para las comparaciones de cadenas.

Constante	Secuencia de ordenación
dbLangGeneral	Inglés, Alemán, Francés, Portugués, Italiano y Español moderno
dbLangArabic	Árabe
dbLangChineseSimplified	Chino simplificado
dbLangChineseTraditional	Chino tradicional
dbLangCyrillic	Ruso
dbLangCzech	Checo
dbLangDutch	Holandés
dbLangGreek	Griego
dbLangHebrew	Hebreo
dbLangHungarian	Húngaro
dbLangIcelandic	Islandés
dbLangJapanese	Japonés
dbLangKorean	Coreano
dbLangNordic	Lenguas nórdicas (sólo motor de base de datos Microsoft Jet versión 1.0)
dbLangNorwDan	Noruego y Danés
dbLangPolish	Polaco
dbLangSlovenian	Esloveno
dbLangSpanish	Español tradicional
dbLangSwedFin	Sueco y Finlandés
dbLangThai	Tailandés
dbLangTurkish	Turco

Puede utilizar una de las siguientes constantes en el argumento *opciones* para especificar si desea o no codificar la base de datos mientras se compacta.

Constante	Descripción
dbEncrypt	Codifica la base de datos durante la compactación.
dbDecrypt	Descodifica la base de datos durante la compactación.

Si omite una constante de codificación o si incluye a la vez **dbDecrypt** y **dbEncrypt**, *nuevabasededatos* tendrá la misma codificación que *antiguabasededatos.*

Puede usar una de las siguientes constantes en el argumento *opciones* para especificar la versión del formato de los datos para la base de datos compactada. Esta constante afecta sólo a la versión del formato de datos de *nuevabasededatos* y no afecta a la versión de ninguno de los objetos definidos por Microsoft Access, como formularios e informes.

Constante	Descripción
dbVersion10	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 1.0 durante la compactación.

DbVersion11	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 1.1 durante la compactación.
DbVersion20	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 2.0 durante la compactación.
DbVersion30	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 3.0 (compatible con la versión 3.5) durante la compactación.

Sólo puede especificar una constante de versión. Si omite una constante de versión, *nuevabasededatos* tendrá la misma versión que *antiguabasededatos*. Sólo puede compactar *nuevabasededatos* a una versión igual o posterior a la de *antiguabasededatos.*

Comentarios

Al cambiar datos de una base de datos, el archivo de base de datos puede fragmentarse y utilizar más espacio en disco del necesario. Regularmente, puede usar el método **CompactDatabase** en la base de datos para desfragmentar el archivo de base de datos. La base de datos compactada suele ser más pequeña y ejecutarse con más rapidez. También puede cambiar la secuencia de ordenación, la codificación o la versión del formato de datos, mientras copia y compacta la base de datos.

Tiene que cerrar *antiguabasededatos* antes de compactarla. En un entorno multiusuario, los demás usuarios no pueden tener abierta *antiguabasededatos* mientras usted la compacta. Si *antiguabasededatos* no está cerrada o no se encuentra disponible para su uso exclusivo, se producirá un error.

Puesto que **CompactDatabase** crea una copia de la base de datos, deberá disponer de espacio suficiente en disco para la base de datos original y la duplicada. La operación de compactación fracasará si no hay suficiente espacio disponible en disco. La base de datos duplicada *nuevabasededatos* no tiene por qué estar en el mismo disco que *antiguabasededatos.* Después de compactar una base de datos, puede eliminar el archivo *antiguabasededatos* y cambiar el nombre del archivo compactado *nuevabasededatos* por el nombre del archivo original.

El método **CompactDatabase** copia todos los datos y valores de permisos de seguridad de la base de datos especificada en *antiguabasededatos* a la base de datos especificada en *nuevabasededatos*.

Si utiliza el método **CompactDatabase** para convertir una base de datos versión 1.*x* a una versión 2.5 o 3.*x*, solamente las aplicaciones que utilizan las versiones de Microsoft Jet 2.5 o 3.*x* pueden abrir la base de datos convertida.

Nota En un espacio de trabajo ODBCDirect, que utiliza el método **CompactDatabase** no devuelve un error, pero, en su lugar, carga el motor de base de datos Microsoft Jet en memoria.

Precaución Debido a que el método **CompactDatabase** no convertirá objetos Microsoft Access, no es recomendable utilizar **CompactDatabase** para convertir una base de datos que contenga dichos objetos. Para convertir una base de datos que contenga objetos Microsoft Access, en el menú **Herramientas**, elija **Utilidades de la base de datos** y después haga clic en **Convertir base de datos**.

CopyQueryDef (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCopyQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCopyQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCopyQueryDefA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCopyQueryDefS"}

Devuelve un objeto **QueryDef** que es una copia del **QueryDef** utilizado para crear el objeto **Recordset** representado por el marcador de posición *recordset* (sólo espacio de trabajo Microsoft Jet).

Sintaxis

Set *querydef* = *recordset*.**CopyQueryDef**

La sintaxis del método **CopyQueryDef** consta de las siguientes partes:

Argumento	Descripción
querydef	Una variable de objeto que representa la copia de un objeto **QueryDef** que desea crear.
recordset	Una variable de objeto que representa el objeto **Recordset** creado con el objeto **QueryDef** original.

Comentarios

Puede utilizar el método **CopyQueryDef** para crea un nuevo **QueryDef** que es un duplicado del **QueryDef** utilizado para crear el **Recordset**.

Si no se utilizó un **QueryDef** para crear este **Recordset**, se producirá un error. Primero debe abrir un objeto **Recordset** con el método **OpenRecordset** antes de utilizar el método **CopyQueryDef**.

Este método es útil cuando crea un objeto **Recordset** desde otro **QueryDef**, pasa el objeto **Recordset** a una función y la función debe volver a crear el equivalente SQL de la consulta, por ejemplo, para modificarlo.

CreateDatabase (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateDatabaseS"}

Crear un nuevo objeto **Database**, guarda la base de datos en el disco y devuelve un objeto **Database** abierto (sólo espacio de trabajo Microsoft Jet).

Sintaxis

Set *basededatos* = *espaciodetrabajo*.**CreateDatabase (***nombre*, *escenario*, *opciones***)**

La sintaxis del método **CreateDatabase** utiliza estos argumentos.

Argumento	Descripción
basededatos	Una variable de objeto que hace referencia al objeto **Database** que va a crear.
espaciodetrabajo	Una variable de objeto que representa el objeto **Workspace** existente que contendrá la base de datos. Si omite *espaciodetrabajo*, **CreateDatabase** utiliza el **Workspace** predeterminado.
nombre	Un tipo de datos **String** de hasta 255 caracteres de longitud que es el nombre del archivo de base de datos que esta creando. Puede ser la ruta de acceso completa y el nombre del archivo, como `"C:\db1.mdb"`. Si no proporciona un nombre de extensión de archivo, se anexa .mdb. Si su red lo admite, también puede especificar una ruta de acceso de red, como `"\\serv1\comp1\dir1\db1"`. Sólo puede crear archivos de bases de datos .mdb con este método.

escenario	Una expresión de cadena utilizada para especificar la secuencia de ordenación para la creación de la base de datos, como se especifica en Configuraciones. Si no proporciona este argumento se produce un error.
	También puede crear una contraseña para el objeto **Database** nuevo concatenando la contraseña (que comienza con `";pwd="`) con una constante en el argumento *escenario*, como este:
	`dbLangSpanish & ";pwd=NuevaContraseña"`
	Si desea utilizar el *escenario* predeterminado, pero especificar una contraseña, simplemente escriba una contraseña en el argumento *escenario*:
	`";pwd=NuevaContraseña"`

opciones	Opcional. Una constante o combinación e constantes que indican una o más opciones, como se especifica en Configuraciones. Puede combinar opciones resumiendo las constantes correspondientes.

Valores

Puede utilizar una de las siguientes constantes para el argumento *escenario* para especificar la propiedad **CollatingOrder** del texto para comparaciones de cadenas.

Constante	Secuencia de ordenación

dbLangGeneral	Inglés, Alemán, Francés, Portugués, Italiano y Español moderno
dbLangArabic	Árabe
dbLangChineseSimplified	Chino simplificado
dbLangChineseTraditional	Chino tradicional
dbLangCyrillic	Ruso
dbLangCzech	Checo
dbLangDutch	Holandés
dbLangGreek	Griego
dbLangHebrew	Hebreo
dbLangHungarian	Húngaro
dbLangIcelandic	Islandés
dbLangJapanese	Japonés
dbLangKorean	Coreano
dbLangNordic	Lenguas nórdicas (sólo motor de base de datos Microsoft Jet versión 1.0)
dbLangNorwDan	Noruego y Danés
dbLangPolish	Polaco
dbLangSlovenian	Esloveno
dbLangSpanish	Español tradicional
dbLangSwedFin	Sueco y Finlandés
dbLangThai	Tailandés
dbLangTurkish	Turco

Puede utilizar una o más de las siguientes constantes en el argumento *opciones* para especificar qué versión debería tener el formato de datos y si desea o no codificar la base de datos.

Constante	Descripción
dbEncrypt	Crea una base de datos codificada.
dbVersion10	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 1.0.
dbVersion11	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 1.1.
DbVersion20	Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 2.0.
DbVersion30	(Predeterminado) Crea una base de datos que utiliza el motor de base de datos Microsoft Jet versión 3.0. (compatible con la versión 3.5).

Si omite la constante de codificación, **CreateDatabase** creará una base de datos no codificada. Sólo puede especificar una constante de versión. **CreateDatabase** crea una base de datos que utilizará el motor de base de datos Microsoft Jet versión 3.0.

Comentarios

El método **CreateDatabase** y abre una nueva base de datos vacía y devuelve el objeto **Database**. Usted debe completar su estructura y contenido utilizando un objeto DAO. Si desea hacer una copia parcial o completa de una base de datos ya existente, puede usar el método **CompactDatabase** para hacer una copia que luego podrá personalizar.

CreateField (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateFieldA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateFieldS"}

Crea un nuevo objeto **Field** (sólo espacio de trabajo Microsoft Jet).

Sintaxis

Set *campo* = *objeto*.**CreateField (***nombre*, *tipo*, *tamaño***)**

La sintaxis del método **CreateField** utiliza los siguientes argumentos.

Argumento	Descripción
campo	Una variable de objeto que representa el objeto **Field** que desea crear.
objeto	El nombre de variable del objeto **Index**, **Relation** o **TableDef** que quiere usar para crear el nuevo objeto **Field**.
nombre	Opcional. Un tipo **Variant** (subtipo **String**) que designa de manera única al nuevo objeto **Field**. Consulte la propiedad **Name** para conocer más detalles sobre nombres válidos para **Field**.
tipo	Opcional. Una constante que determina el tipo de datos del nuevo objeto **Field**. Consulte la propiedad **Type** para conocer los tipos de datos válidos.
tamaño	Opcional. Un tipo de datos **Variant** (subtipo **Integer**) que indica el tamaño máximo, en bytes, de un objeto **Field** que contiene texto. Consulte la propiedad **Size** para conocer los valores válidos de *tamaño*. Este argumento se ignora en campos numéricos y de anchura fija.

Comentarios

Puede utilizar el método **CreateField** para crear un campo nuevo, también para especificar el nombre, tipo de datos y tamaño del campo. Si omite uno o más de los argumentos opcionales cuando utilice **CreateField**, podrá usar una instrucción de asignación apropiada para establecer o restablecer la propiedad correspondiente antes de agregar el nuevo objeto a una colección Después de agregar el nuevo objeto, podrá cambiar algunos valores de la propiedad, pero no todos. Consulte los temas relativos a las propiedades individuales para conocer más detalles.

Los argumentos *tipo* y *tamaño* se aplican sólo a los objetos **Field** de un objeto **TableDef**. Estos argumentos se ignoran cuando un objeto **Field** se asocia con objetos **Index** o **Relation**.

Si el *nombre* hace referencia a un objeto que ya es miembro de la colección, se producirá un error interceptable cuando utilice el método **Append**.

Para eliminar un objeto **Field** de una colección **Fields**, utilice el método **Delete** en dicha colección. No se puede eliminar un objeto **Field** de un objeto **TableDef** de una colección **Fields** una vez que se ha creado un índice que hace referencia al campo.

CreateGroup (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateGroupX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateGroupA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateGroupS"}

Crea un nuevo objeto **Group** (sólo espacio de trabajo Microsoft Jet).

Sintaxis

Set *grupo* = *objeto***.CreateGroup (***nombre*, *pid***)**

La sintaxis del método **CreateGroup** utiliza los siguientes argumentos.

Argumento	Descripción
grupo	Un tipo variable de objeto que representa el objeto **Group** que desea crear.
objeto	El nombre de la variable del objeto **User** o **Workspace** que utilizará para crear el nuevo objeto **Group**.
nombre	Opcional. Un tipo **Variant** (subtipo **String**) que designa de manera única al nuevo objeto **Group**. Consulte la propiedad **Name** para conocer detalles sobre nombres válidos para **Group**.
pid	Opcional. Un tipo de datos **Variant** (subtipo **String**) que contiene el PID de una cuenta de grupo. El identificador debe contener de 4 a 20 caracteres alfanuméricos. Consulte la propiedad **PID** para conocer más información sobre identificadores personales válidos.

Comentarios

Puede utilizar el método **CreateGroup** para crear un objeto **Group** nuevo para un **User** o **Workspace**. Si omite uno o más argumentos opcionales al utilizar **CreateGroup**, podrá usar una instrucción de asignación apropiada para establecer o restablecer la propiedad correspondiente antes de agregar el nuevo objeto a la colección. Después de agregar el objeto, podrá cambiar algunos valores de la propiedad, pero no todos. Consulte los temas relativos a las propiedades individuales para conocer más detalles al respecto.

Si *nombre* hace referencia a un objeto que ya es miembro de la colección, se producirá un error interceptable cuando utilice el método **Append**.

Para eliminar un objeto **Group** de una colección, utilice el método **Delete** en la colección **Groups**.

CreateIndex (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateIndexS"}

Crea un nuevo objeto **Index** (sólo espacio de trabajo Microsoft Jet).

Sintaxis

Set *índice* = *TableDef***.CreateIndex (***nombre***)**

La sintaxis del método **CreateIndex** utiliza los siguientes argumentos.

Argumento	Descripción
índice	Un tipo variable de objeto que representa el índice que desea crear.
TableDef	El nombre de variable del objeto **TableDef** que utilizará para crear el nuevo objeto **Index**.
nombre	Opcional. Un tipo **Variant** (subtipo **String**) que designa de manera única al nuevo objeto **Index**. Consulte la propiedad **Name** para conocer más detalles sobre nombres válidos para **Index**.

Comentarios

Puede utilizar el método **CreateIndex** para crear un nuevo objeto **Index** de un objeto **TableDef**. Si omite el argumento opcional *nombre* cuando utilice **CreateIndex**, podrá utilizar una instrucción de asignación apropiada para establecer o restablecer la propiedad **Name** antes de agregar el nuevo objeto a la colección. Después de agregar el objeto, podrá establecer su propiedad **Name**, dependiendo del tipo de objeto en el que resida la colección **Indexes**. Consulte el tema relativo a la propiedad **Name** para conocer más detalles al respecto.

Si *nombre* hace referencia a un objeto que ya es miembro de la colección, se producirá un error interceptable cuando utilice el método **Append**.

Para eliminar un objeto **Index** de una colección, utilice el método **Delete** en dicha colección.

CreateProperty (Método)

Crea un nuevo objeto **Property** (sólo espacios de trabajo Microsoft Jet).

Sintaxis

Set *propiedad* = *objeto***.CreateProperty (***nombre*, *tipo*, *valor*, *DDL***)**

La sintaxis del método **CreateProperty** utiliza los siguientes argumentos.

Argumento	Descripción
propiedad	Un variable de objeto que representa el objeto **Property** que desea crear.
objeto	El nombre de variable del objeto **Database**, **Field**, **Index**, **QueryDef**, **Document** o **TableDef** que utilizará para crear el nuevo objeto **Property**.
nombre	Opcional. Un tipo de datos **Variant** (subtipo **String**) que designa de manera única al nuevo objeto **Property**. Consulte la propiedad **Name** para conocer más detalles sobre los nombres válidos para **Property**.
tipo	Opcional. Una constante que define los tipos de datos del nuevo objeto **Property**. Consulte la propiedad **Type** para conocer más información sobre tipos de datos válidos.
valor	Opcional. Una variable de tipo **Variant** que contiene el valor inicial de la propiedad. Consulte la propiedad **Value** para conocer más detalles al respecto.
DDL	Opcional. Un tipo de datos **Variant** (subtipo **Boolean**) que indica si **Property** es un objeto DDL o si no lo es. El valor predeterminado es **False**. Si *DDL* es **True**, es objeto **Property** no podrá ser modificado ni eliminado a menos que el usuario tenga autorización **dbSecWriteDef**.

Comentarios

Sólo puede crear un objeto **Property** definido por el usuario en la colección **Properties** de un objeto que sea persistente.

Si omite uno o más argumentos opcionales cuando utilice **CreateProperty**, podrá usar una instrucción de asignación apropiada para establecer o restablecer la propiedad correspondiente antes de añadir el nuevo objeto a una colección. Después de añadir el objeto, podrá cambiar algunos valores de propiedad, pero no todos. Consulte los temas de las propiedades **Name**, **Type** y **Value** para obtener más información.

Si *nombre* hace referencia a un objeto que ya es miembro de la colección, se producirá un error interceptable cuando utilice el método **Append**.

Para eliminar un objeto **Property** de la colección, utilice el método **Delete** en la colección **Properties**. No se pueden eliminar las propiedades incorporadas.

Nota Si omite el argumento *DDL*, el valor predeterminado **False** (no DDL). Como no se expone ninguna propiedad correspondiente a DDL, será necesario eliminar y volver a crear el objeto **Property** que vaya a cambiar de DDL a no DDL.

CreateQueryDef (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateQueryDefA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateQueryDefS"}

Crea un nuevo objeto **QueryDef** en un objeto **Connection** o **Database** especificado.

Sintaxis

Set *QueryDef* = *objeto*.**CreateQueryDef (***nombre*, *textosql***)**

La sintaxis del método **CreateQueryDef** utiliza los siguientes argumentos.

Argumento	Descripción
QueryDef	Una variable de objeto que hace referencia al objeto **QueryDef** que desea crear.
objeto	Una variable de objeto que representa un objeto **Connection** o **Database** abierto que contendrá el nuevo objeto **QueryDef**.
nombre	Opcional. Un tipo de datos **Variant** (subtipo **String**) que nombra únicamente al nuevo objeto **QueryDef**.
textosql	Opcional. Un tipo de datos **Variant** (subtipo **String**) que es una instrucción SQL que define la **QueryDef**. Si omite este argumento, puede definir la **QueryDef** estableciendo su propiedad **SQL** antes o después de añadirla a una colección.

Comentarios

En un espacio de trabajo Microsoft Jet, si proporciona algo distinto de una cadena de longitud cero para el nombre cuando crea un objeto **QueryDef**, el objeto **QueryDef** resultante se anexa automáticamente a la colección **QueryDefs**. En un espacio de trabajo ODBCDirect, los objetos **QueryDef** son siempre temporales.

En un espacio de trabajo ODBCDirect, el argumento *textosql* puede especificar una instrucción SQL o un procedimiento almacenado de Microsoft SQL Server y sus parámetros.

Si el objeto especificado por *nombre* es ya miembro de la colección **QueryDefs**, se producirá un error interceptable. Se puede crear una **QueryDef** temporal utilizando una cadena de longitud cero para el argumento *nombre* cuando ejecute el método **CreateQueryDef**. Puede establecer la propiedad **Name** de la **QueryDef** recién creada a una cadena de longitud cero (""). Temporalmente los objetos **QueryDef** son útiles si quiere utilizar repetidamente instrucciones SQL dinámicas sin tener que crear ningún objeto nuevo permanente el la colección **QueryDefs**. No puede anexar un objeto **QueryDef** temporal a ninguna colección porque una cadena de longitud cero no es un nombre válido para un objeto **QueryDef** permanente. Se pueden establecer las propiedades **Name** y **SQL** de la **QueryDef** recién creada y posteriormente añadir la **QueryDef** a la colección **QueryDefs**.

Para ejecutar una instrucción SQL en un objeto **QueryDef**, utilice el método **Execute** u **OpenRecordset**.

Utilizar un objeto **QueryDef** es la mejor manera de realizar consultas de paso a través de SQL con bases de datos ODBC.

Para eliminar un objeto **QueryDef** de una colección **QueryDefs** en una base de datos Microsoft Jet, utilice el método **Delete** en la colección. Para una base de datos ODBCDirect, utilice el método **Close** en el objeto **QueryDef**.

CreateRelation (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateRelationX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateRelationA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateRelationS"}

Crea un nuevo objeto **Relation** (sólo espacios de trabajo Microsoft Jet).

Sintaxis

Set *relación* = *basededatos*.**CreateRelation (***nombre*, *tabla*, *tablaexterna*, *atributos***)**

La sintaxis del método **CreateRelation** utiliza los siguientes argumentos.

Argumentos	Descripción
relación	Una variable de objeto que representa el objeto **Relation** que desea crear.
basededatos	El nombre de variable del objeto **Database** que se quiere utilizar para crear el nuevo objeto **Relation**.
nombre	Opcional. Un tipo de datos **Variant** (subtipo **String**) que designa de manera única al nuevo objeto **Relation**. Consulte la propiedad **Name** para conocer más detalles sobre nombre válidos para **Relation**.
tabla	Opcional. Un tipo de datos **Variant** (subtipo **String**) que da nombre a la tabla principal de la relación. Si la tabla no existe antes de añadir el objeto **Relation**, se producirá un error interceptable.
tablaexterna	Opcional. Un tipo de datos **Variant** (subtipo **String**) que le da nombre a la tabla externa de la relación. Si la tabla no existe antes de añadir el objeto **Relation**, se producirá un error interceptable.
atributos	Opcional. Una constante o combinación de constantes que contiene información acerca del tipo de relación. Consulte la propiedad **Attributes** para conocer más detalles al respecto.

Comentarios

El objeto **Relation** traslada información al motor de base de datos Microsoft Jet sobre la relación entre campos de dos objetos **TableDef** o **QueryDef**. Puede proporcionar integridad referencial por medio de la propiedad **Attributes**.

Si omite uno o más de los argumentos opcionales cuando use **CreateRelation**, podrá usar una instrucción de asignación apropiada para establecer o restablecer la propiedad correspondiente antes de añadir el nuevo objeto a una colección. Después de añadir el objeto, no podrá cambiar ninguno de los valores de la propiedad. Consulte los temas relativos a las propiedades individuales para conocer más detalles al respecto.

Antes de utilizar el método **Append** en un objeto **Relation**, debe añadir los objetos **Field** apropiados para definir las tablas de relación de las claves principal y externa.

Si *nombre* hace referencia a un objeto que ya es miembro de la colección o si los nombres del objeto **Field** proporcionados en la colección **Fields** subordinada son inválidos, se producirá un error interceptable cuando utilice el método **Append**.

No puede establecer o mantener una relación entre una tabla replicada y una tabla local.

Para eliminar un objeto **Relation** de una colección **Relations**, utilice el método **Delete** de la colección.

CreateTableDef (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateTableDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateTableDefA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateTableDefS"}

Crea un nuevo objeto **TableDef** (sólo espacios de trabajo Microsoft Jet).

Sintaxis

Set *TableDef* = *basededatos*.**CreateTableDef (***nombre*, *atributos*, *origen*, *conectar***)**

La sintaxis del método **CreateTableDef** utiliza los siguientes argumentos.

Argumento	Descripción
TableDef	Una variable de objeto que representa el objeto **TableDef** que desea crear.
basededatos	El nombre de variable del objeto **Database** que se utilizará para crear el nuevo objeto **TableDef**.
nombre	Opcional. Un tipo de datos **Variant** (subtipo **String**) que le asigna un nombre único al nuevo objeto **TableDef**. Consulte la propiedad **Name** para conocer más detalles sobre nombres válidos para **TableDef**.
atributos	Opcional. Una constante o combinación de constantes que indica una o más características del nuevo objeto **TableDef**. Consulte la propiedad **Attributes** para conocer más información al respecto.
origen	Opcional. Un tipo de datos **Variant** (subtipo **String**) que contiene el nombre de una tabla de una base de datos externa que es el origen de los datos. La cadena *origen* pasa a ser el valor de la propiedad **SourceTableName** del nuevo objeto **TableDef**.
conectar	Opcional. Un tipo de datos **Variant** (subtipo **String**) que contiene información acerca del origen de una base de datos abierta, una base de datos utilizada en una consulta de paso a través o en una tabla adjunta. Consulte la propiedad **Connect** para más información sobre cadenas de conexión válidas.

Comentarios

Si omite uno o más argumentos opcionales cuando utilice **CreateTableDef**, podrá usar una instrucción de asignación apropiada para establecer o restablecer la propiedad correspondiente antes de añadir el nuevo objeto a una colección. Después de añadir el objeto, podrá cambiar algunas de las propiedades, pero no todas. Consulte los temas relativos a las propiedades individuales para conocer más detalles al respecto.

Si *nombre* hace referencia a un objeto que ya es miembro de la colección o especifica un nombre inválido en el objeto **TableDef** o **Field** que va a añadir, se producirá un error interceptable cuando utilice el método **Append**. Tampoco puede anexar un objeto **TableDef** a la colección **TableDefs** hasta que defina como mínimo un **Field** para el objeto **TableDef**.

Para eliminar un objeto **TableDef** de una colección **TableDefs**, utilice el método **Delete**.

CreateUser (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateUserC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateUserA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateUserS"}

Crea un nuevo objeto **User** (sólo espacios de trabajo Microsoft Jet).

Sintaxis

Set *user* = *objeto*.**CreateUser (***nombre*, *pid*, *contraseña***)**

La sintaxis del método **CreateUser** utiliza los siguientes argumentos.

Argumento	Descripción
user	Una variable de objeto que representa el objeto **User** que desea crear.
objeto	El nombre de variable del objeto **Group** o **Workspace** que quiere utilizar para crear el nuevo objeto **User**.
nombre	Opcional. Un tipo de datos **Variant** (subtipo **String**) que designa de manera única al nuevo objeto **User**. Consulte la propiedad **Name** para conocer más detalles sobre los nombres válidos para **User**.
pid	Opcional. Un tipo de datos **Variant** (subtipo **String**) que contiene el PID de una cuenta de usuario. El identificador debe contener de 4 a 20 caracteres alfanuméricos. Consulte la propiedad **PID** para más información sobre identificadores personales válidos.
contraseña	Opcional. Un tipo de datos **Variant** (subtipo **String**) que contiene la contraseña para el nuevo objeto **User**. La contraseña puede tener hasta 14 caracteres de longitud y puede incluir cualquier carácter excepto el carácter ASCII 0 (nulo). Consulte la propiedad **Password** para más información sobre contraseñas válidas.

Comentarios

Si omite uno o más argumentos opcionales cuando utilice **CreateUser**, podrá usar una instrucción de asignación apropiada para establecer o restablecer la propiedad correspondiente antes de añadir el nuevo objeto a una colección. Después de añadir el objeto, podrá cambiar algunos valores de la propiedad, pero no todos. Consulte los temas de las propiedades **PID**, **Name** y **Password** para obtener más información.

Si *nombre* hace referencia a un objeto que ya es miembro de la colección, se producirá un error interceptable cuando utilice el método **Append**.

Para eliminar **User** de una colección **Users**, utilice el método **Delete** en dicha colección.

CreateWorkspace (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCreateWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCreateWorkspaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCreateWorkspaceA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCreateWorkspaceS"}

Crea un nuevo objeto **Workspace**.

Sintaxis

Set *WorkSpace* = **CreateWorkspace(***nombre*, *usuario*, *contraseña*, *tipo***)**

La sintaxis del método **CreateWorkspace** utiliza los siguientes argumentos.

Argumento	Descripción
WorkSpace	Una variable de objeto que representa el objeto **Workspace** que desea crear.
nombre	Un tipo de datos **String** que designa de manera única al nuevo objeto **Workspace**. Consulte la propiedad **Name** para conocer más detalles sobre los nombres válidos para **Workspace**.
usuario	Un tipo de datos **String** que identifica al propietario del nuevo objeto **Workspace**. Consulte la propiedad **UserName** para más información al respecto.
contraseña	Un tipo de datos **String** que contiene la contraseña para el nuevo objeto **Workspace**. La contraseña puede tener una longitud de hasta 14 caracteres y puede incluir cualquier carácter excepto el carácter ASCII 0 (nulo). Consulte la propiedad **Password** para más información sobre contraseñas válidas.
tipo	Opcional. Una constante que indica el tipo de espacio de trabajo, como se describe en Valores.

Valores

Puede utilizar las siguientes constantes para *tipo*.

Constante	Descripción
dbUseJet	Crea un espacio de trabajo Microsoft Jet.
dbUseODBC	Crea un espacio de trabajo ODBCDirect.

Comentarios

Una vez que utilice **CreateWorkspace** para crear un nuevo objeto **Workspace**, se iniciará una sesión **Workspace**, y podrá hacer referencia al objeto **Workspace** en su aplicación.

Los objetos **Workspace** no son permanentes y no puede guardarlos en disco. Una vez que crea un objeto **Workspace**, no puede alterar ninguno de sus valores de propiedad, excepto la propiedad **Name**, la cual puede modificar antes de anexar el objeto **Workspace** a la colección **Workspaces**.

No es necesario añadir el nuevo objeto **Workspace** a una colección antes de poder usarlo. Un objeto **Workspace** recién creado se agrega sólo si será necesario hacer referencia a él por medio de la colección **Workspaces**.

La opción *tipo* determina si el **Workspace** nuevo es un espacio de trabajo Microsoft Jet o de ODBCDirect. Si establece *tipo* a **dbUseODBC** y no tiene creado todavía ningún espacio de trabajo Microsoft Jet el motor de base de datos Microsoft Jet no se cargará en memoria y toda actividad se producirá con el origen de datos ODBC identificado el en objeto **Connection**. Si omite *tipo*, la propiedad **DefaultType** del objeto **DBEngine** determinará a qué tipo de origen de datos está

conectado el **Workspace**. Puede tener abiertos al mismo tiempo espacios de trabajo Microsoft Jet y de ODBCDirect.

Para eliminar **Workspace** de la colección **Workspaces**, cierre todas las bases de datos abiertas y a continuación utilice el método **Close** en el objeto **Workspace**.

Delete (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthdeleteC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthDeleteX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthDeleteA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthDeleteS"}

- Objetos **Recordset**: elimina el registro activo de un objeto **Recordset** de tipo Dynaset o Table. Para espacios de trabajo ODBCDirect, el tipo de controlador determina si los objetos **Recordset** se pueden actualizar y, por tanto, admiten el método **Delete**.
- Colecciones: elimina un objeto persistente almacenado de una colección.

Sintaxis

recordset.**Delete**
colección.**Delete** *nombreobjeto*

La sintaxis del método **Delete** utiliza los siguientes argumentos.

Argumentos	Descripción
recordset	Una variable de objeto que identifica un objeto **Recordset** de tipo Dynaset o Table abierto, que contiene el registro que desea eliminar.
colección	Una variable de objeto que representa una colección de la que se elimina *nombreobjeto*.
nombreobjeto	Un tipo de datos **String** que es el valor de la propiedad **Name** de un objeto existente en una *colección*.

Comentarios

Puede utilizar el método **Delete** para eliminar un registro activo de un objeto **Recordset** o un miembro de una colección, como una tabla almacenada de una base de datos, un campo almacenado de una tabla y un índice de una tabla.

Recordsets

Un objeto **Recordset** debe contener un registro activo antes de que utilice el método **Delete**; en caso contrario se produce un error en tiempo de ejecución.

En objetos **Recordset**, **Delete** elimina el registro activo y lo hace inaccesible. Aunque no pueda modificarlo o utilizarlo, el registro eliminado permanecerá activo. Sin embargo, una vez que se desplace a otro registro no podrá volver a convertir en activo el registro eliminado. Las referencias subsiguientes a un registro eliminado en un **Recordset** no son válidas y producen un error.

Puede recuperar un registro utilizando transacciones y el método **Rollback**.

Si la tabla base es la tabla principal en una relación de eliminación de cascada, al eliminar el registro activo también se eliminarán uno o más registros de una tabla externa.

Nota Para agregar, modificar o eliminar un registro, debe tener un índice único en el registro en el origen de datos de base. Si no es así, se producirá un error "Permiso denegado" en la llamada al método **AddNew**, **Delete** o **Edit** en un espacio de trabajo Microsoft Jet, o se producirá un error "Argumento no válido" el la llamada al método **Update** en un espacio de trabajo ODBCDirect.

Colecciones

Puede utilizar el método **Delete** para eliminar un objeto persistente. Sin embargo, si la colección es una colección **Databases**, **Recordsets** o **Workspaces** (cada una de las cuales se almacena sólo en memoria), los objetos abiertos o activos se eliminarán cerrando ese objeto mediante el método **Close**.

La eliminación de objetos almacenados ocurre inmediatamente, pero debe utilizar el método **Refresh** en cualquier otra colección que se pueda ver afectada por cambios en la estructura de la base de datos.

Cuando elimine un objeto **TableDef** de una colección **TableDefs**, se eliminará la definición de la tabla y los datos de la misma.

La tabla siguiente muestra algunas limitaciones en el uso del método **Delete**. El objeto de la primera columna contiene la colección de la segunda columna. La tercera columna indica cuándo se puede eliminar un objeto, si es que ello es posible (por ejemplo, nunca puede eliminar un objeto **Container** de una colección **Containers** de un objeto **Database**).

Objeto	Colección	¿Cuándo puede utilizar el método Delete?
DBEngine	**Workspaces**	Nunca. Al cerrar los objetos los elimina.
DBEngine	**Errors**	No
Workspace	**Connections**	No. Al cerrar los objetos los elimina.
Workspace	**Databases**	Nunca. Al cerrar los objetos los elimina.
Workspace	**Groups**	Si
Workspace	**Users**	Si
Connection	**QueryDefs**	No
Connection	**Recordsets**	Nunca. Al cerrar los objetos los elimina.
Database	**Containers**	No
Database	**QueryDefs**	Si
Database	**Recordsets**	Nunca. Al cerrar los objetos los elimina.
Database	**Relations**	Si
Database	**TableDefs**	Si
Group	**Users**	Si
User	**Groups**	Si
Container	**Documents**	No
QueryDef	**Fields**	No
QueryDef	**Parameters**	No
Recordset	**Fields**	No
Relation	**Fields**	Sólo cuando el objeto **Relation** es un objeto nuevo no anexado.
TableDef	**Fields**	Sólo cuando el objeto **TableDef** es nuevo y no se ha anexado a la base de datos o cuando la propiedad **Updatable** del objeto **TableDef** se establece a **True**.
TableDef	**Indexes**	Sólo cuando el objeto **TableDef** es nuevo y no se ha anexado a la base de datos o cuando la propiedad **Updatable** del objeto **TableDef** se establece a **True**.
Index	**Fields**	Sólo cuando el objeto **Index** es nuevo y no se ha anexado a la base de datos.
Database, **Field**, **Index**, **QueryDef**, **TableDef**	**Properties**	Sólo cuando la propiedad está definida por el usuario.
DBEngine, **Parameter**, **Recordset**,	**Properties**	No

Workspace

Edit (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthEditC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthEditX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthEditA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthEditS"}

Copia el registro activo de un objeto **Recordset** al búfer de copia para su posterior edición.

Sintaxis

recordset.**Edit**

El *recordset* representa el nombre de un objeto **Recordset** abierto y que se puede actualizar que contiene el registro que desea modificar.

Comentarios

Una vez que utiliza el método **Edit**, los cambios realizados en los campos del registro activo son copiados al búfer de copia. Después de realizar los cambios deseados en el registro, utilice el método **Update** para guardar los cambios.

El registro activo permanece activo después de utilizar el método **Edit**.

Precaución Si modifica un registro y a continuación pasa a otro registro sin utilizar antes **Update**, los cambios se perderán sin previo aviso. Además, si cierra *recordset* o finaliza el procedimiento que declara el **Recordset** o el objeto **Database** o **Connection**, el registro modificado se descarta sin previo aviso.

La utilización de **Edit** produce un error bajo las siguientes condiciones:

- No hay ningún registro activo.
- El objeto **Connection**, **Database** o **Recordset** se abrió de sólo lectura.
- No hay campos que se pueden actualizar en el registro.
- El objeto **Database** o **Recordset** se abrió para uso en modo exclusivo por otro usuario (espacio de trabajo Microsoft Jet).
- Otro usuario ha bloqueado la página que contiene su registro (espacio de trabajo Microsoft Jet).

En un espacio de trabajo Microsoft Jet, cuando el valor de la propiedad **LockEdits** del objeto **Recordset** es **True** (bloqueado de forma pesimista) en un entorno multiusuario, el registro permanece bloqueado desde el momento en que se utiliza **Edit** hasta que se completa la actualización. Si el valor de la propiedad **LockEdits** es **False** (bloqueado de forma optimista), el registro se bloquea y se compara con el registro modificado previamente, justo antes de la actualización en la base de datos. Si el registro hubiera cambiado desde la utilización del método **Edit**, la operación **Update** fracasaría produciendo un error interceptable si utiliza **OpenRecordset** sin especificar **dbSeeChanges**. De forma predeterminada, las bases de datos ODBC conectada de Microsoft Jet y las instalables ISAM siempre utilizan el bloqueo optimista.

En un espacio de trabajo ODBCDirect, una vez que modifica (y utiliza el método **Update** para actualizar) un campo de clave principal de un registro, no puede modificar los campos de ese registro hasta que cierre el objeto **Recordset** y recupere el registro otra vez en una consulta posterior.

Nota Para agregar, modificar o eliminar un registro, debe tener un índice único en el registro en el origen de datos de base. Si no es así, se producirá un error "Permiso denegado" en la llamada al método **AddNew**, **Delete** o **Edit** en un espacio de trabajo Microsoft Jet, o se producirá un error "Argumento no válido" el la llamada al método **Update** en un espacio de trabajo ODBCDirect.

Execute (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthExecuteC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthExecuteX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthExecuteA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthExecuteS"}

Ejecuta una consulta de acciones o una instrucción SQL en un objeto **Connection** o **Database** especificado.

Sintaxis

objeto.**Execute** *origen*, *opciones*
QueryDef.**Execute** *opciones*

La sintaxis del método **Execute** utiliza los siguientes argumentos.

Argumento	Descripción
objeto	Una variable de objeto **Connection** o **Database** en la que se ejecutará la consulta.
QueryDef	El nombre del objeto **QueryDef** cuyo valor de propiedad **SQL** especifica la instrucción SQL que se va a ejecutar.
origen	Un tipo de datos **String** qu es una instrucción SQL o el valor de la propiedad **Name** de un objeto **QueryDef**.
opciones	Opcional. Una constante o una combinación de constantes que determina las características de la integridad de los datos de la consulta, como se especifica en Valores.

Valores

Puede utilizar las siguientes constantes para el argumento *opciones*.

Constante	Descripción
dbDenyWrite	Niega la autorización de escritura a otros usuarios.(sólo espacios de trabajo Microsoft Jet).
dbInconsistent	(Predeterminado) Ejecuta actualizaciones inconsistentes (sólo espacios de trabajo Microsoft Jet).
dbConsistent	Ejecuta actualizaciones consistentes (sólo espacios de trabajo Microsoft Jet).
DbSQLPassThrough	Ejecuta un paso a través de SQL. Obliga a que la instrucción SQL tenga que pasar a un base de datos ODBC para su proceso (sólo espacios de trabajo Microsoft Jet).
DbFailOnError	Anula las actualizaciones si se produce un error.(sólo espacios de trabajo Microsoft Jet).
DbSeeChanges	Genera un error en tiempo de ejecución si otro usuario cambia datos que usted está modificando.(sólo espacios de trabajo Microsoft Jet).
DbRunAsync	Ejecuta la consulta asíncronamente (sólo para objetos ODBCDirect **Connection** y **QueryDef**).
DbExecDirect	Ejecuta la instrucción sin llamar antes a la función API de ODBC **SQLPrepare** (sólo para

objetos ODBCDirect **Connection** y **QueryDef**).

Nota Las constantes **dbConsistent** y **dbInconsistent** se excluyen mutuamente. Puede utilizar una u otra, pero nunca ambas en una instancia determinada de un objeto **OpenRecordset**. El uso de **dbConsistent** y **dbInconsistent** provoca un error.

Comentarios

El método **Execute** sólo es válido para consultas de acciones. Si utiliza **Execute** en otro tipo de consulta, se producirá un error. Puesto que una consulta de acciones no devuelve registros, **Execute** no devuelve ningún **Recordset**. (Ejecutar una consulta de paso a través de SQL en un espacio de trabajo ODBCDirect no devolverá un error si no se devuelve un objeto **Recordset**.)

Utilice la propiedad **RecordsAffected** del objeto **Connection**, **Database** o **QueryDef** para determinar el número de registros afectados por el método **Execute**. Por ejemplo, **RecordsAffected** contiene el número de registros borrados, actualizados o insertados mientras se estaba ejecutando una consulta de acciones. Cuando utilice el método **Execute** para realizar una **QueryDef**, la propiedad **RecordsAffected** del objeto **QueryDef** tendrá el valor correspondiente al número de registros afectados.

En un espacio de trabajo Microsoft Jet, si proporciona una instrucción SQL sintácticamente correcta y tiene los permisos apropiados, el método **Execute** no falla - aunque no se pueda modificar o eliminar ni una sola línea. Por tanto, utilice siempre la opción **dbFailOnError** cuando use el método **Execute** para realizar una consulta de actualización o eliminación. Esta opción genera un error interceptable y anula todos los cambios, si cualquiera de los registros afectados está bloqueado y no se pueden actualizar o eliminar.

Para obtener un mejor rendimiento, en espacio de trabajo Microsoft Jet, especialmente en un entorno multiusuario, anide el método **Execute** en una transacción. Utilice el método **BeginTrans** en el objeto **Workspace** activo y a continuación utilice el método **Execute** y complete la transacción utilizando el método **CommitTrans** en el **Workspace**. Esto evita cambios en el disco y libera los bloqueos que se puedan haber establecido mientras se ejecutaba la consulta.

En un espacio de trabajo ODBCDirect, si incluye la constante **dbRunAsync** opcional, la consulta se ejecuta asíncronamente. Para determinar si un consulta asíncrona se está ejecutando, compruebe el valor de la propiedad **StillExecuting** en el objeto del que se llamó al método **Execute**, utilice el método **Cancel**.

FillCache (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthFillCacheC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthFillCacheX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthFillCacheA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthFillCacheS"}

Rellena todo o parte de la memoria caché local para un objeto **Recordset** que contiene datos de un origen de datos ODBC conectado a Microsoft Jet (sólo origen de datos ODBC conectado a Microsoft Jet).

Sintaxis

recordset.**FillCache** *filas*, *marcadorinicio*

La sintaxis del método **FillCache** utiliza los siguientes argumentos.

Argumento	Descripción
recordset	Una variable de objeto que representa un objeto **Recordset** creado por un origen de datos ODBC, como un objeto **TableDef** que representa una tabla vinculada o un objeto **QueryDef** que proviene de un objeto **TableDef**.
filas	Opcional. Un tipo de datos **Variant** (subtipo **Integer**) que especifica el número de filas para rellenar la memoria caché. Si omite este argumento, el valor estará determinado por el valor de la propiedad **CacheSize**.
marcadorinicio	Opcional. Un tipo de datos **Variant** (subtipo **String**) que establece un marcador. El caché se rellena comenzando por el registro indicado en este marcador. Si omite este argumento, la memoria caché se rellenará comenzando por el registro indicado en la propiedad **CacheStart**.

Comentarios

El caché mejora el rendimiento de una aplicación que recupera o busca datos en un servidor remoto. Un caché es un espacio en la memoria local que conserva los datos recuperados más recientemente del servidor, en el supuesto de que los datos se vuelvan a solicitar mientras se ejecuta la aplicación. Cuando se solicitan los datos, el motor de base de datos Microsoft Jet comprueba en primer lugar si los datos están en la memoria caché, en lugar de buscarlos en el servidor, lo que le llevaría más tiempo. Los datos que no provienen de un origen de datos OBDC no se guardan en la memoria caché.

En lugar de esperar a que la memoria caché se rellene con registros según se buscan, puede rellenarlo de forma explícita en cualquier momento mediante el método **FillCache**. Este es un método más rápido para rellenar la memoria caché, pues **FillCache** busca varios registros de forma inmediata en lugar de hacerlo de uno en uno. Por ejemplo, mientras se muestra una pantalla completa de registros, puede hacer que su aplicación utilice **FillCache** para buscar la siguiente pantalla completa de registros.

Cualquier base de datos Microsoft Jet conectada a ODBC a la que accedan objetos **Recordset** puede tener un caché local. Para crear un caché, abra un objeto **Recordset** desde el origen de datos remoto y a continuación establezca las propiedades **CacheSize** y **CacheStart** del **Recordset**.

Si *filas* e *marcadorinicio* crean un rango que está parcial o totalmente fuera del rango especificado por las propiedades **CacheSize** y **CacheStart**, se ignorará la parte de *recordset* que esté fuera de rango y no se cargará en la memoria caché.

Si **FillCache** solicita más registros que el número restante en el origen de datos remoto, Microsoft Jet

sólo recupera los registros restantes y no se produce ningún error.

Notas

- Los registros recuperados desde la memoria caché no reflejan los cambios efectuados simultáneamente en el origen de datos por otros usuarios.
- **FillCache** sólo recupera los registros que no están ya en la memoria caché. Para forzar una actualización de todos los datos de la memoria caché, establezca la propiedad **CacheSize** del **Recordset** como 0, establézcala como el tamaño de la memoria caché que solicitó originalmente y, a continuación, utilice **FillCache**.

FindFirst, FindLast, FindNext, FindPrevious (Métodos)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthFindFirstC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthFindFirstX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthFindFirstA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthFindFirstS"}

Buscan el primero, último, siguiente o anterior registro de un objeto **Recordset** de tipo Dynaset o Snapshot que cumpla el criterio especificado y lo convierte en el registro activo (sólo espacios de trabajo Microsoft Jet).

Sintaxis

recordset.{**FindFirst** | **FindLast** | **FindNext** | **FindPrevious**} *criterios*

Los métodos **Find** utilizan los siguientes argumentos.

Argumento	Descripción
recordset	Una variable de objeto que representa un objeto **Recordset** existente de tipo Dynaset o Snapshot.
criterios	Un tipo de datos **String** utilizado para localizar el registro. Es como la cláusula **WHERE** en una instrucción SQL, pero sin la palabra WHERE.

Comentarios

Si desea incluir todos los registros en la búsqueda - no sólo aquellos que cumplan un condición específica - utilice los métodos Move para pasar de un registro a otro. Para localizar un registro en un **Recordset** de tipo Table, utilice el método **Seek**.

Si no se localiza un registro que coincida con los criterios, el puntero del registro activo será indeterminado y la propiedad **NoMatch** será **True**. Si *recordset* contiene más de un registro que satisfaga el criterio, **FindFirst** localizará la primera aparición, **FindNext** localizará la segunda, etc.

Cada método **Find** comienza a buscar desde la posición y en la dirección especificadas en la siguiente tabla:

Métodos Find	Comienzo	Dirección de búsqueda
FindFirst	Comienzo del recordset	Fin del recordset
FindLast	Fin del recordset	Comienzo del recordset
FindNext	Registro activo	Fin del recordset
FindPrevious	Registro activo	Comienzo del recordset

Cuando utilice el método **FindLast**, el motor de base de datos Microsoft Jet llenará totalmente el objeto **Recordset** antes de iniciar la búsqueda, en caso de que no se hubiera hecho aún.

Sin embargo, utilizar uno de los métodos **Find** no es lo mismo que utilizar un método **Move**, que simplemente convierte en activo el primero, último, siguiente o anterior registro, sin especificar una condición. Se puede establecer que una operación Move siga a una operación Find.

Compruebe siempre el valor de la propiedad **NoMatch** para determinar si la operación Find ha tenido éxito. Si la búsqueda es exitosa, **NoMatch** será **False**. Si fracasa, **NoMatch** será **True** y el registro activo no estará definido. En este caso, debe volver a colocar el puntero del registro activo en un registro válido, como se muestra en el siguiente ejemplo:

Utilizar los métodos **Find** con conjuntos de registros de ODBC conectados a Microsoft Jet puede no ser eficaz. Es posible que expresar con otras palabras los *criterios* para localizar un registro específico es más rápido, sobre todo cuando se trabaja con un número de registros.

En un espacio de trabajo ODBCDirect, los métodos **Find** y **Seek** no están disponibles para ningún tipo de objeto **Recordset**, porque ejecutar un **Find** o **Seek** mediante una conexión ODBC no es muy

eficaz sobre una red. En vez de ello, debería diseñar la consulta (utilizando el argumento *origen* del método **OpenRecordset**) con la cláusula WHERE apropiada que restringe los registros devueltos a sólo aquellos que cumplen el criterio que desea, en caso contrario utilice un método **Find** o **Seek**.

Cuando trabaje con bases de datos Microsoft Jet conectada a ODBC y hojas de respuesta dinámica grandes, los objetos **Recordset** que podría descubrir que utilizar los métodos **Find** o utilizar la propiedad **Sort** o **Filter** es más lento. Para mejorar el rendimiento, utilice consultas SQL con las cláusulas ORDER BY o WHERE personalizadas, consultas de parámetros o los objetos **QueryDef** que recuperan registros indexados específicos.

Debería utilizar el formato de fechas de EE.UU. (mes-día-año) cuando busque en campos que contienen fechas, incluso si no está utilizando la versión de EE.UU. del motor de base de datos Microsoft Jet; en caso contrario, los datos podrían no encontrarse. utilice la función de Visual Basic **Format** para convertir la fecha. Por ejemplo:

```
rstEmpleados.FindFirst "FechaContratación > #" _
   & Format(mifecha, 'm-d-yy' ) & "#"
```

Si *criterio* se compone de una cadena a string concatenada con un valor no entero y los parámetros del sistema especifican un signo decimal no de EE.UU. como una coma (por ejemplo, `strSQL = "PRECIO > " & lngPrecio`, y `lngPrecio = 125,50`), se produce un error cuando intenta llamar al método. Esto es debido a que durante la concatenación, el número se convertirá en una cadena utilizando el signo decimal predeterminado del sistema y Microsoft Jet SQL sólo acepta signos decimales de EE.UU.

Notas

- Para un mejor rendimiento, el *criterio* debería estar en el formato "*campo* = *valor*" donde *campo* es un campo indexado el la tabla base o "*campo* LIKE *prefijo*" donde *campo* es un campo indexado de la tabla base y *prefijo* es un prefijo de una cadena de búsqueda (Por ejemplo, `"ART*"`).
- En general, para tipos equivalentes de búsquedas, el método **Seek** proporciona mejor rendimiento que los métodos **Find**. Esto asume que los objetos **Recordset** de tipo Table por sí solos pueden satisfacer sus necesidades.

GetChunk (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthGetChunkC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthGetChunkX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthGetChunkA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthGetChunkS"}

Devuelve todo o una parte del contenido de un objeto **Field** de tipo Memo o Long Binary de la colección **Fields** de un objeto **Recordset**.

Sintaxis

Set *variable* = *recordset* **!** *campo***.GetChunk (***desplazamiento*, *númerobytes***)**

La sintaxis del método **GetChunk** utiliza los siguientes argumentos.

Argumento	Descripción
variable	Una variable de tipo **Variant** (subtipo **String**) que recibirá los datos del objeto **Field** designado por *campo.*
recordset	Una variable de objeto que especifica el objeto **Recordset** que contiene la colección **Fields**.
campo	Una variable de objeto que representa un objeto **Field** cuya propiedad **Type** se establece como **dbMemo** (Memo) o **dbLongBinary** (Long Binary).
desplazamiento	Un valor de tipo **Long** igual al número de bytes que se deben omitir antes de iniciar la copia.
númerobytes	Un valor **Long** igual al número de bytes que quiere devolver.

Comentarios

Los bytes devueltos por el método **GetChunk** son asignados a *variable*. Use **GetChunk** para devolver una parte del valor total de los datos de una vez. Puede utilizar el método **AppendChunk** para volver a ensamblar las piezas.

Si *desplazamiento* es 0, **GetChunk** comenzará a copiar desde el primer byte del campo.

Si *númerobytes* es mayor que el número de bytes del campo, **GetChunk** devolverá el número real de bytes sobrantes en el campo.

Precaución Utilice un campo de tipo Memo para texto y ponga datos binarios sólo en los campos de tipo Long Binary. De lo contrario obtendrá resultados no deseables.

GetRows (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthGetRowsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthGetRowsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthGetRowsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthGetRowsS"}

Recupera múltiples filas de un objeto **Recordset**

Sintaxis

Set *varmatriz* = *recordset*.**GetRows (***númfilas***)**

La sintaxis del método **GetRows** utiliza los siguientes argumentos.

Argumento	Descripción
varmatriz	Una variable de tipo **Variant** que almacena los datos devueltos.
recordset	Una variable objeto que identifica un objeto **Recordset**.
númfilas	Una variable de tipo **Variant** que es igual al número de filas a recuperar.

Comentarios

Utilice el método **GetRows** para copiar registros de un **Recordset**. **GetRows** devuelve una matriz de dos dimensiones. El primer índice identifica el campo y el segundo identifica el número de fila. Por ejemplo, `intCampo` identifica el campo e `intRegistro` identifica el número de fila:

```
mvarRegistros(intCampo, intRegistro)
```

Para que el valor del primer campo sea devuelto en la segunda fila, puede hacer lo siguiente:

```
campo1 = mvarRegistros(0,1)
```

Para que el valor del segundo campo sea devuelto en la primera fila, puede hacer lo siguiente:

```
campo2 = mvarRegistros(1,0)
```

La variable `mvarRegistros` crea automáticamente una matriz de dos dimensiones cuando **GetRows** devuelve los datos.

Si necesita más filas de las que hay disponibles, entonces **GetRows** devuelve sólo el número de filas disponibles. Puede utilizar la función **UBound** de Visual Basic para determinar cuántas filas recupera **GetRows** en realidad, porque la matriz es diseñada para ajustar el número de filas devueltas. Por ejemplo, si devuelve los resultados en una variable **Variant** llamada `varA`, puede utilizar lo siguiente para determinar cuántas filas son devueltas en realidad:

```
numDevuelto = UBound(varA,2) + 1
```

Necesita utilizar `"+ 1"` porque la primera fila devuelta es en el elemento 0 de la matriz. El número de filas que puede recuperar está limitado por la cantidad de memoria disponible. No debe utilizar **GetRows** para recuperar una tabla entera en una matriz si se trata de una tabla grande.

Dado que el método **GetRows** devuelve todos los campos del objeto **Recordset** en la matriz, incluyendo campos Memo y Long Binary, es posible que quiera realizar una consulta para limitar la cantidad de campos devueltos.

Después de llamar al método **GetRows**, el registro activo se situará en la siguiente fila sin lectura. Es decir, **GetRows** tiene un efecto de posicionamiento equivalente a **Move** *númfilas*.

Si intenta recuperar todas las filas utilizando llamadas al método **GetRows** múltiples, use la propiedad **EOF** para asegurarse de que se encuentra al final del objeto **Recordset**. **GetRows** devuelve una cantidad inferior a la solicitada si llega al final del objeto **Recordset**, o si no puede recuperar una fila en el rango solicitado. Por ejemplo, si intenta recuperar 10 registros pero no puede recuperar el quinto, **GetRows** devuelve cuatro registros y hace que el quinto registro sea el activo.

Esto no generará un error en tiempo de ejecución. Esto puede ocurrir si otro usuario elimina un registro en un objeto **Recordset** de tipo Dynaset. Utilice el ejemplo para ver una demostración de cómo hacer esto.

Idle (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthIdleC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthIdleX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthIdleA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthIdleS"}

Suspende el procesamiento de datos, permitiendo al motor de base de datos Microsoft Jet para completar las tareas pendientes, tales como la optimización de memoria o las esperas de página (sólo espacios de trabajo Microsoft Jet).

Sintaxis

DBEngine.Idle [**dbRefreshCache**]

Comentarios

El método **Idle** permite al motor de base de datos Microsoft realizar tareas en segundo plano que no han podido ser actualizadas debido a la intensidad del procesamiento de datos. Esto ocurre a menudo en entornos multiusuario y multitarea que no disponen de tiempo suficiente de proceso en segundo plano para mantener actualizados todos los registros en un **Recordset**.

Normalmente, se eliminan los bloqueos de lectura y los datos de los objetos **Recordset** de tipo Dynaset locales se actualizan sólo cuando no se están realizando otras tareas (incluyendo movimientos del *mouse*). Si utiliza periódicamente el método **Idle**, permitirá al motor de base de datos de Microsoft Jet procesar tareas en segundo plano liberando los bloqueos de lectura que ya no se necesiten.

Si especifica **dbRefreshCache** como argumento fuerza cualquier escritura pendiente a archivos .mdb y actualiza la memoria con los datos más actuales del archivo .mdb.

No necesita utilizar este método en entornos de un solo usuario a menos que se estén ejecutando múltiples instancias de una aplicación. El método **Idle** puede aumentar el rendimiento en entornos multiusuario porque fuerza al motor de base de datos a escribir los datos en el disco, liberando los bloqueos de memoria.

Nota También puede liberar bloqueos de memoria haciendo que las operaciones sean parte de una transacción.

MakeReplica (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthMakeReplicaC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthMakeReplicaX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthMakeReplicaA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthMakeReplicaS"}

Crea una nueva réplica de otra base de datos replicable (sólo espacios de trabajo Microsoft Jet).

Sintaxis

basededatos.**MakeReplica** *réplica*, *descripción*, *opciones*

La sintaxis del método **MakeReplica** tiene los siguientes argumentos.

Argumentos	Descripción
basededatos	Una variable de objeto que identifica una **Database** existente replicable.
réplica	Un tipo de datos **String** que contiene la ruta y el nombre del archivo de la nueva réplica. Si *réplica* es un nombre de archivo existente se producirá un error.
descripción	Un tipo de datos **String** que describe la réplica que se va a crear.
opciones	Opcional. Una constante o combinación de constantes que especifica las características de la réplica que ha creado, como se especifica en Valores.

Valores

Puede utilizar una o varias de las siguientes constantes como argumento en *opciones*.

Constantes	Descripción
dbRepMakePartial	Crea una réplica parcial.
dbRepMakeReadOnly	Impide que los usuarios modifiquen los objetos replicables de una nueva réplica; sin embargo, cuando sincroniza la nueva réplica con otro miembro del conjunto de réplicas, diseña y cambia los datos que serán propagados a la nueva réplica.

Comentarios

Una réplica parcial creada nuevamente deberá tener todas las propiedades **ReplicaFilter** establecidas como **False**, significa que los datos no estarán en las tablas.

Move (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthMoveC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthMoveX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthMoveA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthMoveS"}

Mueve la posición del registro activo de un objeto **Recordset**.

Sintaxis

recordset.**Move** *filas*, *inicio*

La sintaxis del método **Move** tiene los siguientes argumentos.

Argumentos	Descripción
recordset	Una variable objeto que identifica el objeto **Recordset** cuando la posición del registro activo se ha movido.
filas	Un valor de tipo **Long** con signo que especifica el número de filas que moverá la posición. Si *filas* es mayor que 0, la posición se moverá hacia adelante (hacia el final del archivo). Si *filas* es menor que 0, la posición se moverá hacia atrás (hacia el comienzo del archivo).
iniciomarcador	Opcional. Un valor de tipo **Variant** (subtipo **String**) que identifica un marcador. Si especifica *iniciomarcador,* el movimiento comenzará en relación con este marcador. Si no, el método **Move** comienza desde el registro activo.

Comentarios

Si utiliza el método **Move** para posicionar el registro antes del primer registro, la posición del registro activo se desplaza al comienzo del archivo. Si el objeto **Recordset** no contiene registros y la propiedad **BOF** es **True**, al utilizar este método para retroceder se producirá un error.

Si utiliza el método **Move** para posicionar el registro después del último registro, la posición del registro activo se desplaza al final del archivo. Si el **Recordset** no contiene registros y la propiedad **EOF** es **True**, al utilizar este método para avanzar se producirá un error.

Si las propiedades **BOF** o **EOF** son **True** e intenta utilizar el método **Move** sin un marcador válido se producirá un error en tiempo de ejecución.

Notas

- Cuando utiliza **Move** en un objeto **Recordset** de tipo Forward-only, el argumento *filas* debe ser un entero positivo y no se permitirán marcadores. Esto significa que sólo se puede mover hacia adelante.
- Para convertir en registro activo el registro primero, último, siguiente o anterior de un **Recordset**, utilice el método **MoveFirst, MoveLast, MoveNext o MovePrevious**.
- Utilizar **Move** con *filas* igual a 0 es un camino fácil para recuperar los datos para el registro activo. Esto es útil si se quiere asegurar de que los datos de las tablas de la base de datos son los más actuales. Esto también cancelará cualquier llamada pendiente al método **Edit** o **AddNew**.

MoveFirst, MoveLast, MoveNext, MovePrevious (Métodos)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthMoveFirstC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthMoveFirstX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthMoveFirstA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthMoveFirstS"}

Mueven al registro primero, último, siguiente o anterior de un objeto **Recordset** y lo convierten en el registro activo.

Sintaxis

recordset.{**MoveFirst** | **MoveLast** [**dbRunAsync**] | **MoveNext** | **MovePrevious**}

El marcador de posición *recordset* es una variable de objeto que representa un objeto **Recordset** abierto.

Comentarios

Utilice el método **Move** para ir de un registro a otro sin aplicar una condición.

Precaución Si modifica el registro activo, utilice el método **Update** para guardar los cambios antes de ir a otro registro. Si va a otro registro sin actualizar, los cambios se perderán sin previo aviso.

Al abrir un **Recordset**, el primer registro está activo y la propiedad **BOF** es **False**. Si el **Recordset** no contiene registros la propiedad **BOF** es **True** y no habrá ningún registro activo.

Si el primer o el último registro está activo cuando utiliza **MoveFirst** o **MoveLast**, el registro activo no cambia.

Si utiliza **MovePrevious** cuando el primer registro está activo, la propiedad **BOF** es **True** y no habrá ningún registro activo. Si vuelve a utilizar **MovePrevious** se producirá un error y **BOF** será **True**.

Si utiliza **MoveNext** cuando el último registro está activo, la propiedad **EOF** es **True** y no habrá ningún registro activo. Si vuelve a utilizar **MoveNext** se producirá un error y **EOF** será **True**.

Si *recordset* hace referencia a un objeto **Recordset** de tipo Table (sólo espacios de trabajo Microsoft Jet), el movimiento seguirá el índice activo. Puede establecer el índice activo utilizando la propiedad **Index**. Si no establece el índice activo, el orden de los registros devueltos no estará definido.

Importante Puede utilizar el método **MoveLast** para llenar completamente un objeto **Recordset** de tipo Dynaset o Snapshot para obtener el número de registros activos en el **Recordset**. Sin embargo, si utiliza **MoveLast** puede hacer más lentas sus aplicaciones. Sólo debe utilizar **MoveLast** para obtener un total de registros si es absolutamente necesario obtener un total exacto en un **Recordset** abierto recientemente. Si utiliza la constante **dbRunAsync** con **MoveLast**, el método llamado es asíncrono.

No puede utilizar los métodos **MoveFirst**, **MoveLast** y **MovePrevious** en un **Recordset** de tipo Forward-only.

Para mover la posición del registro activo en un objeto **Recordset** un número específico de registros hacia adelante o hacia atrás, utilice el método **Move**.

NewPassword (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthNewPasswordC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthNewPasswordX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthNewPasswordA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthNewPasswordS"}

Cambia la contraseña de una cuenta de usuario existente o de la base de datos Microsoft Jet (sólo espacios de trabajo Microsoft Jet).

Sintaxis

objeto.**NewPassword** *antiguacontraseña*, *nuevacontraseña*

La sintaxis del método **NewPassword** utiliza los siguientes argumentos.

Argumento	Descripción
objeto	Una variable de objeto que representa el objeto **User** o el objeto **Database** de Microsoft Jet 3.*x* cuya propiedad **Password** quiere cambiar.
antiguacontraseña	Un **String** que es el valor actual de la propiedad **Password** del objeto **User** o **Database** de Microsoft Jet 3.*x*.
nuevacontraseña	Un **String** que es el nuevo valor de la propiedad **Password** del objeto **User** o **Database** de Microsoft Jet 3.*x*.

Comentarios

Las cadenas *antiguacontraseña* y *nuevacontraseña* pueden tener hasta 14 caracteres de longitud y pueden incluir cualquier carácter excepto el carácter ASCII 0 (nulo). Para eliminar la contraseña, utilice una cadena de longitud cero ("") para *nuevacontraseña*.

Las contraseñas distinguen entre mayúsculas y minúsculas.

Si *objeto* hace referencia a un objeto **User** que no es un elemento de una colección **Users**, se producirá un error. Para establecer una nueva contraseña, se debe haber registrado como el usuario cuya cuenta de usuario va a cambiar o debe ser miembro del grupo Administradores. La propiedad **Password** de un objeto **User** es sólo de escritura, los usuarios no pueden leer el valor activo.

Si *objeto* hace referencia al objeto **Database** Microsoft Jet de versión 3.0 o posterior, este método ofrece alguna seguridad mediante la protección de contraseña. Cuando crea o abre un archivo .mdb Microsoft Jet 3.*x* , una parte de cadena de conexión del método **Connect** describe la contraseña.

Si una base de datos no tiene contraseña, Microsoft Jet creará una automáticamente poniendo una cadena de longitud cero ("") en la antigua contraseña.

Precaución Si pierde su contraseña, no podrá volver a abrir su base de datos.

OpenDatabase (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthOpenDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthOpenDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthOpenDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthOpenDatabaseS"}

Abre una base de datos especificada en un objeto **Workspace** y devuelve una referencia al objeto **Database** que la representa.

Sintaxis

Set *basededatos* = *espaciodetrabajo*.**OpenDatabase (***nombrebasededatos*, *opciones*, *sólolectura*, *conexión*)

La sintaxis del método **OpenDatabase** consta de las siguientes partes.

Argumento	Descripción
basededatos	Una variable de objeto que representa el objeto **Database** que va a abrir.
espaciodetrabajo	Opcional. Una variable de objeto que representa el objeto **Workspace** existente que contendrá la base de datos. Si no incluye un valor para *espaciodetrabajo*, **OpenDatabase** utiliza el espacio de trabajo predeterminado.
nombrebasededatos	Un tipo de datos **String** que es el nombre de un archivo de base de datos Microsoft Jet existente o el nombre del origen de datos (DSN) de un origen de datos ODBC existente. Consulte la propiedad **Name** para obtener más información acerca de este valor.
opciones	Opcional. Un tipo de datos **Variant** que establece varias opciones para la base de datos, como se especifica en Valores.
sólolectura	Opcional. Un valor de tipo de datos **Variant** (subtipo **Boolean**) que es **True** si desea abrir la base de datos con acceso de sólo lectura o **False** (predeterminado) si desea abrir la base de datos con acceso de lectura/escritura.
conexión	Opcional. Un tipo de datos **Variant** (subtipo **String**) que especifica información variada sobre la conexión, incluyendo las contraseñas.

Valores

Para los espacios de trabajo Microsoft Jet, puede utilizar los siguientes valores para el argumento *opciones*:

Valor	Descripción
True	Abre la base de datos en modo exclusivo.
False	(Predeterminado) Abre la base de datos en modo compartido.

Para los espacios de trabajo ODBCDirect, el argumento *opciones* determina si y cuando informar al usuario acerca de establecer la conexión. Puede utilizar una de las siguientes constantes:

Constante	Descripción
dbDriverNoPrompt	El Administrador del controlador de

	ODBC utiliza la cadena de conexión suministrada en *nombrebasededatos* y *conexión*. Si no proporciona información suficiente, se produce un error en tiempo de ejecución.
DbDriverPrompt	El Administrador del controlador de ODBC muestra el cuadro de diálogo **ODBC Data Sources**, que muestra cualquier información interesante suministrada en *nombrebasededatos* o *conexión*. La cadena de conexión se realiza con el DSN que el usuario selecciona en los cuadros de diálogo o, si el usuario no especifica un DSN, se utiliza el valor predeterminado de DSN.
DbDriverComplete	(Predeterminado) Si los argumentos *conexión* y *nombrebasededatos* incluyen toda la información necesaria para completar la conexión, el Administrador del controlador de ODBC utiliza la cadena en *conexión*. De otro modo funciona como cuando especifica **dbDriverPrompt**.
DbDriverCompleteRequired	Esta opción funciona como **dbDriverComplete** excepto que el controlador ODBC desactiva el envío de información que no necesita para completar la conexión.

Comentarios

Cuando abre una base de datos, automáticamente se agrega a la colección **Databases**. Además, en un espacio de trabajo ODBCDirect, el objeto **Connection** correspondiente al objeto **Database** nuevo también se crea y anexa a la colección **Connections** del mismo objeto **Workspace**.

Estas son algunas consideraciones que debe aplicar cuando utilice *nombrebasededatos*:

- Si hace referencia a una base de datos que ya está abierta para acceso en modo exclusivo por otro usuario, se produce un error.
- Si no hace referencia a una base de datos existente o a un nombre de un origen de datos ODBC válido, se produce un error.
- Si es una cadena de longitud cero ("") y *conexión* es `"ODBC;"`, se muestra un cuadro de diálogo que enumera todos los nombres de orígenes de datos ODBC registrados para que el usuario pueda seleccionar una base de datos.
- Si está abriendo una base de datos mediante un espacio de trabajo ODBCDirect y proporciona el DSN en *conexión*, puede establecer *nombrebasededatos* a una cadena de su elección que puede utilizar para hacer referencia a esta base de datos en el código más adelante.

El argumento *conexión* se expresa en dos partes: el tipo de base de datos, seguido por punto y coma (;) y los argumentos opcionales. Primero debe proporcionar el tipo de base de datos, como `"ODBC;"` o `"FoxPro 2.5;"`. A continuación, os argumentos opcionales sin un orden concreto, separados por punto y coma. Uno de los parámetros puede ser la contraseña (si hay alguna asignada). Por ejemplo:

```
"FoxPro 2.5; pwd=micontraseña"
```

Utilizar el método **NewPassword** en un objeto **Database** distinto de una base de datos de ODBCDirect cambia el parámetro de contraseña que aparece en la parte `";pwd=..."` de este

argumento. Debe aplicar los argumentos *opciones* y *sólolectura* para proporcionar una cadena de origen. Consulte la propiedad **Connect** para su sintaxis.

Para cerrar una base de datos y, de este modo, quitar el objeto **Database** de la colección **Databases**, utilice el método **Close** en el objeto .

Nota Cuando tiene acceso a un origen de datos ODBC conectado a Microsoft Jet, puede mejorar el rendimiento de sus aplicaciones abriendo un objeto **Database** conectado al origen de datos ODBC, en vez de vincular objetos **TableDef** individuales a tablas específicas en el origen de datos ODBC.

OpenRecordset (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthOpenRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthOpenRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthOpenRecordsetA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthOpenRecordsetS"}

Crea un nuevo objeto **Recordset** y lo añade a la colección **Recordsets**.

Sintaxis

Para los objetos **Connection** y **Database**:

Set *variable* = *objeto***.OpenRecordset (***origen***,** *tipo***,** *opciones***,** *bloquearmodificaciones***)**

Para los objetos **QueryDef**, **Recordset** y **TableDef**:

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

La sintaxis del método **OpenRecordset** consta de las siguientes partes.

Argumento	Descripción
variable	Una variable de objeto que representa el objeto **Recordset** que desea abrir.
objeto	Una variable de objeto que representa un objeto existente desde el que desea crear el objeto **Recordset** nuevo.
origen	Un tipo de datos **String** que especifica el origen de los registros para el nuevo **Recordset**. El origen puede ser un nombre de tabla, un nombre de consulta o una instrucción SQL que devuelve registros. Para objetos **Recordset** de tipo bases de datos Microsoft Jet, el origen sólo puede ser un nombre de tabla.
tipo	Opcional. Una constante que indica el tipo de objeto **Recordset** a abrir, como se especifica en Valores.
opciones	Opcional. Una combinación de constantes que especifican las características del objeto **Recordset** nuevo, como se especifica en Valores.
bloquearmodificaciones	Opcional. Una constante que determina el bloqueo para el objeto **Recordset**, como se especifica en Valores.

Valores

Puede utilizar una de las siguientes constantes para el argumento *tipo*.

Constante	Descripción
dbOpenTable	Abre un objeto **Recordset** de tipo Table (sólo espacios de trabajo Microsoft Jet).
DbOpenDynamic	Abre un objeto **Recordset** de tipo Dynamic, que es parecido a un cursor dinámico ODBC (sólo espacios de trabajo ODBCDirect)
dbOpenDynaset	Abre un objeto **Recordset** de tipo Dynaset, que es parecido a un cursor de

	conjunto de claves ODBC.
dbOpenSnapshot	Abre un objeto **Recordset** de tipo Snapshot, que es parecido a un cursor estático ODBC.
dbOpenForwardOnly	Abre un objeto **Recordset** de tipo Forward-only.

Nota Si abre un objeto **Recordset** en un espacio de trabajo Microsoft Jet y no especifica un tipo, el método **OpenRecordset** crea una objeto **Recordset** de tipo Table, si es posible. Si especifica una tabla vinculada o una consulta, el método **OpenRecordset** crea un objeto **Recordset**.de tipo Dynaset. En un espacio de trabajo ODBCDirect, el valor predeterminado es **dbOpenForwardOnly**.

Puede utilizar una combinación de las siguientes constantes para el argumento *opciones*:

Constante	Descripción
dbAppendOnly	Permite al usuario anexar registros nuevos al objeto **Recordset**, pero impide la creación o eliminación de registros existentes (sólo objetos **Recordset** de tipo Dynaset de Microsoft Jet).
DbSQLPassThrough	Transfiere una instrucción SQL a un origen de datos ODBC conectado a Microsoft Jet para procesar (sólo un objeto **Recordset** de tipo Snapshot de Microsoft Jet).
DbSeeChanges	Genera un error en tiempo de ejecución si otro usuario está cambiando los datos que usted está modificando.(Sólo en objetos **Recordset** de tipo Snapshot de Microsoft Jet). Esto es útil en aplicaciones donde varios usuarios tiene acceso de lectura/escritura simultáneo a los mismos datos.
dbDenyWrite	Previene que otros usuarios puedan modificar o agregar registros (sólo objetos **Recordset** de Microsoft Jet).
dbDenyRead	Previene que otros usuarios puedan leer datos de una tabla (sólo objetos **Recordset** de tipo Table de Microsoft Jet).
dbForwardOnly	Crea un objeto **Recordset** de tipo Forward-only (sólo objetos **Recordset** de tipo Snapshot de Microsoft Jet). Se proporciona sólo para compatibilidad con versiones anteriores y debe utilizar la constante **dbOpenForwardOnly** en el argumento *tipo* en vez de utilizar esta opción.
dbReadOnly	Previene que otros usuarios puedan hacer cambios el objeto **Recordset** (sólo Microsoft Jet). La constante **dbReadOnly** en el argumento *bloquearmodificaciones* reemplaza esta opción, la cual se proporciona para compatibilidad con versiones anteriores.

DbRunAsync	Ejecuta una consulta asíncrona (sólo espacios de trabajo ODBCDirect).
dbExecDirect	Ejecuta una consulta saltando el método **SQLPrepare** y llamando directamente al método **SQLExecDirect** (sólo espacios de trabajo ODBCDirect). Utilice esta opción sólo cuando no se abra un objeto **Recordset** basándose en una consulta de parámetros. Para obtener más información, consulte la "Referencia del programador de Microsoft ODBC 3.0."
dbInconsistent	Permite actualizaciones inconsistentes (sólo objetos **Recordset** de tipo Dynaset de Microsoft Jet).
dbConsistent	Permite sólo actualizaciones consistentes (sólo objetos **Recordset** de tipo Dynaset de Microsoft Jet).

Nota Las constantes **dbConsistent** y **dbInconsistent** se excluyen mutuamente y el uso de ambos produce un error. Proporcionar un argumento *bloquearmodificaciones* cuando el argumento *opciones* utiliza la constante **dbReadOnly** también produce un error.

Puede utilizar las siguientes constantes para el argumento *bloquearmodificaciones*:.

Constante	Descripción
dbReadOnly	Previene que los usuarios hagan cambios al **Recordset** (predeterminado en espacios de trabajo ODBCDirect). Puede utilizar **dbReadOnly** en el argumento *opciones* o en el argumento *bloquearmodificaciones*, pero nunca en ambos. Si lo utiliza en ambos argumentos, se produce un error en tiempo de ejecución.
dbPessimistic	Utiliza el bloqueo pesimista para determinar cómo se pueden hacer cambios al objeto **Recordset** en un entorno multiusuario. La página que contiene el registro que está modificando está bloqueada mientras utiliza el método **Edit** (predeterminado en espacios de trabajo Microsoft Jet).
dbOptimistic	Utiliza el bloqueo optimista para determinar cómo se pueden hacer cambios al objeto **Recordset** en un entorno multiusuario. La página que contiene el registro que está modificando está bloqueada mientras se ejecuta el método **Update**.
dbOptimisticValue	Utiliza la concurrencia optimista basándose en valores de fila (sólo espacios de trabajo ODBCDirect).
dbOptimisticBatch	Activa la actualización optimista por lotes (sólo espacios de trabajo ODBCDirect).

Comentarios

En un espacio de trabajo Microsoft Jet, si *objeto* hace referencia a un objeto **QueryDef** o **Recordset** de tipo Dynaset o Snapshot o si *origen* hace referencia a una instrucción SQL o un **TableDef** que representa una tabla adjunta, no podrá utilizar **dbOpenTable** para el argumento *tipo* y si lo hace, se producirá un error interceptable. Si desea utilizar una consulta de paso a través de SQL en una tabla adjunta origen de datos ODBC conectado a Microsoft Jet, debe establecer primero la propiedad **Connect** de la base de datos de la tabla adjunta a una cadena de conexión ODBC válida. Si sólo necesita hacer una única pasada a un **Recordset** abierto desde un origen de datos ODBC conectado a Microsoft Jet, puede mejorar el rendimiento utilizando **dbOpenForwardOnly** para el argumento *tipo*.

Si *objeto* hace referencia a un **Recordset** de tipo Dynaset o Snapshot, el **Recordset** nuevo es del mismo tipo *objeto*. Si *objeto* hace referencia a un objeto **Recordset** de tipo Table, el tipo del objeto nuevo es un objeto **Recordset** de tipo Dynaset. No puede abrir objetos **Recordset** nuevos desde objetos **Recordset** de tipo Forward-only u ODBCDirect.

En un espacio de trabajo ODBCDirect, puede abrir un objeto **Recordset** que contiene más de una consulta de selección en el argumento *origen*, como

```
"SELECT Apellido, Nombre FROM Autores
WHERE Apellidos = 'García';
SELECT Título, Código FROM Títulos
WHERE Código Like '1-55615-*'"
```

El objeto **Recordset** devuelto se abrirá con el resultado de la primera consulta. Para obtener el conjunto de registros de resultado de consultas subsiguientes, utilice el método **NextRecordset**.

Nota Puede enviar consultas DAO a una gran variedad de servidores de bases de datos con ODBCDirect, diferentes servidores reorganizarán ligeramente los diferentes dialectos de SQL. Por esto, la Ayuda sensible al contexto de SQL de Microsoft Jet no es muy extensa, aunque la Ayuda en pantalla de SQL de Microsoft Jet esté todavía incluida en el menú Ayuda. Compruebe que la documentación de referencia para el dialecto SQL de su base de datos del servidor es la apropiada cuando utilice cualquier conexión ODBCDirect, o consultas de paso a través, en aplicaciones cliente/servidor conectadas a Microsoft Jet.

Utilice la constante **dbSeeChanges** en espacio de trabajo Microsoft Jet si desea captar los cambios realizados mientras dos o más usuarios están modificando o eliminando el mismo registro. Por ejemplo, si dos usuarios empiezan a modificar el mismo registro, el primer usuario que ejecute el método **Update** consigue realizar la operación. Cuando el segundo usuario ejecute el método **Update** ocurre un error de tiempo de ejecución. Del mismo modo, si el segundo usuario intenta utilizar el método **Delete** para eliminar un registro y el primer usuario ha cambiado ya el mismo, se produce un error de tiempo de ejecución.

En general, si al usuario se le presenta este error mientras está actualizando, su código debe actualizar el contenido de los campos y leer los valores recientemente modificados. Si se produce el error durante el proceso de eliminación, el código puede mostrar al usuario los nuevos datos del registro junto con un mensaje que indica que se han modificado recientemente los datos. En este momento, el código puede solicitar una confirmación de que el usuario desea aún eliminar el registro.

También podría utilizar la constante **dbSeeChanges** si abre un objeto **Recordset** en un espacio de trabajo ODBC conectado a Microsoft Jet contra una tabla de Microsoft SQL Server 6.0 (o posterior) que tiene una columna IDENTITY, en caso contrario se puede producir un error.

En un espacio de trabajo ODBCDirect, puede ejecutar consultas asíncronas estableciendo la constante **dbRunAsync** en el argumento *opciones*. Esto permite a su aplicación seguir procesando otras instrucciones mientras se ejecuta la consulta en segundo plano. Pero, no puede tener acceso a los datos del objeto **Recordset** hasta que se haya completado la consulta. Para determinar si la consulta a terminado, compruebe la propiedad **StillExecuting** del objeto **Recordset** nuevo. Si la consulta tarda más tiempo en acabar del esperado, puede terminar la ejecución de la consulta con el método **Cancel**.

Abrir más de un objeto **Recordset** en un origen de datos ODBC puede fallar porque las conexiones están ocupadas con una llamada al método **OpenRecordset**. prioritaria. Una alternativa a esto es utilizar un cursor del lado del servidor y ODBCDirect, si el servidor lo admite. Otra solución es llenar completamente objeto **Recordset** utilizando el método **MoveLast** en cuanto se abre el **Recordset**.

Si abre un objeto **Connection** con **DefaultCursorDriver** establecido a **dbUseClientBatchCursor**, puede abrir un objeto **Recordset** para hacer cambios a los datos en la memoria caché (conocido como actualización por lotes) en un espacio de trabajo ODBCDirect. Incluya **dbOptimisticBatch** en el argumento *bloquearmodificaciones* para activar la actualización en la memoria caché. Consulte el tema del método **Update** para obtener más detalles acerca de cómo grabar los cambios en el disco inmediatamente o hacer cambios en la memoria caché y grabarlos en disco como un proceso por lotes.

Al cerrar un **Recordset** utilizando el método **Close**, se eliminará automáticamente de la colección **Recordsets**.

Nota Si *origen* hace referencia a una instrucción SQL compuesta por una cadena concatenada con valores no enteros y los parámetros del sistema especifican un signo decimal no de EE.UU. como una coma (por ejemplo, `strSQL = "PRECIO > " & lngPrecio` y `lngPrecio = 125,50`), se produce un error cuando intenta abrir el objeto **Recordset**. Esto es debido a que durante la concatenación, el número se convertirá en una cadena utilizando su signo decimal predeterminado del sistema y SQL sólo acepta signos decimales de EE.UU.

Refresh (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthRefreshC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthRefreshX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthRefreshA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthRefreshS"}

Actualiza los objetos de una colección para reflejar el esquema de la base de datos actual.

Sintaxis

colección.**Refresh**

El marcador de posición *colección* es una variable de objeto que representa una colección persistente.

Comentarios

El método **Refresh** no se puede utilizar con colecciones que no sean persistentes, tales como **Connections**, **Databases**, **Recordsets**, **Workspaces** o la colección **QueryDefs** del objeto **Connection**.

Para determinar la posición que utiliza el motor de base de datos Microsoft Jet para los objetos **Field** en la colección **Fields** de un objeto **QueryDef**, **Recordset** o **TableDef**, utilice la propiedad **OrdinalPosition** de cada objeto **Field**. Cambiar la propiedad **OrdinalPosition** de un objeto **Field** puede no cambiar el orden de los objetos **Field** en la colección hasta que utilice el método **Refresh**.

Utilice el método **Refresh** en entornos multiusuario en los que otros usuarios puedan hacer cambios en la base de datos. Puede que también necesite utilizarlo en las colecciones que resulten afectadas indirectamente por cambios en la base de datos. Por ejemplo, si cambia una colección **Users**, es posible que necesite actualizar una colección **Groups** antes de utilizar la colección **Groups**.

Una colección se rellena con objetos la primera vez que se hace referencia a ella. La colección no reflejará los cambios subsiguientes realizados en ella por otros usuarios. Si es probable que una colección haya sufrido cambios por parte de otro usuario, utilice el método **Refresh** en la colección inmediatamente antes de llevar a cabo cualquier tarea de su aplicación que asuma la presencia o ausencia de un objeto particular en la colección. Esto asegurará que la colección esté tan actualizada como sea posible. De otra forma, utilizando el método **Refresh** puede ralentizar innecesariamente el rendimiento.

RefreshLink (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthRefreshLinkC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthRefreshLinkX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthRefreshLinkA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthRefreshLinkS"}

Actualiza la información de conexión para una tabla vincualda (sólo espacios de trabajo Microsoft Jet).

Sintaxis

deftabla.**RefreshLink**

El argumento *deftabla* especifica el objeto **TableDef** que representa la tabla adjunta cuya información de conexión desea actualizar.

Comentarios

Para cambiar la información de conexión de una tabla adjunta, restablezca la propiedad **Connect** del objeto **TableDef** correspondiente y utilice a continuación el método **RefreshLink** para actualizar la información. Cuando utilice el método **RefreshLink** no cambiarán las propiedades y los objetos **Relation** de la tabla adjunta.

Para lograr que esta información de conexión exista en todas las colecciones asociadas al objeto **TableDef** que representa la tabla adjunta, deberá utilizar también el método **Refresh** en cada colección.

RegisterDatabase (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthRegisterDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthRegisterDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthRegisterDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthRegisterDatabaseS"}

Introduce información de conexión para un origen de datos ODBC en el Registro de Windows. El controlador ODBC necesita información de conexión cuando el ODBC abre el origen de datos durante una sesión.

Sintaxis

DBEngine.RegisterDatabase *nombrebasededatos*, *controlador*, *silencio*, *atributos*

La sintaxis del método **RegisterDatabase** utiliza estas partes.

Argumento	Descripción
nombrebasededatos	Un tipo de datos **String** que es el nombre utilizado en el método **OpenDatabase** que hace referencia a un bloque de información descriptiva acerca del origen de datos. Por ejemplo, si el origen de los datos es una base de datos de ODBC remota, será el nombre del servidor.
controlador	Un tipo de datos **String** que es el nombre del controlador ODBC. Éste no es el nombre del archivo de DLL del controlador ODBC. Por ejemplo, SQL Server es el nombre de un controlador, pero SQLSRVR.dll es el nombre de un archivo .dll. Debe tener ODBC y el controlador apropiado ya instalados.
silencio	Un tipo de datos **Boolean** que es **True** si no desea mostrar los cuadros de diálogo del controlador ODBC que solicitan información específica para el controlador o **False** si desea mostrar los cuadros de diálogo del controlador ODBC. Si *silencio* es **True**, *atributos* deberá contener toda la información necesaria específica del controlador o de lo contrario, aparecerán los cuadros de diálogo.
atributos	Un tipo de datos **String** que es una lista de palabras clave que se agregan al Registro de Windows. Las palabras clave tienen el formato de cadenas delimitadas por retornos de carro.

Comentarios

Si la base de datos ya se ha registrado (la información de conexión ya se ha introducido) en Registro de Windows cuando utilice el método **RegisterDatabase** la información de conexión se actualizará.

Si el método **RegisterDatabase** falla por cualquier razón, no se hará ningún cambio en Registro de Windows y se producirá un error.

Para obtener más información sobre los controladores ODBC tales como SQL Server, consulte el archivo de Ayuda suministrado con el controlador.

Podría utilizar el cuadro de diálogo **Orígenes de datos ODBC** en el Panel de control para agregar de datos nuevo o para hacer cambios a entradas existentes. Sin embargo, si opta por usar el método **RegisterDatabase**, es aconsejable establecer la opción *silencio* en **True**.

RepairDatabase (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthRepairDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthRepairDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthRepairDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthRepairDatabaseS"}

Intenta reparar una base de datos dañada que accede a base de datos Microsoft Jet (sólo bases de datos Microsoft Jet).

Sintaxis

DBEngine.RepairDatabase *nombrebasededatos*

El argumento *nombrebasededatos* representa un tipo de datos **String** que es la ruta de acceso y el nombre de un archivo de base de datos del motor Microsoft Jet existente. Si se omite la ruta, sólo se buscará en el directorio activo. Si el sistema admite la convención uniforme para nombres (UNC), también puede especificar una ruta de acceso de red, como por ejemplo `"\\server1\share1\dir1\db1.mdb"`.

Comentarios

Debe cerrar la base de datos especificada por *nombrebasededatos* antes de repararla. En un entorno multiusuario, los demás usuarios no podrán tener abierto *nombrebasededatos* mientras usted la repara. Si no está cerrado *nombrebasededatos* o no está disponible para uso exclusivo, se producirá un error.

Este método intenta reparar una base de datos marcada como posiblemente dañada por una operación de escritura incompleta. Esto puede ocurrir si una aplicación que utiliza el motor de base de datos Microsoft Jet termina inesperadamente debido a un corte en el suministro eléctrico o un problema de hardware. La base de datos no se marcará como posiblemente dañada si utiliza el método **Close** o si sale de la aplicación de una manera normal.

El método **RepairDatabase** también intenta validar todas las tablas del sistema y todos los índices. Se descartan los datos que no se puedan reparar. Si no se puede reparar la base de datos, se produce un error interceptable.

Cuando intente abrir o compactar una base de datos dañada, normalmente se producirá un error interceptable. En algunas situaciones, sin embargo, puede que no se detecte una base de datos dañada y no se produzca ningún error. Es conveniente ofrecer a los usuarios un método de ejecutar el método **RepairDatabase** en su aplicación, si la base de datos se comporta de manera impredecible.

Algunos tipos de bases de datos se pueden dañar si un usuario termina una aplicación sin cerrar los objetos **Database** o **Recordset** y el motor de base de datos Microsoft Jet, Microsoft Windows no tienen la oportunidad de vaciar las memorias de caché de datos. Para evitar que se dañen las bases de datos, establezca procedimientos para cerrar las aplicaciones y apagar los sistemas que aseguren que todas las páginas de la memoria caché están guardadas en la base de datos. En algunos casos, puede que sean necesarias fuentes de alimentación ininterrumpida para evitar pérdidas de datos por las fluctuaciones del suministro eléctrico.

Nota Después de reparar una base de datos, también es conveniente compactar la misma utilizando el método **CompactDatabase** para defragmentar el archivo y recuperar espacio en disco.

Requery (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthRequeryC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthRequeryX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthRequeryA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthRequeryS"}

Actualiza los datos de un objeto **Recordset** volviendo a ejecutar la consulta en la que se basa el objeto.

Sintaxis

recordset.**Requery** *nuevadefconsulta*

La sintaxis del método **Requery** tiene las siguientes partes.

Argumento	Descripción
recordset	Una variable de objeto que representa un objeto **Recordset** de tipo Dynaset, Snapshot o Forward-only de Microsoft Jet o un objeto **Recordset** de ODBCDirect.
nuevadefconsulta	Opcional. Un tipo de datos **Variant** que representa el valor de la propiedad **Name** de un objeto **QueryDef** (sólo espacios de trabajo Microsoft Jet).

Comentarios

Utilice este método para comprobar que un **Recordset** contiene los datos más recientes. Este método vuelve a llenar el objeto **Recordset** utilizando los parámetros de la consulta actual o (en un espacio de trabajo Microsoft Jet) las nuevas proporcionadas por el argumento *nuevadefconsulta*.

En un espacio de trabajo ODBCDirect, si la consulta original era asíncrona el método **Requery** también ejecutará una consulta asíncrona.

Si no especifica un argumento *nuevadefconsulta*, el objeto **Recordset** se rellena basándose en la misma definición y los mismos parámetros de consulta utilizados para llenar el objeto **Recordset** originalmente. Cualquier cambio a los datos base se reflejarán mientras se vuelve a llenar. Si no utilizó un método **QueryDef** para crear el objeto **Recordset**, el objeto **Recordset** se vuelve a crear sin datos.

Si especifica el método **QueryDef** original en el argumento *nuevadefconsulta* el objeto **Recordset** se necesita al utilizar los parámetros especificados por el **QueryDef**. Cualquier cambio a los datos base se reflejarán mientras se vuelve a llenar. Para reflejar cualquier cambio a los valores de los parámetros de la consulta en el objeto **Recordset**, debe aplicar el argumento *nuevadefconsulta* proporcionado.

Si especifica un objeto **QueryDef** diferente del que utilizó originalmente para crear el objeto **Recordset**, el **Recordset** se vuelve a crear sin datos.

Cuando utilice **Requery**, el primer registro del **Recordset** se convertirá en el registro activo.

No puede utilizar el método **Requery** en los objetos **Recordset** de tipo Dynaset o Snapshot cuya propiedad **Restartable** sea **False**. Sin embargo, si indica el argumento *nuevadefconsulta* opcional, se ignora la propiedad **Restartable**.

Si los valores de propiedad **BOF** y **EOF** del objeto **Recordset** son **True** después de utilizar el método **Requery**, la consulta no devolvió ningún registro y el **Recordset** no contiene datos.

Seek (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthSeekC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthSeekX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthSeekA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthSeekS"}

Busca el registro de un objeto **Recordset** indexado de tipo Table que cumple el criterio especificado para el índice activo y lo convierte en el registro activo (sólo espacios de trabajo Microsoft Jet).

Sintaxis

recordset.**Seek** *comparación*, *clave1*, *clave2...clave13*

La sintaxis del método **Seek** tiene las siguientes partes.

Argumento	Descripción
recordset	Una variable de objeto que representa un objeto **Recordset** de tipo Table que tiene definido un índice como se especifica en la propiedad **Index** del objeto **Recordset**.
comparación	Una de esta expresiones de cadena: <, <=, =, >=, or >.
clave1, clave2...clave13	Uno o más valores que corresponden a los campos en el índice activo del objeto **Recordset**, como se especifica en el valor de la propiedad **Index**. Puede utilizar un argumento de hasta 13 *claves*.

Comentarios

Debe establecer el índice activo con la propiedad **Index** antes de usar **Seek**. Si el índice identifica un campo clave no único, **Seek** busca el primer registro que cumpla los criterios.

El método **Seek** busca en los campos clave especificados y localiza el primer registro que cumpla el criterio especificado por *comparación* y *clave1*. Cuando lo encuentra, convierte ese registro en activo y la propiedad **NoMatch** se establece en **False**. Si el método **Seek** no consigue localizar ninguna coincidencia, la propiedad **NoMatch** se establece en **True** y el registro activo es indefinido.

Si *comparación* es igual (=),mayor o igual (>=) o mayor que (>), **Seek** empezará al principio del índice y buscará hacia adelante.

Si *comparación* es menor que (<) o mayor o igual que (<=), **Seek** empezará al final del índice y buscará hacia atrás, a menos que haya entradas de índice duplicadas al final. En tal caso, **Seek** empezará en una entrada cualquiera entre las entradas duplicadas existentes al final del índice.

Debe especificar valores para todos los campos definidos en el índice. Si utiliza **Seek** con un índice de múltiples columnas y no especifica un valor de comparación para cada campo del índice, no podrá usar el operador de igual (=) en la comparación. Esto se debe a que algunos de los campos de criterio(*clave2*, *clave3*, etc) estarán predeterminados en **Null**, lo que posiblemente no concordará. Por tanto, el operador de igual sólo funcionará correctamente si tiene un registro que sea **Null** n su totalidad, excepto la clave que está buscando. Es aconsejable usar el operador mayor que o igual en su lugar.

El argumento *clave1* debe tener el mismo tipo de dato de campo que el campo correspondiente del índice activo. Por ejemplo, si el índice activo hace referencia a un campo clave numérico (como IdEmpleado), *clave1* debe ser numérica. Del mismo modo, si el índice activo hace referencia a un campo tipo Texto (como Apellidos), *clave1* debe ser una cadena.

No es necesario que haya un registro activo cuando utilice **Seek**.

Puede utilizar la colección **Indexes** para enumerar los índices existentes.

Para localizar un registro en un **Recordset** de tipo Dynaset o Snapshot que satisfaga una condición

específica, utilice los métodos **Find**. Para incluir todos los registros, no simplemente los que cumplan una condición específica, utilice los métodos **Move** para desplazarse entre los registros.

No puede utilizar el método **Seek** en una tabla adjunta de ningún tipo, porque las tablas adjuntas deben estar abiertas como objetos **Recordset** de tipo Dynaset o Snapshot. Sin embargo, si utiliza el método **OpenDatabase** para abrir directamente una base de datos de ISAM instalable (no de ODBC), podrá utilizar **Seek** en las tablas de esa base de datos.

En un espacio de trabajo ODBCDirect, los métodos **Find** y **Seek** no están disponibles para ningún tipo de objeto **Recordset**, porque ejecutar un **Find** o **Seek** mediante una conexión ODBC no es muy eficaz en una red. En vez de ello, podría diseñar la consulta (utilizando el argumento *origen* del método **OpenRecordset**) con una cláusula WHERE apropiada que limita los registros devueltos a sólo aquellos que encuentran el criterio que quería utilizar en un **Find** o **Seek**.

Synchronize (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthSynchronizeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthSynchronizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthSynchronizeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthSynchronizeS"}

Sincroniza dos réplicas. (sólo bases de datos Microsoft Jet).

Sintaxis

*basededatos.***Synchronize** *rutaacceso***,** *intercambio*

La sintaxis del método **Synchronize** tiene las siguientes partes.

Parte	Descripción
basededatos	Una variable de objeto que representa un objeto **Database** que es una réplica.
rutaacceso	Un tipo de datos **String** que contiene la ruta de acceso a la réplica de destino con la que *basededatos* se sincronizará. La extensión del nombre del archivo.mdb es opcional.
intercambio	Opcional. Una constante que indica la dirección en la que se van a realizar los cambios entre las dos bases de datos, como se especifica en Valores.

Valores

Puede utilizar las siguientes constantes en el argumento *intercambio*. Puede utilizar una de las primeras tres constantes con o sin la cuarta:

Constante	Descripción
dbRepExportChanges	Envía los cambios desde *basededatos* a *rutaacceso.*
dbRepImportChanges	Recibe los cambios desde *rutaacceso* a *basededatos.*
dbRepImpExpChanges	(Predeterminado) Envía los cambios desde *basededatos* a *rutaacceso* y viceversa, también se conoce como intercambio bidireccional.
dbRepSyncInternet	Intercambia los datos entre los archivos conectados a través de Internet.

Comentarios

Utilice **Synchronize** para realizar un intercambio de datos y de diseño entre las dos bases de datos. Los cambios de diseño se realizan siempre primero. Ambas bases de datos deben estar en el mismo nivel de diseño antes de que se puedan intercambiar los datos. Por ejemplo, un intercambio del tipo **dbRepExportChanges** puede hacer que los cambios de diseño se realicen en una réplica, aunque los cambios de datos fluyan sólo desde *basededatos* a *rutaacceso*.

La réplica identificada en *rutaacceso* debe formar parte del mismo conjunto de réplicas. Si ambas réplicas tienen el mismo **ReplicaID** o son Diseños maestros de dos conjuntos de réplica diferentes, fallará la sincronización.

Cuando sincroniza dos réplicas en Internet, debe utilizar la constante **dbRepSyncInternet**. En este caso, especifique una dirección URL para el argumento *rutaacceso* en vez de especificar una ruta de acceso de red.

Nota No puede sincronizar réplicas parciales con otras réplicas parciales. Consulte el método **PopulatePartial** para obtener más información.

La sincronización en Internet necesita el Administrador de réplica, que sólo está disponible en Microsoft Office 97, Edición de desarrollo.

Update (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthUpdateC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthUpdateX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthUpdateA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthUpdateS"}

Guarda el contenido del búfer de copia en un objeto **Recordset** de tipo Dynaset o Table especificado.

Sintaxis

recordset.**Update (***tipo*, *obligar* **)**

La sintaxis del método **Update** tiene las siguientes partes.

Parte	Descripción
recordset	Una variable de objeto que representa un objeto **Recordset** abierto que se puede actualizar.
tipo	Opcional. Una constante que indica el tipo de actualización, como se especifica en Valores (sólo espacios de trabajo ODBCDirect).
obligar	Opcional. Un valor de tipo **Boolean** que indica si se pueden o no obligar los cambios en la base de datos, sin tener en cuenta si los datos base se han cambiado por otro usuario desde la llamada al método **AddNew**, **Delete** o **Edit**. Si es **True**, los cambios se fuerzan y los cambios hechos por otros usuarios se sobrescriben. Si es **False** (predeterminado), los cambios hechos por otros usuarios mientras la actualización está pendiente provocarán que falle la actualización para aquellos cambios conflictivos. No se produce ningún error, pero las propiedades **BatchCollisionCount** y **BatchCollisions** indicarán el número de conflictos y las filas afectadas por los mismos, respectivamente (sólo espacios de trabajo ODBCDirect).

Valores

Puede utilizar los siguientes valores en el argumento *tipo*. Puede utilizar los valores no predeterminados sólo cuando está activada la actualización por lotes.

Constante	Descripción
dbUpdateRegular	Predeterminado. Los cambios pendientes no pasan a la memoria caché y se graban en el disco inmediatamente.
dbUpdateBatch	Todos se graban en disco los cambios pendientes en la memoria caché de actualización.
dbUpdateCurrentRecord	Sólo se graban en disco los cambios pendientes del registro activo.

Comentarios

Utilice **Update** para guardar el registro activo y los cambios que haya efectuado en él.

Precaución Los cambios realizados en el registro activo se perderán si:

- Utiliza el método **Edit** o **AddNew** y a continuación, pasa a otro registro sin actualizarlo previamente mediante **Update**.
- Utiliza **Edit** o **AddNew** y, a continuación, vuelve a usar **Edit** o **AddNew** sin utilizar previamente

Update.

- Establece la propiedad **Bookmark** para otro registro.
- Cierra el conjunto de registros a los que hace referencia *recordset* sin utilizar primero **Update**.
- Cancela la operación **Edit** utilizando el método **CancelUpdate**.

Para modificar un registro, utilice el método **Edit** para copiar el contenido del registro activo al búfer de copia. Si no utiliza **Edit** en primer lugar, se producirá un error cuando utilice **Update** o intente cambiar el valor de un campo.

En un espacio de trabajo ODBCDirect, puede realizar actualizaciones por lotes, proporcionadas por la biblioteca de cursores compatible con actualizaciones por lotes y si el objeto **Recordset** se abrió con la opción de bloqueo optimista.

En un espacio de trabajo Microsoft Jet, cuando el objeto **Recordset** de la propiedad **LockEdits** establecida como **True** (bloqueo pesimista) en un entorno multiusuario, el registro permanecerá bloqueado desde el momento en que se utiliza **Edit** hasta que se ejecuta el método **Update** o se cancele la edición. Si la configuración de la propiedad **LockEdits** es **False** (bloqueo optimista), el registro se bloquea y se compara con el registro previamente modificado justo antes de se actualizado en la base de datos. Si ha cambiado el registro desde que utilizó el método **Edit**, la operación **Update** falla. El bloqueo optimista se utiliza siempre en Microsoft Jet conectado a ODBC y ISAM instalable. Para que la operación **Update** continúe con los cambios, utilice de nuevo el método **Update**. Para volver al registro, tal como lo cambió el otro usuario, actualice el registro activo usando los métodos `Move 0`.

Nota Para agregar, modificar o eliminar un registro, debe haber un índice único en el registro del origen de datos base. Se obtiene no lo hay, se producirá un error "Permiso denegado" en la llamada al método **AddNew**, **Delete** o **Edit** en un espacio de trabajo Microsoft Jet, se producirá un error "Argumento no válido" en la llamada al método **Update** en un espacio de trabajo ODBCDirect.

Cancel (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthCancelC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthCancelX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthCancelA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthCancelS"}

Cancela la ejecución de un método asícrono de llamada (sólo espacios de trabajo ODBCDirect).

Sintaxis

objeto.**Cancel**

La sintaxis del método **Cancel** consta de las siguientes partes:

Parte	Descripción
objeto	Una expresión de cadena que evalúa uno de los objetos de la lista "Aplicable a"

Comentarios

Utilice el método **Cancel** para finalizar la ejecución de una llamada asíncrona de los métodos **Execute**, **OpenConnection** u **OpenRecordset** (es decir el método se invocó con la opción **dbRunAsync**). **Cancel** devolverá un error de tiempo de ejecución si en el método que está intentando finalizar no se utilizó **dbRunAsync**.

La siguiente tabla muestra qué tarea se finaliza cuando utiliza el método **Cancel** en un tipo de objeto en particular.

Si el objeto es un	Se finaliza este método
Connection	**Execute** u **OpenConnection**
QueryDef	**Execute**
Recordset	**OpenRecordset**

Se producirá un error si, a continuación de una llamada de método **Cancel**, intenta hacer referencia al objeto que se habría creado por una llamada asíncrona **OpenConnection** u **OpenRecordset** (es decir, el objeto **Connection** o **Recordset** desde el que llamó al método **Cancel**).

NextRecordset (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthNextRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthNextRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthNextRecordsetA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthNextRecordsetS"}

Obtiene el siguiente conjunto de registros, si hay alguno, devuelto por una consulta de selección de múltiples partes en una llamada **OpenRecordset** y devuelve un valor de tipo **Boolean** que indica si hay uno o más registros pendientes (sólo espacios de trabajo ODBCDirect).

Sintaxis

Set *boolean* = *recordset*.**NextRecordset**

La sintaxis del método **NextRecordset** consta de las siguientes partes:

Parte	Descripción
boolean	Una variable de tipo **Boolean**. **True** indica que el siguiente conjunto de registros está disponible en *recordset*; **False** indica que no hay más registros pendientes y que *recordset* está ahora vacío.
recordset	Una variable existente de un objeto **Recordset** a la que quiere devolver los registros pendientes.

Comentarios

En un espacio de trabajo ODBCDirect puede abrir un objeto **Recordset** , que contenga más de una consulta de selección en el argumento ***origen*** de **OpenRecordset** o la propiedad **SQL** de un objeto **QueryDef** de una consulta de selección, como en el siguiente ejemplo.

```
SELECT Apellidos, Nombre FROM Autores
WHERE Apellidos = 'Pérez';
SELECT Título, ISBN FROM Títulos
WHERE Id_Pub = 9999
```

El objeto **Recordset** devuelto se abrirá con los resultados de la primera consulta. Para obtener los conjuntos de resultados de registros de consultas subsecuentes, utilice el método **NextRecordset**.

Si hay más registros disponibles (es decir si había otra consulta de selección en la llamada del método **OpenRecordset** o en la propiedad **SQL**), los registros devueltos desde la siguiente consulta se cargarán en el objeto **Recordset** y el método **NextRecordset** devolverá **True** para indicar que los registros están disponibles. Cuando ya no hay más registros disponibles (es decir, se han cargado en el objeto **Recordset** los resultados del la última consulta de selección), el método **NextRecordset** devolverá **False** y el objeto **Recordset** estará vacío.

También puede utilizar el método **Cancel** para desechar los contenidos de un objeto **Recordset**. Sin embargo, **Cancel** también deshecha los registros adicionales que todavía no están cargados.

OpenConnection (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthOpenConectionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthOpenConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthOpenConectionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthOpenConectionS"}

Abre un objeto **Connection** en un origen de datos ODBC (sólo espacios de trabajo ODBCDirect).

Sintaxis

Set *conexión* = *espaciodetrabajo*.**OpenConnection (***nombre, opciones, sólolectura, conectar***)**

La sintaxis del método **OpenConnection** consta de las siguientes partes:

Parte	Descripción
conexión	Una variable de un objeto **Connection** a la que se asignará la nueva conexión.
espaciodetrabajo	Opcional. Una variable de un tipo de datos de un objeto **Workspace** que referencia al objeto **Workspace** existente que contendrá la nueva conexión.
nombre	Una expresión de cadena. Ver en comentarios.
opciones	Opcional. Un valor de tipo **Variant** que establece varias opciones para la conexión, tal y como se especifica en Valores que se pueden establecer. En base a este valor el administrador del controlador de ODBC solicita al usuario información sobre la conexión, como nombre del origen de datos (DSN), nombre del usuario y contraseña.
sólolectura	Opcional. Un valor de tipo **Boolean** que es **True** si la conexión se va a abrir con acceso de sólo lectura y **False** si la conexión se va a abrir para acceso de lectura/escritura (predeterminado)
conectar	Opcional. Una cadena de conexión de ODBC. Vea la propiedad **Connect** para los elementos específicos y la sintaxis de esta cadena. Se requiere un "ODBC". Si se omite *conectar*, el UID y/o el PWD se obtendrán de las propiedades **UserName** y **Password** del objeto **Workspace**.

Valores que se pueden establecer

El argumento *opciones* determina si se solicita al usuario que establezca la conexión y cuándo. Puede utilizar una de las siguientes constantes.

Constante	Descripción
dbDriverNoPrompt	El administrador del controlador de

	ODBC utiliza la cadena de conexión proporcionada en *nombrebasedatos* y *conectar.* Si no proporciona suficiente información, se producirá un error de tiempo de ejecución.
dbDriverPrompt	El administrador del controlador de ODBC muestra el cuadro de diálogo de **ODBC Data Sources** que muestra cualquier información relevante suministrada en *nombrebasedatos* o *conectar*. La cadena de conexión se hace con el DSN que selecciona el usuario por medio de los cuadros de diálogo, si no se especifica un DSN, se utilizará el predeterminado.
dbDriverComplete	Predeterminado. Si el argumento *conectar* incluye toda la información necesaria para completar una conexión, el administrador del controlador de ODBC utiliza la cadena en *conectar*. En caso contrario se comporta como cuando se especifica **dbDriverPrompt**.
dbDriverCompleteRequired	Esta opción se comporta como **dbDriverComplete** excepto porque el controlador de ODBC deshabilita las peticiones de información que no sea necesaria para completar la conexión.

Comentarios

Utilice el método **OpenConnection** para establecer una conexión con un origen de datos de un espacio de trabajo ODBCDirect. El método **OpenConnection** es similar, pero no equivalente a **OpenDatabase**. La principal diferencia es que el método **OpenConnection** sólo está disponible en un espacio de trabajo ODBCDirect.

Si especifica un nombre de origen de datos (DSN) de ODBC registrado en el argumento *conectar*, entonces el argumento *nombre* puede ser cualquier cadena válida y también proporcionará la propiedad **Name** para el objeto **Connection**. Si en el argumento *conectar* no se incluye un DSN válido, entonces *nombre* debe hacer referencia a un DSN de ODBC válido, que también será la propiedad **Name**. Si ni *nombre* ni *conectar* contienen un DSN válido, el administrador del controlador de ODBC se puede establecer (por el argumento *opciones*) para que solicite al usuario la información necesaria. El DSN suministrado por la petición proporcionará la propiedad **Name**.

El método **OpenConnection** devuelve un objeto **Connection** que contiene información sobre la conexión. El objeto **Connection** es similar a un objeto **Database**. La principal diferencia es que un objeto **Database** normalmente representa una base de datos, aunque se puede utilizar para representar una conexión con un origen de datos ODBC de un espacio de trabajo Microsoft Jet.

SetOption (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthSetOptionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthSetOptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthSetOptionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthSetOptionS"}

Establece valores nuevos para las teclas del motor de base de datos Microsoft Jet en el Registro de Windows (sólo espacios de trabajo Microsoft Jet).

Sintaxis

DBEngine.SetOption *parámetro, nuevovalor*

La sintaxis del método **SetOption** consta de las siguientes partes:.

Parte	Descripción
parámetro	Una constante de tipo **Long** tal y como se describe en Valores que se pueden establecer.
nuevovalor	Un valor de tipo **Variant** al que quiere establecer el *parámetro.*

Valores que se pueden establecer

Cada constante hace referencia a la clave de registro correspondiente en la ruta de acceso Jet\3.5\Engines\Jet 3.5\ (es decir, **dbSharedAsyncDelay** se corresponde con la clave Jet\3.5\Engines\Jet 3.5\SharedAsyncDelay, etc.).

Constante	Descripción
dbPageTimeout	La clave PageTimeout
dbSharedAsyncDelay	La clave SharedAsyncDelay
dbExclusiveAsyncDelay	La clave ExclusiveAsyncDelay
dbLockRetry	La clave LockRetry
dbUserCommitSync	La clave UserCommitSync
dbImplicitCommitSync	La clave ImplicitCommitSync
dbMaxBufferSize	La clave MaxBufferSize
dbMaxLocksPerFile	La clave MaxLocksPerFile
dbLockDelay	La clave LockDelay
dbRecycleLVs	La clave RecycleLVs
dbFlushTransactionTimeout	La clave FlushTransactionTimeout

Comentarios

Utilice el método **SetOption** para sustituir valores del Registro en tiempo de ejecución. Los nuevos valores establecidos con el método **SetOption** no tienen efecto hasta que cambie de nuevo a otra llamada **SetOption**, o hasta que cierre el objeto **DBEngine**.

Para más detalles sobre qué hacen las claves de registro y sus valores adecuados, vea Inicialización del motor de base de datos Microsoft Jet 3.5.

PopulatePartial (Método)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"damthPopulatePartialC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"damthPopulatePartialX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"damthPopulatePartialA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"damthPopulatePartialS"}

Sincroniza todos los cambios en la réplica parcial con la réplica completa, elimina todos los registros en la réplica parcial y luego vuelve a rellenar la réplica parcial en base a los filtros de réplica actuales. (sólo bases de datos Microsoft Jet .)

Sintaxis

basededatos.**PopulatePartial** *nombrebasededatos*

La sintaxis del método **PopulatePartial** tiene las siguientes partes.

Parte	Descripción
basededatos	Una variable de objeto que referencia el objeto **Database** de la réplica parcial que quiere rellenar.
nombrebasededatos	Un valor de tipo **String** que especifica la ruta de acceso y el nombre de la réplica completa desde la que rellena registros.

Comentarios

Cuando sincroniza una réplica parcial con una réplica completa, es posible crear registros "huérfanos" en la réplica parcial. Por ejemplo, suponga que tiene una tabla Clientes con su propiedad **ReplicaFilter** establecido a `"Región = 'Lara'"`. Si el usuario cambia la región de los clientes de Lara a Lera en la réplica parcial y luego tiene lugar una sincronización por medio del método **Synchronize**, el cambio se copiará a la réplica completa, pero el registro que contenga Lera en la réplica parcial está huérfano, porque no cumple con los criterios del filtro de la réplica.

Para resolver el problema, puede utilizar el método **PopulatePartial**. El método **PopulatePartial** es similar al método **Synchronize**, pero sincroniza todos los cambios con la réplica completa, elimina todos los registros en la réplica parcial y luego vuelve a rellenar la réplica parcial en base a los filtros de réplica actuales. Incluso si sus filtros de réplica no han cambiado, el método **PopulatePartial** siempre eliminará todos los registros en la réplica parcial y la volverá a rellenar en base a los filtros actuales.

De manera general, debería utilizar el método **PopulatePartial** cuando cree una réplica parcial y siempre que cambie los filtros de réplica. Si su aplicación cambia filtros de réplica, debería seguir estos pasos:

1. Sincronizar su réplica completa con la réplica parcial en la que se cambian los filtros.
2. Utilizar las propiedades **ReplicaFilter** y **PartialReplica** para realizar los cambios deseados en el filtro de réplica.
3. Llamar al método **PopulatePartial** para eliminar todos los registros de la réplica parcial y transferir todos los registros de la réplica completa que cumplan con los requisitos del filtro de réplica nuevo.

Si un filtro de réplica ha cambiado y se llama al método **Synchronize** antes de llamar a **PopulatePartial**, tendrá lugar un error interceptable.

El método **PopulatePartial** solamente se puede llamar en una réplica parcial que se ha abierto en modo exclusivo. Además, no puede llamar al método **PopulatePartial** desde el código que está ejecutando en la réplica parcial misma. Lo que puede hacer es abrir la réplica parcial en modo exclusivo desde la réplica completa u otra base de datos y luego llamar a **PopulatePartial**.

Nota A pesar de que el método **PopulatePartial** lleva a cabo una sincronización de sentido único

antes de limpiar y volver a rellenar la réplica parcial, es una buena idea llamar a **Synchronize** antes de llamar a **PopulatePartial**. La razón de esto es que si la llamada a **Synchronize** falla, tiene lugar un error interceptable. Puede utilizar este error para decidir si sigue o no con el método **PopulatePartial** (que elimina todos los registros en la réplica parcial). Si se llama a **PopulatePartial** sola y ocurre un error mientras se están sincronizando los registros, todavía se eliminarán los registros en la réplica parcial, lo que puede que no sea el resultado deseado.

AbsolutePosition (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproAbsolutePositionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproAbsolutePositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproAbsolutePositionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproAbsolutePositionS"}

Establece o devuelve el número de registro relativo de un registro activo del objeto **Recordset**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long**. El valor es un entero que va de 0 a uno menos del número de registros del objeto **Recordset**. Corresponde a la posición ordinal del registro activo en el objeto **Recordset**.

Comentarios

La propiedad **AbsolutePosition** le permite situar el puntero del registro activo en un registro específico basado en su posición ordinal en un objeto **Recordset** de tipo Dynaset o Snapshot . También puede determinar el número del registro activo examinando el valor de la propiedad **AbsolutePosition**.

Dado que el valor de la propiedad **AbsolutePosition** parte de cero (es decir, un valor 0 hace referencia al primer registro del objeto **Recordset**), no se puede establecer para la misma un valor superior o igual al número de registros ocupados. Si se hiciera así, se generaría un error interceptable. Puede determinar el número de registros ocupados en el objeto **Recordset** verificando el valor de la propiedad **RecordCount**. El valor máximo permitido para **AbsolutePosition** es **RecordCount** – 1.

Si no existe registro activo, como cuando no hay registros en el objeto **Recordset**, el valor es -1. Si se elimina el registro activo, no estará definido el valor de la propiedad **AbsolutePosition**, y se produciría un error interceptable si se hace referencia al mismo. Los nuevos registros se agregan al final de la secuencia.

Esta propiedad no está concebida para usarse como un número de registro sustituto. Los marcadores son todavía la forma recomendada de mantener y volver a una posición determinada y son la única manera de situar el registro activo en todos los tipos de objetos **Recordset**. En concreto, la posición de un registro concreto cambia cuando se eliminan uno o más registros que preceden al mismo. No se puede asegurar que un registro determinado vaya a tener la misma posición absoluta si se vuelve a crear el objeto **Recordset** , puesto que el orden de los registros individuales dentro de un objeto **Recordset** no está garantizado, salvo si ha sido creado con una instrucción SQL usando una cláusula ORDER BY.

Notas

- Estableciendo la propiedad **AbsolutePosition** a un valor mayor que cero en un objeto **Recordset** abierto nuevamente, pero sin llenar provoca un error interceptable. Primero llene el objeto **Recordset** con el método **MoveLast**.
- La propiedad **AbsolutePosition** no está disponible en un **Recordset** de tipo Snapshot de movimiento progresivo o en un objeto **Recordset** abierto desde una consulta de paso a través en una base de datos Microsoft Jet conectada a ODBC.

AllowZeroLength (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproAllowZeroLengthC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproAllowZeroLengthX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproAllowZeroLengthA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproAllowZeroLengthS"}

Indica si una cadena de longitud cero ("") es un valor válido para la propiedad **Value** de un objeto **Field** con un tipo de datos **Texto** o **Memo**.

Valores que se pueden establecer y obtener

El valor que se obtiene es un tipo de datos **Boolean** que indica si se admite un valor. El valor es **True** si el objeto **Field** admite una cadena de longitud cero como la propiedad **Value**; el valor predeterminado es **False**.

Comentarios

Para un objeto todavía no anexado a la colección **Fields**, esta propiedad es de lectura/escritura.

Una vez que se anexa a la colección **Fields**, la disponibilidad de la propiedad **AllowZeroLength** depende del objeto que contiene la colección **Fields**, como se muestra en la siguiente tabla.

Si la colección Fields pertenece a	AllowZeroLength es
Index (Objeto)	No se admite
QueryDef (Objeto)	Sólo lectura
Recordset (Objeto)	Sólo lectura
Relation (Objeto)	No se admite
TableDef (Objeto)	Lectura/escritura

Puede utilizar esta propiedad junto con la propiedad **Required**, **ValidateOnSet**, o **ValidationRule** para determinar la validez del valor de un campo.

AllPermissions (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproAllPermissionsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproAllPermissionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproAllPermissionsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproAllPermissionsS"}

Indica todas las autorizaciones que se aplican a la propiedad **UserName** actual del objeto **Container** o **Document**, incluyendo las autorizaciones que son específicas para el usuario, así como las autorizaciones que hereda un usuario de los miembros de los grupos (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

Para cualquier objeto **Container** o **Document**, el valor que se obtiene es un tipo de datos **Long** o una constante que puede incluir lo siguiente.

Constante	Descripción
dbSecReadDef	Puede leer la definición de tabla, incluyendo información de columna e índice.
dbSecWriteDef	Puede modificar o eliminar la definición de tabla, incluyendo la información de columna e índice.
dbSecRetrieveData	Puede recuperar datos desde el objeto **Document**.
dbSecInsertData	Puede agregar registros.
dbSecReplaceData	Puede modificar registros.
dbSecDeleteData	Puede eliminar registros.

Además, el objeto **Container** de las bases de datos o cualquier objeto **Document** de una colección **Documents** puede incluir lo siguiente:

Constante	Descripción
dbSecDeleteData	Puede borrar registros.
dbSecDBAdmin	Puede realizar réplicas de bases de datos y cambiar la contraseña de la base de datos.
dbSecDBCreate	Puede crear nuevas bases de datos. Este valor sólo es válido en el contenedor Databases en el archivo de información del grupo de trabajo (System.mdw).
dbSecDBExclusive	Tiene acceso exclusivo a la base de datos.
dbSecDBOpen	Puede abrir la base de datos.

Comentarios

Esta propiedad contrasta con la propiedad **Permissions**, que define sólo las autorizaciones que son específicas para el usuario y no incluye ninguna autorización que pueda tener también el usuario como miembro de los grupos. Si el valor actual de la propiedad **UserName** es un grupo, **AllPermissions** muestra los mismos valores que **Permissions**.

Attributes (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproAttributesC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproAttributesX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproAttributesA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproAttributesS"}

Indica una o más características de un objeto **Field**, **Relation**, o **TableDef**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** y el valor predeterminado es 0.

Para un objeto **Field**, este valor especifica las características del campo representadas por el objeto **Field** y puede ser la suma de estas constantes:

Constante	Descripción
dbAutoIncrField	El valor del campo para los nuevos registros se incrementa de forma automática en un entero único de tipo **Long** que no se puede cambiar (en un espacio de trabajo Microsoft Jet, admitido sólo para tablas de base de datos Microsoft Jet(.mdb)).
dbDescending	El campo se ordena en sentido descendente (Z-A o 100-0); sólo aplicable a los objetos **Field** de una colección **Fields** de un objeto **Index**. Si omite esta constante, el campo se ordena en sentido ascendente (A-Z o 0-100). Este es el valor predeterminado para los campos **Index** y **TableDef** (sólo en espacios de trabajo Microsoft Jet).
dbFixedField	El tamaño del campo es fijo (predeterminado para campos tipo Numérico).
dbHyperlinkField	El campo contiene información de vínculo (sólo campos tipo Memo).
dbSystemField	El campo almacena la información de réplica para las réplicas; no puede eliminar este tipo de campo (sólo en espacios de trabajo Microsoft Jet).
dbUpdatableField	El valor del campo puede cambiar.
dbVariableField	El tamaño del campo es variable (sólo campos de tipo Texto).

En el caso de un objeto **Relation**, el valor especifica las características de la relación representada por el objeto **Relation** y puede ser la suma de estas constantes:

Constante	Descripción
dbRelationUnique	La relación es uno a uno.
dbRelationDontEnforce	La relación no es impuesta (no hay integridad referencial).
dbRelationInherited	La relación existe en una base de datos no activa que contiene las dos tablas vinculadas.
dbRelationUpdateCascade	Las actualizaciones se realizarán en cascada.
dbRelationDeleteCascade	Las eliminaciones se realizarán en cascada.

Nota Si establece la propiedad **Attributes** del objeto **Relation** para activar las operaciones de cascada, el motor de base de datos Microsoft Jet actualizará o eliminará automáticamente los

registros de una o más tablas cuando se realicen cambios en las tablas de claves principales.

Por ejemplo, supongamos que establece una relación tipo eliminar en cascada entre una tabla de clientes y otra de pedidos. Cuando elimina registros de la tabla de clientes, se eliminan también los registros de la tabla de pedidos relacionados con ese cliente. Además, si establece relaciones de eliminación de cascada entre la tabla de pedidos y otras tablas, los registros de esas tablas se eliminarán automáticamente cuando se eliminen registros de la tabla de clientes.

Para un objeto **TableDef**, este valor especifica las características de la tabla representada por el objeto **TableDef** y puede ser una suma de las siguientes constantes de tipo **Long**.

Constante	Descripción
dbAttachExclusive	Para bases de datos que utilizan el motor de base de datos Microsoft Jet, indica que la tabla es una tabla adjunta abierta para uso exclusivo. Es posible establecer esta constante en un objeto **TableDef** añadido para un tabla local, pero no para una tabla remota.
dbAttachSavePWD	En las bases de datos que usan el motor de base de datos Microsoft Jet, indica que la identificación de usuario y la contraseña para la tabla conectada de forma remota se guardan junto con la información de la conexión. Es posible establecer esta constante en un objeto **TableDef** añadido para un tabla remota, pero no para una tabla local.
dbSystemObject	Indica que la tabla es una tabla del sistema proporcionada por el motor de base de datos Microsoft Jet. Es posible establecer esta constante en un objeto **TableDef** añadido.
dbHiddenObject	Indica que la tabla es una tabla oculta suministrada por el motor de base de datos Microsoft Jet. Es posible establecer esta constante en un objeto **TableDef** añadido.
dbAttachedTable	Indica que la tabla es una tabla adjunta de una base de datos que no es de ODBC como Microsoft Jet o Paradox (de sólo lectura).
dbAttachedODBC	Indica que la tabla es una tabla adjunta de una base de datos ODBC, como el Microsoft SQL Server (de sólo lectura).

Comentarios

Para un objeto todavía no anexado a una colección, esta propiedad es de lectura/escritura.

En un objeto **Field**, el uso de la propiedad **Attributes** depende del objeto que contenga la colección **Fields**.

Si el objeto Field pertenece a un	Attributes es
Index (Objeto)	Lectura/escritura hasta que el **TableDef** al que pertenece **Index** sea añadido a una **Database**; la propiedad es de sólo lectura.
QueryDef (Objeto)	Sólo lectura
Recordset (Objeto)	Sólo lectura
Relation (Objeto)	No se admite

TableDef (Objeto) Lectura/escritura

Para un objeto **Relation**, el valor de la propiedad **Attributes** es de sólo lectura para un objeto añadido a una colección.

Para un objeto **TableDef** anexado, la propiedad es lectura/escritura, aunque no puede establecer todas las constantes si el objeto se anexa, como se especifica en Configuraciones y valores devueltos.

Al establecer varios atributos, puede combinar los mismos sumando las constantes adecuadas. Se ignora cualquier valor no significativo no se producirá un error.

BOF, EOF (Propiedades)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproBOFC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproBOFX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproBOFA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproBOFS"}

- **BOF**: indica si la posición del registro activo es anterior al primer registro de un objeto **Recordset**.
- **EOF**: indica si la posición del registro activo es posterior al último registro de un objeto **Recordset**.

Valores que se pueden obtener

Los valores que se obtienen para las propiedades **BOF** y **EOF** son un tipo de datos **Boolean**.

La propiedad **BOF** devuelve el valor **True**, si la posición del registro activo es anterior al primer registro y **False** si el registro activo está en el registro activo o en el posterior.

La propiedad **EOF** devuelve el valor **True**, si la posición del registro activo es posterior al último registro y **False** si el registro activo está en el último registro o antes de éste.

Comentarios

Puede utilizar las propiedades **BOF** y **EOF** para determinar si un objeto **Recordset** contiene registros o si se han sobrepasado los límites de un objeto **Recordset** al ir pasando de un registro a otro.

La ubicación del puntero de l registro activo determina los valores devueltos de **BOF** y **EOF**.

Si **BOF** o **EOF** tienen el valor **True**, no existe registro activo.

Si abre un objeto **Recordset** que no contiene ningún registro, tanto **BOF** como **EOF** tendrán el valor **True** y el valor de la propiedad **RecordCount** será 0. Cuando se abre un objeto **Recordset** que contiene al menos un registro, el primer registro pasa a ser el registro activo y tanto **BOF** como **EOF** tendrán el valor **False**; que conservará hasta que se sobrepase el principio o el final del objeto **Recordset**, mediante los métodos **MovePrevious** o **MoveNext** respectivamente. Cuando se sobrepasan el principio o el final del conjunto de registros, no hay registro activo o no existe ningún registro.

Si elimina el último registro que queda en el objeto **Recordset**, tanto **BOF** como **EOF** podrán seguir teniendo el valor **False** hasta que intente volver a situar el registro activo.

Si utiliza el método **MoveLast** en un objeto **Recordset** que contiene registros, el último registro pasará a ser el registro activo. Si a continuación utiliza el método **MoveNext**, el registro activo dejará de ser válido y **EOF** pasará a tener el valor **True**. Por el contrario, si utiliza el método **MoveFirst** en un objeto **Recordset** que contiene registros, el primer registro pasa a ser el registro activo. Si a continuación utiliza el método **MovePrevious**, el registro activo dejará de ser válido y **BOF** asumirá el valor **True**.

Normalmente, al trabajar con todos los registros de un objeto **Recordset**, el código irá pasando de un registro a otro mediante **MoveNext,** hasta que la propiedad **EOF** presente el valor **True**.

Si utiliza **MoveNext** cuando **EOF** tiene el valor **True** o **MovePrevious** cuando **BOF** tiene el valor **True**, se producirá un error.

Esta tabla muestra qué métodos **Move** están admitidos con las distintas combinaciones de **BOF**/**EOF**.

	MoveFirst, MoveLast	**MovePrevious, Move** < 0	**Move** 0	**MoveNext, Move** > 0
BOF=True, EOF=False	Permitido	Error	Error	Permitido
BOF=False, EOF=True	Permitido	Permitido	Error	Error
Ambos **True**	Error	Error	Error	Error

Ambos **False**	Permitido	Permitido	Permitido	Permitido

Si se permite usar un método Move, ello no implica necesariamente que el método vaya a situar de forma correcta un registro. Simplemente indica que se permite intentar utilizar el método de movimiento en cuestión, sin que se produzca ningún error. El estado de los indicadores de **BOF** y **EOF** puede cambiar como resultado de intentar la operación Move.

El método **OpenRecordset** invoca internamente **MoveFirst**. Por tanto, un **OpenRecordset** en un conjunto de registros vacío provoca que tanto **BOF** como **EOF** tengan el valor **True**. (Consulte la tabla que sigue para comprobar el comportamiento de un método **MoveFirst** que ha fallado).

Todos los métodos Move que sitúan sin problemas un registro borran (con valor **False**) tanto **BOF** como **EOF**.

En un espacio de trabajo Microsoft Jet, si agrega un registro a un objeto **Recordset** vacío, **BOF** pasará a **False**, pero **EOF** permanecerá en **True**, que indica que la posición actual es al final del objeto **Recordset**. En un espacio de trabajo de ODBCDirect, tanto **BOF** como **EOF** pasarán a **False**, que indica que la posición actual es la del registro nuevo.

Ningún método **Delete**, incluso si se elimina el único registro que quede en un conjunto de registros, modificará el valor de **BOF** ni el de **EOF**.

En la tabla que sigue se muestra el efecto que tienen los métodos Move que no sitúan un registro en los parámetros de **BOF** y **EOF**.

	BOF	**EOF**
MoveFirst, **MoveLast**	**True**	**True**
Move 0	Sin cambio	Sin cambio
MovePrevious, **Move** < 0	**True**	Sin cambio
MoveNext, **Move** > 0	Sin cambio	**True**

Bookmark (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproBookmarkC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproBookmarkX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproBookmarkA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproBookmarkS"}

Establece o devuelve un marcador que identifica de forma única el registro activo en un objeto **Recordset**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es una expresión de cadena o expresión Variant que constituye un marcador válido. El tipo de datos es una matriz de tipo **Variant** con datos de tipo **Byte**.

Comentarios

Para un objeto **Recordset** basado en su totalidad en tablas de Microsoft Jet, el valor de la propiedad **Bookmarkable** es **True** y se pueden utilizar marcadores. Sin embargo, puede que otros productos de base de datos no admitan marcadores. Por ejemplo, no puede usar marcadores en ningún objeto **Recordset** basado en una tabla de Paradox adjunta que no posea ninguna clave principal.

Cuando se crea o abre un objeto **Recordset**, cada uno de sus registros tiene ya un marcador único. Puede guardar el marcador del registro activo asignando el valor de la propiedad **Marcador (Bookmark)** a una variable. Para volver rápidamente a ese registro después de haber pasado a otro registro, basta con establecer el valor de esta variable en la propiedad **Bookmark** del objeto **Recordset**.

No hay ningún límite en el número de marcadores que se pueden establecer. Para crear un marcador para un registro distinto del activo, desplácese hasta ese registro y asigne el valor de la propiedad **Bookmark** a una variable de tipo **String** que identifique el registro.

Para asegurarse de que el objeto **Recordset** admite marcadores, observe el valor de su propiedad **Bookmarkable** antes de utilizar la propiedad **Bookmark**. Si **Bookmarkable** es **False**, indicará que el objeto **Recordset** no admite marcadores y si se utiliza la propiedad **Bookmark** se producirá un error interceptable.

Si crea una copia de un objeto **Recordset** utilizando el método **Clone**, los valores de **marcador** del objeto **Recordset** original y del duplicado serán idénticos y se podrán utilizar indistintamente. Sin embargo, no es posible utilizar del mismo modo marcadores de distintos objetos **Recordset**, aunque se hayan creado utilizando el mismo objeto o con la misma instrucción del SQL.

Si establece como propiedad **Bookmark** un valor que represente un registro eliminado, se producirá un error interceptable.

El valor de la propiedad **Bookmark** es distinto del número de registro.

Bookmarkable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproBookmarkableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproBookmarkableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproBookmarkableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproBookmarkableS"}

Devuelve un valor que indica si el objeto **Recordset** admite marcadores que el usuario puede establecer mediante la propiedad **Bookmark**.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que devuelve **True** si el objeto admite marcadores.

Comentarios

Compruebe el valor de la propiedad **Bookmarkable** antes de intentar establecer o utilizar la propiedad **Bookmark**, para asegurarse así de que el objeto **Recordset** admite marcadores.

En el caso de un objeto **Recordset** basado por completo en tablas de Microsoft Jet, el valor de la propiedad **Bookmarkable** es **True**, por lo que se pueden utilizar marcadores. Sin embargo, puede que otros productos de bases de datos no admitan marcadores. Por ejemplo, no puede utilizar marcadores en un objeto **Recordset** basado en una tabla vinculada Paradox adjunta que no tenga clave principal.

CacheSize (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproCacheSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproCacheSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproCacheSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproCacheSizeS"}

Establece o devuelve el número de registros recuperados de un origen de datos ODBC que será almacenado localmente en memoria caché.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** y debe estar comprendido entre 5 y 1200, pero nunca mayor se permitirá mayor que la memoria disponible. Un valor normal es 100. Un valor de 0 desactiva la memoria caché.

Comentarios

Los datos almacenados en memoria caché mejoran el rendimiento si utiliza objetos **Recordset** para recuperar datos de un servidor remoto. Una memoria caché es el espacio de la memoria local que almacena los datos leídos más recientemente del servidor, en el supuesto de que se pueden volver a solicitar los datos de nuevo mientras se está ejecutando la aplicación. Cuando se solicitan los datos, el motor de base de datos Microsoft Jet examina la memoria caché en busca de los datos solicitados antes de recuperarlos del servidor, que es un proceso más lento. Los datos que no proceden de un origen de datos ODBC no se almacenan en la memoria caché .

Cualquier origen de datos ODBC conectado a Microsoft Jet, como una tabla adjunta, puede tener un caché local. Para crear la memoria caché , abra un objeto **Recordset** desde el origen de datos remoto, establezca las propiedades **CacheSize** y **CacheStart** y, a continuación, utilice el método **FillCache** o desplácese por los registros por medio de los métodos Move.

Un espacio de trabajo de ODBCDirect puede utilizar una memoria caché local. Para crear la memoria caché, establezca la propiedad **CacheSize** en un objeto **QueryDef**. En un objeto **Relation**, **CacheSize** es de sólo lectura y depende del valor de la propiedad **CacheSize** de un objeto **QueryDef**. No puede utilizar la propiedad **CacheStart** en el método **FillCache** en un espacio de trabajo de ODBCDirect.

El valor de la propiedad **CacheSize** se puede basar en el número de registros con los que puede trabajar la aplicación al mismo tiempo. Por ejemplo, si está usando un objeto **Recordset** como origen de los datos que se mostrarán en la pantalla, puede establecer su propiedad **CacheSize** en 20 para mostrar 20 registros a la vez.

El motor de base de datos Microsoft Jet solicitará al caché los registros que se encuentren dentro del rango de la memoria caché y solicitará al servidor los registros que se encuentren fuera.

Los registros leídos desde la memoria caché no reflejarán los cambios realizados de forma concurrente en los datos de origen por otros usuarios.

Para forzar una actualización de todos los datos de la memoria caché , establezca como 0 la propiedad **CacheSize** del objeto **Recordset**, establézcala luego como el tamaño de la memoria caché que solicitó originalmente y, a continuación, utilice el método **FillCache**.

Clustered (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproClusteredC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproClusteredX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproClusteredA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproClusteredS"}

Establece o devuelve un valor que indica si un objeto **Index** representa un índice por grupos en una tabla (sólo en espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** si el objeto **Index** representa un índice por grupos.

Comentarios

Algunos formatos de base de datos de escritorio ISAM utilizan índices por grupos. Un índice agrupado consta de uno o más campos no clave que, todos juntos, organizan todos los registros de una tabla en un orden predefinido. Un índice agrupado ofrece una manera eficiente de acceder a los registros de una tabla en los que la valores de índice pueden no ser únicos.

La propiedad **Clustered** es de lectura/escritura en un objeto **Index** nuevo que no se ha anexado a una colección y de sólo lectura en un objeto **Index** existente en una colección **Indexes**.

Notas

- La propiedad **Clustered** se ignora para las bases de datos que utilizan las bases de datos Microsoft Jet ya que el motor de base de datos Microsoft Jet no admite índices agrupados.
- Para datos de origen ODBC el valor de la propiedad **Clustered** es siempre False; no detecta si la base de datos ODBC tiene o no un índice agrupado.

CollatingOrder (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproCollatingOrderC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproCollatingOrderX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproCollatingOrderA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproCollatingOrderS"}

Especifica la secuencia de ordenación de caracteres de texto para comparación de cadenas u ordenación de las mismas (sólo en espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Long** o una constante que puede tener uno de los siguientes valores.

Constante	Orden
dbSortGeneral	Utiliza la secuencia de ordenación general (inglés, francés, alemán, portugués, italiano y español moderno).
dbSortArabic	Utiliza la secuencia de ordenación árabe.
dbSortChineseSimplified	Chino simplificado
dbSortChineseTraditional	Chino tradicional
dbSortCyrillic	Utiliza la secuencia de ordenación rusa.
dbSortCzech	Utiliza la secuencia de ordenación checo.
dbSortDutch	Utiliza la secuencia de ordenación holandés.
dbSortGreek	Utiliza la secuencia de ordenación griego.
dbSortHebrew	Utiliza la secuencia de ordenación hebreo.
dbSortHungarian	Utiliza la secuencia de ordenación húngaro.
dbSortIcelandic	Utiliza la secuencia de ordenación islandés.
dbSortJapanese	Utiliza la secuencia de ordenación japonés.
dbSortKorean	Utiliza la secuencia de ordenación coreano.
dbSortNeutral	Utiliza la secuencia de ordenación neutral.
dbSortNorwDan	Utiliza la secuencia de ordenación noruego o danés.
dbSortPDXIntl	Utiliza la secuencia de ordenación internacional de Paradox.
dbSortPDXNor	Utiliza la secuencia de ordenación noruego o danés de Paradox.
dbSortPDXSwe	Utiliza la secuencia de ordenación noruego o danés de Paradox.
dbSortPolish	Utiliza la secuencia de ordenación polaco.
dbSortSlovenian	Esloveno
dbSortSpanish	Utiliza la secuencia de ordenación español.
dbSortSwedFin	Utiliza la secuencia de ordenación sueco o finlandés de Paradox.
dbSortThai	Tailandés
dbSortTurkish	Utiliza la secuencia de ordenación turco.
dbSortUndefined	La secuencia de ordenación es indefinida o desconocida.

Comentarios

La disponibilidad de la propiedad **CollatingOrder** depende del objeto que contiene la colección

Fields, como se muestra en la siguiente tabla.

Si la colección Fields pertenece a un	CollatingOrder es
Index (Objeto)	No se admite
QueryDef (Objeto)	Sólo lectura
Recordset (Objeto)	Sólo lectura
Relation (Objeto)	No se admite
TableDef (Objeto)	Sólo lectura

El valor de la propiedad **CollatingOrder** corresponde al argumento *escenario* del método **CreateDatabase** cuando se creó la base de datos o del método **CompactDatabase** cuando la base de datos se compactó por última vez.

Compruebe el valor de la propiedad **CollatingOrder** de un objeto **Database** o **Field** para conocer el método de comparación de cadenas de la base de datos o del campo. Puede establecer la propiedad **CollatingOrder** de un nuevo objeto **Field** si desea que el valor del objeto **Field** sea diferente que el del objeto **Database** que lo contiene.

Los valores de las propiedades **CollatingOrder** y **Attributes** de un objeto **Field** en una colección **Fields** de un objeto **Index** determinan la secuencia y el sentido de ordenación de los caracteres en un índice. Sin embargo, no se puede establecer una secuencia de ordenación para un índice individual - para una tabla completa.

ConflictTable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproConflictTableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproConflictTableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproConflictTableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproConflictTableS"}

Devuelve el nombre de una tabla de conflicto que contiene los registros de la base de datos que entraban en conflicto durante la sincronización de las dos réplicas (sólo en espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **String** que es una cadena de longitud cero si no hay una tabla de conflicto o la base de datos no es una réplica.

Comentarios

Si dos usuarios de dos réplicas distintas realizan cada uno una modificación en el mismo registro de la base de datos, los cambios realizados por uno de los usuarios no se podrán aplicar a la otra réplica. Por tanto, el usuario que no consiguió realizar el cambio debe solucionar el problema mediante un proceso denominado "resolución de conflictos".

Los conflictos se producen en el nivel de los registros, no entre los campos. Por ejemplo, si un usuario cambia el campo Dirección y otro actualiza el campo Teléfono en el mismo registro, se rechazará uno de los cambios. Puesto que el conflicto se produce en el nivel del registro, el rechazo se produce, aunque es muy poco probable que el cambio aceptado y el cambio rechazado den como resultado un verdadero conflicto de información.

El mecanismo de sincronización maneja los conflictos de los registros mediante la creación de tablas de conflictos, que contienen la información que se habría incorporado a la tabla, si el cambio se hubiera realizado sin problemas. Como programador DAO, usted examinará estas tablas de conflictos y trabaja con ellas fila a fila, solucionando los temas que lo requieran.

Todas las tablas de conflicto se denominan *tabla*_conflict, siendo *tabla* el nombre original de la tabla, truncado a la longitud máxima para nombres de tabla.

Connect (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproConnectC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproConnectX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproConnectA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproConnectS"}

Proporciona información acerca del origen de una base de datos abierta, de una base de datos utilizada en una consulta de paso a través o de una tabla adjunta. Para los objetos **Database**, **Connection**, tablas adjuntas y objetos **TableDef** que todavía no se han añadido a una colección, el valor de esta propiedad es de lectura/escritura. Para objetos **QueryDef** y tabla subyacente, esta propiedad es de sólo lectura.

Sintaxis

objeto.**Connect** = *tipobasedatos*;*parámetros*;

La sintaxis de la propiedad **Connect** consta de las siguientes partes:

Parte	Descripción
objeto	Una expresión de objeto que es definida con un objeto en la lista "Aplicable a".
tipobasedatos	Opcional. Los datos del tipo **String** especifican un tipo de base de datos. Para las bases de datos Microsoft Jet, se excluye este argumento; si especifica *parámetros*, utilice un punto y coma (;) como una reserva de espacio.
parámetros	Opcional. Los datos de tipo **String** especifica los parámetros adicionales que se deben trasladar a ODBC o los controladores ISAM instalables. Utilice punto y coma para separar los parámetros.

Valores

El valor de la propiedad **Connect** es del tipo **String** que consta de un especificador del tipo de base de datos y cero o más parámetros separados por punto y coma. La propiedad **Connect** se usa para trasladar información adicional a ODBC y a algunos controladores ISAM según sea necesario.

Para realizar una consulta SQL de paso a través en una tabla adjunta a su archivo .mdb, debe establecer primero la propiedad **Connect** de la base de datos de la tabla adjunta como una cadena de conexión ODBC válida.

Para un objeto **TableDef** que represente una tabla adjunta, el valor de la propiedad **Connect** consta de una o más partes (un especificador del tipo de base de datos y una ruta de acceso a la base de datos), cada uno de las cuales termina con un punto y coma.

La ruta, como se muestra en la tabla que sigue, es la ruta completa del directorio donde se encuentran los archivos de la base de datos y debe ir precedida del identificador "DATABASE=". En algunos casos (como por ejemplo en las bases de datos Microsoft Jet y Microsoft Excel) se incluye un nombre de archivo específico en el argumento de la ruta de la base de datos.

La tabla siguiente muestra los tipos posibles de bases de datos y sus correspondientes especificador y ruta para el valor de la propiedad **Connect**. También puede especificar "FTP://*ruta*/*etc.*" o "HTTP://*ruta*/*etc.*" para la ruta de acceso.

Tipo de base de datos	Especificador	Ejemplo
Base de datos Microsoft Jet	[*basededatos*];	"*unida*d:*ruta**nombrearchivo.mdb*"
dBASE III	dBASE III;	"*unidad*:*ruta*"

dBASE IV	dBASE IV;	"*unidad*:*ruta*"
dBASE 5	dBASE 5.0;	"*unidad*:*ruta*"
Paradox 3.x	Paradox 3.x;	"*unidad*:*ruta*"
Paradox 4.x	Paradox 4.x;	"*unidad*:*ruta*"
Paradox 5.x	Paradox 5.x;	"*unidad*:*ruta*"
FoxPro 2.0	FoxPro 2.0;	"*unidad*:*ruta*"
FoxPro 2.5	FoxPro 2.5;	"*unidad*:*ruta*"
FoxPro 2.6	FoxPro 2.6;	"*unidad*:*ruta*"
Excel 3.0	Excel 3.0;	*"unidad:\ruta\nombrearchivo.xls"*
Excel 4.0	Excel 4.0;	*"unidad:\ruta\nombrearchivo.xls"*
Excel 5.0 or Excel 95	Excel 5.0;	*"unidad:\ruta\nombrearchivo.xls"*
Excel 97	Excel 8.0;	*"unidad:\ruta\nombrearchivo.xls"*
HTML Import	HTML Import;	*"unidad:\ruta\nombrearchivo"*
HTML Export	HTML Export;	*"unidad:\ruta"*
Text	Text;	*"unidad:\ruta"*
ODBC	ODBC; DATABASE=*basededatos*; UID=*usuario*; PWD=*contraseña*; DSN= *nombreorigendatos*; [LOGINTIMEOUT=*segundos*;]	Ninguna
Exchange	Exchange; MAPILEVEL=*rutadecarpeta*; [PROFILE=*perfil*;] [PWD=*conmtraseña*;] [DATABASE=*basededatos*;]	*"unidad:* *\ruta\nombrearchivo.mdb"*

Comentarios

Si el especificador es sólo `"ODBC;"`, se mostrará un cuadro de diálogo que muestra todos los nombres de orígenes de datos ODBC registrados por medio del controlador ODBC, para que el usuario pueda elegir una base de datos.

Si se requiere una contraseña, pero ésta no se incluye en el valor de la propiedad **Connect**, la primera vez que el controlador ODBC accede a una tabla, se muestra un cuadro de diálogo de registro y se mostrará otra vez si se cierra la conexión y se vuelve a abrir.

Para datos de Microsoft Exchange, la clave requerida MAPILEVEL debería ser establecida con el nombre completo de una ruta de acceso existente (por ejemplo, "Mailbox - Pat SmithIAlpha/Today"). La ruta de acceso no incluye el nombre de la carpeta que se abrirá como una tabla; ese nombre de carpeta no debe especificarse como el argumento *name* del método **CreateTable**. La clave TABLETYPE debe establecerse a "0" para abrir una carpeta (predeterminada) o a "1" para abrir una libreta de direcciones. La clave PROFILE se establecerá de forma predeterminada con el perfil actualmente en uso.

Para tablas de la base de datos Microsoft Jet, el valor de la propiedad **Connect** es una cadena de longitud cero ("").

Se puede establecer la propiedad **Connect** para un objeto **Database** proporcionando un argumento *origen* al método **OpenDatabase**. Puede examinar el valor para determinar el tipo, ruta, Id de usuario, contraseña u origen de datos ODBC de la base de datos.

Para un objeto en un espacio de trabajo Microsoft Jet, la propiedad **Connect** se utiliza con la propiedad **ReturnsRecords** para crear una consulta de paso a través. El *tipobasedatos* de la cadena de conexión es "ODBC;" y el resto de la cadena contiene información específica del controlador ODBC utilizado para acceder a los datos remotos. Para obtener más información al respecto, consulte la documentación del controlador específico.

Notas

- Debe establecer la propiedad **Connect** antes de establecer la propiedad **ReturnsRecords**.
- Debe tener permiso de acceso al equipo que contiene el servidor de base de datos a la que intenta tener acceso.

Container (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproContainerX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproContainerA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproContainerS"}

Devuelve el nombre del objeto **Container** al que pertenece el objeto **Document** (sólo en espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **String**.

Count (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproCountC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproCountA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproCountS"}

Indica el número de objetos de una colección.

Valor que se puede obtener

El valor que se obtiene es un tipo de datos **Integer**.

Comentarios

Ya que los miembros de una colección empiezan con 0, debería empezar siempre los bucles de código con el miembro 0 y terminarlos con el valor de la propiedad **Count** menos 1. Si desea hacer un bucle a través de todos los miembros de una colección sin comprobar la propiedad **Count**, puede utilizar un comando **For Each...Next**.

El valor de la propiedad **Count** nunca es **Null**. Si el valor es 0, no habrá ningún objeto en la colección.

DataUpdatable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproDataUpdatableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDataUpdatableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDataUpdatableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDataUpdatableS"}

Devuelve un valor que indica si el dato del campo representado por un objeto **Field** se puede actualizar.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que devuelve **True** si los datos del campo se pueden actualizar.

Comentarios

Utilice esta propiedad para determinar si puede cambiar el valor de la propiedad **Value** de un objeto **Field**. Esta propiedad es siempre **False** en un objeto **Field** cuya propiedad **Attributes** es **dbAutoIncrField**.

Puede utilizar la propiedad **DataUpdatable** en objetos **Field** que se anexan a la colección **Fields** de los objetos **QueryDef**, **Recordset** y **Relation**, pero no la colección **Fields** de objetos **Index** o **TableDef**.

DateCreated, LastUpdated (Propiedades)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproDateCreatedC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDateCreatedX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDateCreatedA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDateCreatedS"}

- **DateCreated** devuelve la fecha y hora en que el objeto se creó (o la tabla base si el objeto es un **Recordset** de tipo Table **Recordset** (sólo espacios de trabajo Microsoft Jet).
- **LastUpdated** devuelve la fecha y hora del último cambio realizado al objeto, o la tabla base si el objeto es un **Recordset** de tipo Table (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Variant** (subtipo **Date/Time**).

Comentarios

Para objetos **Recordset** de tipo Table, los valores de la fecha y la hora provienen del equipo en el que se creó o se actualizó por última vez la tabla base. para otros objetos, **DateCreated** y **LastUpdated** devuelve la fecha y la hora en que se creó o actualizó el objeto. En un entorno multiusuario, los usuarios obtendrían este valor directamente del servidor de archivos para impedir discrepancias en los valores de las propiedades **DateCreated** y **LastUpdated**.

DefaultUser, DefaultPassword (Propiedades)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproDefaultUserC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDefaultUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDefaultUserA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDefaultUserS"}

- **DefaultUser** establece el nombre de usuario utilizado para crear el **Workspace** predeterminado cuando se inicializa.
- **DefaultPassword** establece la contraseña utilizada para crear el **Workspace** predeterminado cuando se inicializa.

Valor

El valor de **DefaultUser** es un tipo de datos **String**. Puede tener entre 1 y 20 caracteres de longitud en espacios de trabajo Microsoft Jet y cualquier longitud en espacios de trabajo ODBCDirect, y admite caracteres alfabéticos, caracteres acentuados, números, espacios y símbolos excepto: " (comillas tipográficas), / (barra), \ (barra invertida), [ ] (corchetes), : (dos puntos), | (barra vertical), < (signo menor o igual), > (signo mayor o igual), + (signo más), = (signo igual), ; (punto y coma), , (coma), ? (interrogación), * (asterisco), espacios en blanco, y caracteres de control (ASCII 00 a ASCII 31).

El valor de **DefaultPassword** es un tipo de datos **String** que puede tener hasta 14 caracteres de longitud en bases de datos Microsoft Jet y cualquier longitud en conexiones ODBCDirect. Puede contener cualquier carácter excepto 0 ASCII.

De modo predeterminado, la propiedad **DefaultUser** se establece a "administrador" y la propiedad **DefaultPassword** se establece a una cadena de longitud cero ("").

Comentarios

Los nombres de usuarios normalmente no distinguen mayúsculas de minúsculas; sin embargo, si está volviendo a crear cuenta de usuario que se eliminó o creó en un grupo de trabajo diferente, el nombre de usuario deben coincidir exactamente las mayúsculas y minúsculas del nombre original. Las contraseñas distinguen mayúsculas de minúsculas.

Normalmente, el método **CreateWorkspace** se utiliza para crear un objeto **Workspace** con un nombre de usuario y una palabra clave determinados. Sin embargo, para conservar la compatibilidad con versiones anteriores y por conveniencia cuando las bases de datos aseguradas no se están utilizando, el motor de la base de datos Microsoft crea automáticamente un objeto **Workspace** predeterminado cuando sea necesario si no hay ya uno abierto. En este caso, los valores de **DefaultUser** y **DefaultPassword** definen el usuario y la palabra clave para el objeto **Workspace** predeterminado.

Para que esta propiedad tenga efecto, debe establecerla antes de llamar a cualquier método DAO.

DefaultValue (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproDefaultValueC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDefaultValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDefaultValueA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDefaultValueS"}

Establece o devuelve el valor predeterminado de un objeto **Field**. Para un objeto **Field** no agregado aún a la colección **Fields**, esta propiedad es de lectura/escritura (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que contiene un máximo de 255 caracteres. Puede ser bien un texto o una expresión. Si el valor de la propiedad es una expresión, no puede contener funciones definidas por el usuario, funciones de agrupamiento SQL del motor de base de datos Microsoft Jet o referencias a consultas, formulario u otros objetos **Field**.

Nota Puede establecer la propiedad **DefaultValue** de un objeto **Field** en un objeto **TableDef** con un valor especial llamado "GenUniqueId()". Esto produce un número aleatorio que se asigna a este campo siempre que se agrega o crea un registro nuevo, de ese modo se da a cada registro un identificador único. La propiedad **Type** del campo debe ser **Long**.

Comentarios

La utilización de la propiedad **DefaultValue** depende del objeto padre de **Fields**, tal y como se muestra en la siguiente tabla:

Si la colección Fields pertenece a un	DefaultValue es
Index (Objeto)	No se admite
QueryDef (Objeto)	De sólo lectura
Recordset (Objeto)	De sólo lectura
Relation (Objeto)	No se admite
TableDef (Objeto)	Lectura/escritura

Cuando se crea un nuevo registro, el valor de la propiedad **DefaultValue** se establece automáticamente como el valor para el campo. Puede cambiar el valor del campo estableciendo su propiedad **Value**.

La propiedad **DefaultValue** no se aplica a los campos de tipo Autonumérico y Objeto binario.

Description (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproDescriptionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDescriptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDescriptionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDescriptionS"}

Devuelve una cadena descriptiva asociada a un error.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **String** que describe el error.

Comentarios

La propiedad **Description** contiene una breve descripción del error. Muestre esta propiedad para alertar al usuario de un error que no pueda o no quiera controlar.

DistinctCount (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproDistinctCountC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDistinctCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDistinctCountA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDistinctCountS"}

Devuelve un valor que indica el número de valores únicos para el objeto **Index** que se incluye en la tabla asociada (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se objeto en un tipo de datos **Long**.

Comentarios

Active la propiedad **DistinctCount** para determinar el número de valores únicos o claves en un índice. Cada clave se cuenta sólo una vez, aun cuando puede haber múltiples ocurrencias de ese valor, si el índice permite valores duplicados. Esta información es útil en aplicaciones que intentan optimizar el acceso a los datos evaluando la información de índices. El número de valores únicos se conoce también como la *cardinalidad* de un objeto **Index**.

La propiedad **DistinctCount** no siempre refleja el número actual de claves en un momento determinado. Por ejemplo, un cambio motivado por la terminación de transacción no se refleja inmediatamente en la propiedad **DistinctCount**. El valor de la propiedad **DistinctCount** tampoco refleja la eliminación de registros con clave única. El número será exacto inmediatamente después de usar el método **CreateIndex**.

EditMode (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproEditModeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproEditModeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproEditModeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproEditModeS"}

Devuelve un valor que indica el estado de edición del registro activo.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Long** que indica el estado de la modificación, como se enumera en la siguiente tabla:

Constante	Descripción
dbEditNone	No hay operaciones de edición en progreso.
DbEditInProgress	El método **Edit** se ha invocado, y el registro activo está en el búfer de copia.
dbEditAdd	El método **AddNew** se ha invocado y el registro activo del búfer de copia es un nuevo registro que no se ha guardado en la base de datos.

Comentarios

La propiedad **EditMode** es útil cuando un proceso de edición se interrumpe, por ejemplo, por un error durante una validación. Puede utilizar el valor de la propiedad **EditMode** para determinar si debería utilizar el método **Update** o **CancelUpdate**.

También puede comprobar si el valor de la propiedad **LockEdits** es **True** y la de la propiedad **EditMode** es **dbEditInProgress** para determinar si la página de datos actual está bloqueada.

Filter (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproFilterC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproFilterX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproFilterA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproFilterS"}

Establece o devuelve un valor que determina los registros incluidos en un **Recordset** abierto (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que contiene la cláusula WHERE de una instrucción SQL sin la palabra reservada WHERE.

Comentarios

Utilice la propiedad **Filter** para aplicar un filtro a un objeto **Recordset** de tipo Dynaset o Snapshot.

Puede utilizar la propiedad **Filter** para restringir los registros que devuelve un objeto ya existente, cuando se abre un nuevo objeto **Recordset** basado en un **Recordset** ya existente.

En muchos casos es más rápido abrir un nuevo objeto **Recordset** utilizando una instrucción SQL que incluya una cláusula WHERE.

Utilice el formato de fecha norteamericano (mes-día-año) cuando filtre campos que contengan fechas, incluso si no está utilizando la versión norteamericana del motor de base de datos Microsoft Jet (en cuyo caso debe ensamblar cualquier fecha concatenando cadenas, por ejemplo, `strMes & "-" & strDía & "-" & strAño`). En caso contrario, los datos no se filtrarán como espera.

Si establece a una cadena concatenada sin ningún valor entero y los parámetros del sistema especifican un signo decimal no de EE.UU. como una coma (por ejemplo, `strFiltro = "PRECIO > " & lngPrecio` y `lngPrecio = 125,50`), se produce un error cuando intenta abrir el siguiente objeto **Recordset**. Esto se produce porque durante la concatenación, el número se convertirá a una cadena utilizando su signo decimal predeterminado del sistema y el SQL de Microsoft Jet sólo acepta signos decimales de EE.UU.

Foreign (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproForeignC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproForeignX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproForeignA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproForeignS"}

Devuelve un valor que indica si un objeto **Index** representa una clave externa para una tabla (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que devuelve **True** si el objeto **Index** representa una clave externa.

Comentarios

Una clave externa consta de uno o más campos de una tabla externa que identifican de forma única todas las filas en la tabla principal.

El motor de base de datos Microsoft Jet crea un objeto **Index** para la tabla externa y establece la propiedad **Foreign** cuando se crea una relación que exige integridad referencial.

ForeignName (Propiedad)

Establece o devuelve un valor que especifica en una relación el nombre del objeto **Field** de la tabla externa que corresponde a un campo de la tabla principal (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener son un tipo de datos **String** que evalúa el nombre de un objeto **Field** en el objeto **TableDef** de la colección **Fields**.

Si el objeto **Relation** no está anexado al objeto **Database**, pero **Field** está anexado al objeto **Relation**, la propiedad **ForeignName** es de lectura/escritura. Una vez que se anexa el objeto **Relation** a la base de datos, la propiedad **ForeignName** es de sólo lectura.

Comentarios

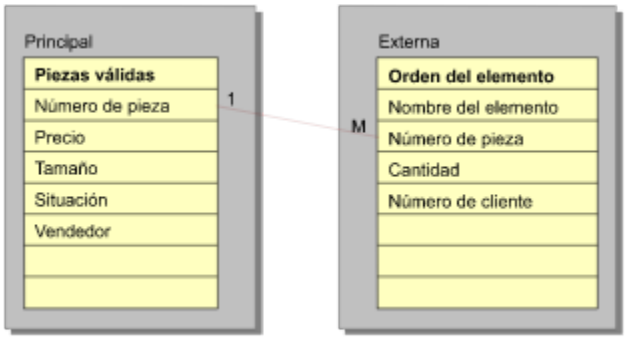
Sólo un objeto **Field** que pertenezca a la colección **Fields** contenida en un objeto **Relation** puede soportar la propiedad **ForeignName**.

Los *valores* de las propiedades **Name** y **ForeignName** para un objeto **Field** especifican los nombres de los campos correspondientes en las tablas principal y externa de la relación. Las configuraciones de un argumento *valor* de las propiedades **Table** y **ForeignTable** de un objeto **Relation** determinan las tablas primarias y externas de la relación.

Por ejemplo, si tiene una lista de códigos de piezas de repuestos válidas (en un campo denominado NúmPieza) almacenadas en una tabla PiezasVálidas, podría establecer una relación con una tabla ElementoPedido en la que si un código de pieza se introduce en la tabla ElementoPedido, tiene que existir también en la tabla PartesVálidas. Si el código de pieza no existe en la tabla PartesVálidas y no se había indicado la exigencia de la integridad referencial en la propiedad **Attributes** del objeto **Relation** a **dbRelationDontEnforce**, se podría producir un error interceptable.

En este caso, la tabla PiezasVálidas sería un ejemplo de tabla externa, así la propiedad **ForeignTable** del objeto **Relation** se podría establecer a PartesVálidas y la propiedad **Table** del objeto **Relation** se podría establecer a ElementoPedido. Las propiedades **Name** y **ForeignName** del objeto **Field** en la colección **Fields** del objeto **Relation** se podría establecer a NúmPieza.

La siguiente ilustración describe lo anterior.



ForeignTable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproForeignTableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproForeignTableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproForeignTableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproForeignTableS"}

Establece o devuelve el nombre de la tabla externa de una relación (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que evalúa el nombre de una tabla en la colección **TableDefs** del objeto **Database**. Esta propiedad es de lectura/escritura para los objetos **Relation** todavía no anexados a una colección y de sólo lectura para un objeto **Relation** existente en la colección **Relations**.

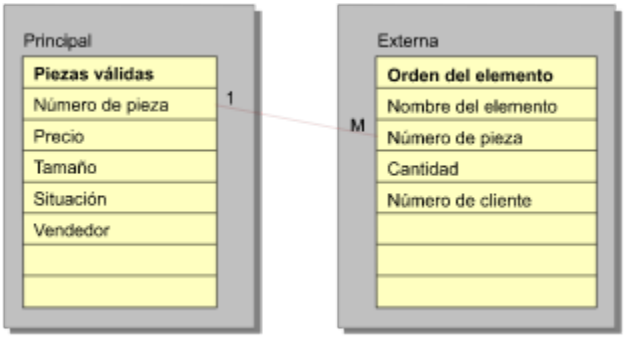
Comentarios

El valor de la propiedad **ForeignTable** de un objeto **Relation** es el valor de la propiedad **Name** del objeto **TableDef** o **QueryDef** que representa la tabla o consulta externa. El valor de la propiedad **Table** es el valor de la propiedad **Name** del objeto **TableDef** o **QueryDef** que representa la tabla principal o consulta primaria.

Por ejemplo, si tiene una lista de códigos de piezas válidas (en un campo llamado NúmPieza) y almacenada en una tabla PiezasVálidas, podría establecer una relación con una tabla ElementosPedidos de forma que si un código de pieza se introduce en la tabla ElementosPedidos, tiene que existir también en la tabla PiezasVálidas. Si no existe el código de pieza en la tabla PiezasVálidas y había establecido la propiedad **Attributes** del objeto **Relation** para **dbRelationDontEnforce** se producirá un error interceptable.

En esta caso, la tabla PartesVálidas es la tabla principal, así que la propiedad **Table** del objeto **Relation** se podría establecer a PartesVálidas y la propiedad **ForeignTable** de objeto **Relation** se podría establecer a ElementoPedido. La propiedad **Name** y la propiedad **ForeignName** del objeto **Field** se establecerían ambas como NúmPieza.

La siguiente ilustración representa la relación descrita anteriormente.



HelpContext, HelpFile (Propiedades)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproHelpContextC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproHelpContextX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproHelpContextA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproHelpContextS"}

- **HelpContext** devuelve un Id de contexto, en forma de una variable de tipo **Long** para un tema del archivo de Ayuda de Microsoft Windows.
- **HelpFile** devuelve un tipo de datos **String** que es una ruta de acceso completamente cualificada para el archivo de Ayuda.

Comentarios

Si se especifica el archivo de ayuda de Microsoft Windows en **HelpFile**, la propiedad **HelpContext** se utiliza para mostrar automáticamente el tema de ayuda que identifica.

Nota Podría escribir rutinas en su aplicación para controlar los errores típicos. Cuando programa un objeto, puede utilizar la ayuda suministrada por el archivo de ayuda del objeto para aumentar la calidad de su control de errores o para mostrar un mensaje informativo al usuario si el error no es recuperable.

IgnoreNulls (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproIgnoreNullsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproIgnoreNullsX":1} {ewc HLP95EN.DLL,DYNALINK,"Se aplica":"daproIgnoreNullsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproIgnoreNullsS"}

Establece o devuelve un valor que indica si los registros que tienen valores **Null** en sus campos de índices tienen entradas de índice (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** si los campos con valores **Null** no tienen una entrada de índice. Esta propiedad es de lectura/escritura para un objeto **Index** nuevo todavía no anexado a una colección y de sólo lectura para un objeto **Index** existente en una colección **Indexes**.

Comentarios

Para acelerar las búsquedas de registros utilizando un campo, puede definir un índice para el campo. Si permite entradas **Null** en un campo indexado y espera que muchas de las entradas sean **Null**, puede establecer la propiedad **IgnoreNulls** para el objeto **Index** como **True,** con el fin de reducir el espacio de almacenamiento que utiliza el índice.

El valor de la propiedad **IgnoreNulls** y de la propiedad **Required** determinan juntas si un registro con un valor de índice **Null** tiene entrada de índice.

Si IgnoreNulls es	Y Required es	Entonces
True	**False**	Valor **Null** permitido; ninguna entrada de índice agregada.
False	**False**	Valor **Null** permitido; entrada de índice agregada.
True o **False**	**True**	Valor **Null** no permitido; ninguna entrada de índice agregada.

Index (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproIndexS"}

Establece o devuelve un valor que indica el nombre del objeto **Index** activo en un **Recordset** de tipo Table (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que evalúa el nombre de un objeto **Index** en la colección **Indexes** de un objeto **Tabledef** o objeto **TableDef** de un objeto **Recordset** de tipo Table.

Comentarios

Los registros de las tablas base no se almacenan en ningún orden particular. El valor de la propiedad **Index** cambia el orden de los registros devueltos de la base de datos, pero no afecta al orden en que se almacenan los registros.

El objeto **Index** especificado tiene que estar ya definido. Si establece la propiedad **Index** a un objeto **Index** que no existe o si la propiedad **Index** no está configurada cuando utiliza el método **Seek**, se genera un error interceptable.

Examine la colección **Indexes** de un objeto **TableDef** para determinar qué objetos **Index** están disponibles para los objetos **Recordset** de tipo Table creados desde ese **TableDef**.

Puede crear un nuevo índice para la tabla creando un nuevo objeto **Index**, configurando sus propiedades, agregándolo a la colección **Indexes** del **TableDef** subyacente y después reabriendo el **Recordset**.

Los registros devueltos de un **Recordset** de tipo Table se pueden ordenar sólo por los índices definidos para la **TableDef** subyacente. Para ordenar registros en algún otro orden, puede abrir un **Recordset** de tipo Dynaset o Snapshot utilizando una instrucción SQL con una cláusula ORDER BY.

Notas

- No tiene porqué crear índices para las tablas, pero en tablas grandes sin índices, el acceso a un registro específico o la creación de un **Recordset** puede llevar mucho tiempo. Por otra parte, tenga en cuenta que la creación de demasiados índices hace más lentas las operaciones de actualización, agregación y eliminación, dado que todos los índices de actualizan automáticamente.
- Los registros que se leen de tablas sin índices se devuelven sin un orden particular.
- La propiedad **Attributes** de cada objeto **Field** del objeto **Index** determina el orden de los registros y consecuentemente determina las técnicas de acceso a utilizar para ese índice.
- Un índice único ayuda a optimizar la búsqueda de registros.
- Los índices no afectan al orden físico de los registros de una tabla base. Los índices sólo afectan a la forma en que se accede a los registros mediante un **Recordset** de tipo Table, cuando se elige un índice particular o cuando se abre **Recordset**.

Inherit (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproInheritC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproInheritX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproInheritA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproInheritS"}

Establece o devuelve un valor que indica si los objetos **Document** nuevos heredarán un valor de propiedad **Permissions** predeterminado (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se obtienen son un tipo de datos **Boolean**. Si establece la propiedad a **True**, los objetos **Document** heredan un valor de propiedad **Permissions** predeterminado.

Comentarios

Utilice la propiedad **Inherit** de conjunto con la propiedad **Permissions** para definir qué autorizaciones se le darán a los nuevos documentos cuando se crean. Si establece la propiedad **Inherit** como **True** y luego establece una autorización para un contenedor, entonces siempre que se cree un nuevo documento en ese contenedor se establecerá esa autorización para el nuevo documento. Esta es una forma muy conveniente de configurar de forma predeterminada las autorizaciones de un objeto.

El valor de la propiedad **Inherit** no afectará a los documentos ya existentes en el contenedor. No es posible modificar todas las autorizaciones de todos los documentos existentes en un contenedor estableciendo la propiedad **Inherit** y un nuevo permiso. Esto tendrá efecto sólo sobre los nuevos documentos que se creen después de definir la propiedad **Inherit**.

Inherited (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproInheritedC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproInheritedX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproInheritedA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproInheritedS"}

Devuelve un valor que indica si un objeto **Property** se hereda de un objeto subyacente.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que es **True** si el objeto **Property** se hereda. Para objetos **Property** incorporados que representan propiedades definidas de forma predeterminada, el único valor de retorno posible es **False**. Esta propiedad siempre es **False** en un espacio de trabajo ODBCDirect.

Comentarios

Puede utilizar la propiedad **Inherited** para determinar si una propiedad definida por el usuario se creó para el objeto al que se aplica o si la propiedad se creó para otro objeto. Por ejemplo, suponga que crea una propiedad para un objeto **QueryDef** y después abre un objeto **Recordset** desde el **QueryDef**. Esta propiedad será parte de la colección **Properties**, y su propiedad **Inherited** se establecerá a **True** porque la propiedad se creó para el objeto **QueryDef**, no para el objeto **Recordset**.

IniPath (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproIniPathC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproIniPathX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproIniPathA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproIniPathS"}

Establece o devuelve información acerca de las claves del Registro de Windows que contienen valores para el motor de base de datos Microsoft Jet (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que señala a una parte de la clave de Registro suministrada por el usuario, la que contiene las configuraciones o los parámetros del motor de base de datos Microsoft Jet necesarios para instalar bases de datos ISAM.

Comentarios

El motor Microsoft Jet se puede configurar con el archivo de Registro. Puede utilizar el Registro para establecer opciones, como DLLs instalables.

Para que esta opción tenga efecto, la propiedad **IniPath** tiene que se establece antes de que cualquier otra función de acceso a los datos se invoque desde su aplicación. Si el motor Jet ya está inicializado, se generará un error interceptable. El alcance de este valor se limita a su aplicación y no se puede cambiar sin reiniciar.

También puede utilizar el Registro para proporcionar los parámetros de inicialización de algunos controladores de bases de datos ISAM instalables. Por ejemplo, para utilizar la versión 4.0 de Paradox, configure la propiedad **IniPath** con una parte del Registro que contenga los parámetros apropiados.

Esta propiedad reconoce tanto HKEY_LOCAL_MACHINE o HKEY_LOCAL_USER. Si no se especifica ninguna clave de ruta, la predeterminada es HKEY_LOCAL_USER.

Las versiones 2.5 y anteriores de Microsoft Jet mantienen archivos .ini de información de inicialización.

IsolateODBCTrans (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproIsolateODBCTransC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproIsolateODBCTransX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproIsolateODBCTransA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproIsolateODBCTransS"}

Establece o devuelve un valor que indica si se aíslan las transacciones múltiples que conciernen a la misma base de datos de ODBC conectada a Microsoft Jet (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** si desea aislar transacciones involucrando a la misma conexión ODBC (Open Database Connectivity). **False** (el valor predeterminado) permitirá múltiples transacciones involucrando a la misma ODBC.

Comentarios

En algunas situaciones, puede ser necesario tener múltiples transacciones simultáneas pendientes de la misma base de datos de ODBC. Para hacer esto, necesita abrir un **Workspace** distinto para cada transacción. Aunque cada **Workspace** puede tener su propia conexión ODBC a la base de datos, esto reduce el rendimiento del sistema. Puesto que el aislamiento de transacciones no se requiere normalmente, las conexiones ODBC de múltiples objetos **Workspace** abiertos por el mismo usuario se comparten de forma predeterminada.

Algunos servidores de ODBC, como el Microsoft SQL Server, no permiten transacciones simultáneas en una conexión sencilla. Si necesita tener más de una transacción a la vez pendiente de una base de datos, establezca la propiedad **IsolateODBCTrans** como **True** en cada **Workspace** tan pronto como la abra. Esto fuerza una conexión ODBC separada para cada **Workspace**.

LastModified (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproLastModifiedC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproLastModifiedX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproLastModifiedA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproLastModifiedS"}

Devuelve un marcador que indica el registro agregado o cambiado más recientemente.

Valores que se pueden obtener

El valor que se obtiene es una matriz de tipo **Variant** con datos de tipo Byte.

Comentarios

Se puede utilizar **LastModified** para mover el registro agregado o actualizado más recientemente. Utilice la propiedad **LastModified** con objetos **Recordset**. Es necesario modificar un registro en **Recordset** para que **LastModified** tenga un valor.

LockEdits (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproLockEditsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproLockEditsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproLockEditsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproLockEditsS"}

Establece o devuelve un valor que indica el bloqueo que está activado durante la edición.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es una expresión **Boolean** que indica el tipo de bloqueo, como se especifica en la siguiente tabla.

Valor	Descripción
True	Predeterminado. Está activado el bloqueo pesimista. La página de 2 KB que contiene el registro que se está modificando se bloquea tan pronto como se utiliza el método **Edit**.
False	Está activado el bloqueo optimista durante la edición. La página de 2 KB que contiene el registro se bloquea sólo mientras el registro se está actualizando con el método **Update**.

Comentarios

Puede utilizar la propiedad **LockEdits** con objetos **Recordset** que se pueden actualizar.

Si una página se bloquea, ningún otro usuario podrá modificar registros de la misma página. Si establece **LockEdits** como **True** y otro usuario ya tiene la página bloqueada, ocurrirá un error cuando utiliza el método **Edit**. Otros usuarios pueden leer datos de páginas bloqueadas.

Si establece **LockEdits** como **False** y más tarde utiliza **Update**, mientras la página está bloqueada por otro usuario, ocurrirá un error. Para ver los cambios realizados a su registro por otro usuario, utilice el método **Move** con el argumento 0; sin embargo, si hace esto, perderá sus cambios.

Cuando trabaja con orígenes de datos ODBC conectados a Microsoft Jet, la propiedad **LockEdits** se establece siempre a **False** o bloqueo optimista. El motor de base de datos Microsoft Jet no tiene control sobre los mecanismos de bloqueo utilizados en los servidores de bases de datos externas.

Nota Puede presentar el valor de **LockEdits** cuando abre por primera vez el objeto **Recordset** estableciendo el argumento *bloquearmodificaciones* del método **OpenRecordset**. Establecer el argumento *bloquearmodificaciones* a **dbPessimistic** establecerá la propiedad **LockEdits** a **True** y establecer *bloquearmodificaciones* a cualquier otro valor establecerá la propiedad **LockEdits** a **False**

LoginTimeout (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproLoginTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproLoginTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproLoginTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproLoginTimeoutS"}

Establece o devuelve el número de segundos que se esperará antes de que se genere un error cuando se intenta conectar a una base de datos de ODBC.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Integer** que representa el número de segundos que se esperará antes de que se produzca un error de tiempo de espera. La configuración predeterminada de **LoginTimeout** es de 20 segundos. Cuando **LoginTimeout** se establece como 0, nunca se generará un error de tiempo de espera.

Comentarios

Cuando se intenta conectar una base de datos de ODBC, como SQL Server, la conexión puede fallar debido errores de la red o porque el servidor no esté en funcionamiento. En lugar de esperar los 20 segundos predeterminados para establecer la conexión, puede especificar cuánto tiempo debe esperar antes de que se produzca el error, si la conexión no se establece. La conexión con el servidor ocurre implícitamente como parte de un número de diferentes eventos, como la ejecución de una consulta sobre una base de datos del servidor externo.

Puede utilizar **LoginTimeout** en el objeto **DBEngine** en espacios de trabajo Microsoft Jet y de ODBCDirect. Puede utilizar **LoginTimeout** en el **Workspace** sólo en espacios de trabajo ODBCDirect. Establecer la propiedad a -1 en un **Workspace** predeterminará el valor actual de **DBEngine.LoginTimeout**. Puede cambiar esta propiedad en un **Workspace** en cualquier momento y el nuevo valor surtirá efecto con el siguiente objeto **Connection** o **Database** abierto.

El valor predeterminado está determinado por el controlador ODBC. En un espacio de trabajo Microsoft Jet, puede suplantar el valor predeterminado del controlador creando una clave "ODBC" nueva la ruta de acceso del Registro **\HKEY_LOCAL_MACHINE\SOFTWARE\Jet\3.5**\, creando un parámetro **LoginTimeout** en esta clave y estableciendo el valor deseado.

LogMessages (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproLogMessagesC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproLogMessagesX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproLogMessagesA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproLogMessagesS"}

Establece o devuelve un valor que especifica si los mensajes devueltos de una base de datos de ODBC conectada a Microsoft Jet se almacenan en una tabla (sólo espacios de trabajo Microsoft Jet).

Nota Antes de que puede establecer usuario obtener el valor de la propiedad **LogMessages**, debe crear la propiedad **LogMessages** con el método **CreateProperty** y anexarlo a la colección **Properties** de un objeto **QueryDef**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** si los mensajes generados por ODBC se graban.

Comentarios

Algunas consultas de paso a través pueden devolver mensajes además de datos. Si establece la propiedad **LogMessages** como **True**, el motor de base de datos Microsoft Jet creará una tabla que almacenará los mensajes devueltos. El nombre de la tabla es el nombre del usuario concatenado con un guión (-) y un número secuencial que comienza en 00. Por ejemplo, en caso de que el nombre de usuario predeterminado sea ADMIN, las tablas devueltas se nombrarán ADMIN-00, ADMIN-01, y así sucesivamente.

Si desea que la consulta devuelva mensajes, cree una propiedad **LogMessages** personalizada para el **QueryDef** y establezca su tipo como Boolean y su valor como **True**.

Una vez que haya procesado el resultado de esas tablas, puede eliminarlas de la base de datos junto con la consulta temporal utilizada para crearlas.

Name (Propiedad)

Establece o devuelve un nombre definido por el usuario para un objeto de acceso a datos (DAO). Para un objeto no agregado a una colección, esta propiedad es de lectura/escritura.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que especifica un nombre. El nombre debe comenzar con una letra. El número máximo de caracteres depende del tipo del objeto **Name** al que se aplica, como se muestra en Comentarios. Puede incluir números y caracteres de subrayado (_) pero no puede incluir signos de puntuación espacios en blanco.

Comentarios

Los objetos **TableDef**, **QueryDef**, **Field**, **Index**, **User** y **Group** no pueden compartir el mismo nombre, ni tampoco pueden hacerlo los objetos de la misma colección.

La propiedad **Name** de un objeto **Recordset** abierto a través de una instrucción SQL comprende los primeros 256 caracteres de la instrucción SQL.

Puede utilizar la propiedad **Name** de un objeto con la instrucción **Dim** en el código, para crear otras instancias del objeto.

Nota Para muchos de los objetos DAO, la propiedad **Name** refleja el nombre con que se conoce la **Database**, como en el nombre de un **Tabledef**, **Field** o **QueryDef**. No hay vinculación directa entre el nombre del objeto **Database** y la variable de objeto utilizada para hacer referencia al mismo.

El uso de lectura/escritura de la propiedad **Name** depende del tipo de objeto al que se aplica y de si el objeto ha sido agregado a una colección. En un espacio de trabajo ODBC, la propiedad **Name** de un objeto anexado es siempre de sólo lectura. La siguiente tabla indica si la propiedad **Name** en un espacio de trabajo Microsoft Jet es de lectura/escritura o de sólo lectura para un objeto que está anexado a una colección y también indica su longitud máxima en los casos donde es de lectura/escritura.

Objeto	Uso	Longitud máxima
Container	Sólo lectura	
Connection	Sólo lectura	
Database	Sólo lectura	
Document	Sólo lectura	
Field		
Sin anexar	Lectura/ escritura	64
Anexado a **Index**	Sólo lectura	
Anexado a **QueryDef**	Sólo lectura	
Anexado a **Recordset**	Sólo lectura	
Anexado a **TableDef** (nativo)	Lectura/ escritura	64
Anexado a **TableDef** (vinculado)	Sólo lectura	
Anexado a **Relation**	Sólo lectura	
Group		

Sin añadir	Lectura/ escritura	20
Anexado	Sólo lectura	
Index		
Sin añadir	Lectura/ escritura	64
Anexado	Sólo lectura	
Parameter	Sólo lectura	
Property		
Sin añadir	Lectura/ escritura	64
Anexado	Sólo lectura	
Interno	Sólo lectura	
QueryDef		
Sin añadir	Lectura/ escritura	64
Temporal	Sólo lectura	
Anexado	Lectura/ escritura	64
Recordset	Sólo lectura	
Relation		
Sin añadir	Lectura/ escritura	64
Anexado	Sólo lectura	
TableDef	Lectura/ escritura	64
User		
Sin añadir	Lectura/ escritura	20
Anexado	Sólo lectura	
Workspace		
Sin añadir	Lectura/ escritura	20
Anexado	Sólo lectura	

NoMatch (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproNoMatchC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproNoMatchX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproNoMatchA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproNoMatchS"}

Devuelve un valor que indica si un registro particular se encontró utilizando el método **Seek** o uno de los métodos **Find** (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que es **True** si no se encontró el registro deseado. Cuando se abre o crea un objeto **Recordset**, su propiedad **NoMatch** se establece como **False**.

Comentarios

Para localizar un registro, utilice el método **Seek** sobre un objeto **Recordset** de tipo Table o uno de los métodos **Find** sobre un objeto **Recordset** de tipo Snapshot. Comprueba la configuración de la propiedad **NoMatch** para ver si se encontró el registro.

Si las operaciones **Seek** o **Find** no tienen éxito y **NoMatch** es **True**, el registro activo no será válido por más tiempo. Asegúrese de obtener el marcador del registro activo antes de utilizar el método **Seek** o un método **Find**, si cree que necesitará volver a ese registro.

Nota El uso de cualquiera de los métodos **Move** de un objeto **Recordset** no afecta a la configuración de su propiedad **NoMatch**.

Number (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproNumberX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproNumberA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproNumberS"}

Devuelve un valor numérico que especifica un error. **Number** es la propiedad predeterminada de un objeto **Error**.

Valores que se pueden obtener

El valor que se obtiene es un número entero de tipo **Long** que representa un número de error.

Comentarios

Utilice la propiedad **Number** para determinar el error que ocurrió. El valor de la propiedad corresponde a un número de captura único que corresponde a una condición de error. Si desea una lista completa de todos los números interceptables y condiciones de error, consulte Errores interceptables de acceso a los datos.

ODBCTimeout (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n ":"daproODBCTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproODBCTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproODBCTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproODBCTimeoutS"}

Indica el número de segundos de espera antes de que se produzca un error de tiempo excedido cuando un objeto **QueryDef** se ejecuta en una base de datos de ODBC.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Integer** que representa el número de segundos de espera antes de que se produzca un error de tiempo excedido.

Cuando la propiedad **ODBCTimeout** se establece a -1, el tiempo de espera predetermina el valor actual de la propiedad **QueryTimeout** del objeto **Connection** o **Database** que contiene el objeto **QueryDef**. Cuando la propiedad **ODBCTimeout** se establece a 0, no se produce error de tiempo excedido.

Comentarios

Cuando está utilizando una base de datos de ODBC, como SQL Server, los retrasos pueden ocurrir a causa del tráfico de la red o debido a usos intensivos del servidor de ODBC. Mejor que esperar indefinidamente, puede especifica cuánto tiempo deberá esperar el motor de base de datos Jet antes de producir el error.

El valor de la propiedad **ODBCTimeout** de un objeto **QueryDef** sobrescribe los valores especificados por la configuración de la propiedad **QueryTimeout** del objeto **Database** que contiene la **QueryDef**, pero sólo para esa **QueryDef**.

Nota En un espacio de trabajo ODBC, después de establecer **ODBCTimeout** a un valor específico puede devolverlo al predeterminado (p.e., -1) sólo durante la vida del objeto **QueryDef**. En caso contrario, se producirá un error.

OrdinalPosition (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproOrdinalPositionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproOrdinalPositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproOrdinalPositionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproOrdinalPositionS"}

Establece o devuelve la posición relativa de un objeto **Field** dentro de una colección **Fields** a la cual está agregado. Para un objeto no agregado aún a la colección **Fields** esta propiedad es de lectura/escritura.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Integer** que especifica el orden numérico de los campos. El valor predeterminado es 0.

Comentarios

El uso de la propiedad **OrdinalPosition** depende de si un objeto se ha agregado a la colección **Fields**, tal y como se muestra en la tabla siguiente.

Si la colección Fields pertenece a un	OrdinalPosition es
Index (Objeto)	No se admite
QueryDef (Objeto)	Sólo-lectura
Recordset (Objeto)	Sólo-lectura
Relation (Objeto)	No se admite
TableDef (Objeto)	Lectura/escritura

Generalmente, la posición ordinal de un objeto que se agrega a una colección depende el orden en el que se agrega el objeto. El primer objeto agregado estará en la primera posición (0), el segundo objeto agregado estará en la segunda posición (1) y así sucesivamente. El último objeto agregado está en la posición ordinal *cuenta* –1, donde *cuenta* es el número de objetos de la colección tal y como se especifica en la configuración de la propiedad **Count**.

Utilizando la propiedad **OrdinalPosition**, puede especificar una posición ordinal para los nuevos objetos **Field** que difiera del orden en el que se agregaron dichos objetos a la colección. Esto le permite especificar un orden de los campos a sus tablas, consultas y conjuntos de registros cuando los utilice en una aplicación. Por ejemplo, el orden en que se devuelve los campos en una consulta SELECT * se pudiera determinar mediante los valores actuales de **OrdinalPosition**.

Puede restablecer permanentemente el orden en que se devuelven los registros de los conjuntos de registros configurando la propiedad **OrdinalPosition** como un número entero cualquiera.

Dos o más objetos **Field** de la misma colección puede tener el mismo valor de **OrdinalPosition**, en cuyo caso se ordenarán alfabéticamente. Por ejemplo, si tiene un campo llamado Edad con un valor de 4 y establece un segundo campo llamado Peso también con 4, Peso se devolverá después de Edad.

Puede especificar un número que sea mayor que el número de campos – 1. El campo se devolverá en un orden relativo al número más grande. Por ejemplo, si establece la propiedad **OrdinalPosition** de un campo con 20 (y sólo hay 5 campos) y ha establecido la propiedad **OrdinalPosition** de otros dos campos como 10 y 30, respectivamente, el campo establecido a 20 se devolverá entre los campos establecidos como 10 y 30.

Nota Incluso si la colección **Fields** de un objeto **TableDef** no se actualizó, el orden del campo en un objeto **Recordset** abierto desde un objeto **TableDef** reflejará los datos de **OrdinalPosition** del objeto **TableDef**. Un objeto **Recordset** de tipo Table tendrá los mismos datos de **OrdinalPosition** que la tabla base, pero cualquier otro tipo de objeto **Recordset** tendrá datos **OrdinalPosition** nuevos (empezando con 0) que siguen el orden determinado por los datos **OrdinalPosition** del objeto **TableDef**.

Owner (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproOwnerC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproOwnerX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproOwnerA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproOwnerS"}

Establece o devuelve un valor que especifica el propietario del objeto (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que se asume bien el nombre de un objeto **User** de la colección **Users** o bien al nombre de un objeto **Group** de la colección **Groups**.

Comentarios

El propietario de un objeto tiene ciertos privilegios que se deniegan a otros usuarios. La configuración de la propiedad **Owner** se puede cambiar en cualquier momento mediante cualquier cuenta de usuario (representada por un objeto **User**) o grupo de cuentas de usuario (representado por un objeto **Group**) que tenga las autorizaciones apropiadas.

Password (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPasswordC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPasswordX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPasswordA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPasswordS"}

Establece la palabra clave de una cuenta de usuario (sólo espacios de trabajo Microsoft Jet).

Valores

El valor que se obtiene es un tipo de datos **String** que puede tener hasta 14 caracteres de longitud y puede incluir cualquier carácter excepto el carácter ASCII 0 (nulo). El valor de esta propiedad es de sólo escritura para los objetos nuevos todavía no anexados a una colección y no está disponible para objetos existentes.

Comentarios

Establezca la propiedad **Password** junto con la propiedad **PID** cuando cree un nuevo objeto **User**.

Utilice el método **NewPassword** para cambiar la configuración de la propiedad **Password** de un objeto **User** ya existente. Para eliminar una palabra clave, establezca el argumento *nuevaclave* del método **NewPassword** como una cadena de longitud cero ("").

En las palabras clave se distingue entre mayúsculas y minúsculas.

Nota Si no tiene permiso de acceso, no podrá cambiar la palabra clave de otro usuario.

PercentPosition (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPercentPositionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPercentPositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPercentPositionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPercentPositionS"}

Establece o devuelve un valor que indica o cambia la ubicación aproximada del registro activo en el objeto **Recordset**, basándose en un porcentaje de los registros del **Recordset**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un número de tipo **Single** entre 0,0 y 100,00.

Comentarios

Para indicar o cambiar la posición aproximada del registro activo en un **Recordset**, puede comprobar o establecer la propiedad **PercentPosition**. Cuando trabaje con un **Recordset** de tipo Dynaset o Snapshot abierto directamente desde una tabla base, rellene primero el objeto **Recordset** moviéndose al último registro antes de establecer o comprobar la propiedad **PercentPosition**. Si utiliza la propiedad **PercentPosition** antes de rellenar el **Recordset**, el movimiento será relativo al número de registros accedidos tal y como se indica en la configuración de la propiedad **RecordCount**. Se puede mover al último registro utilizando el método **MoveLast**.

Nota La utilización de la propiedad **PercentPosition** para mover el registro activo a un registro especificado en un **Recordset** no es recomendable—la propiedad **Bookmark** es mejor para esta tarea.

Una vez que establece un valor para la propiedad **PercentPosition**, el registro de la posición aproximada correspondiente a ese valor se convierte en el activo y la propiedad **PercentPosition** se restablece con un valor que refleja la posición aproximada del registro activo. Por ejemplo, si su **Recordset** contiene sólo cinco registros y establece su valor de **PercentPosition** en 77, el valor devuelto por la propiedad **PercentPosition** pudiera ser 80, no 77.

La propiedad **PercentPosition** se aplica a todos los tipos de objetos **Recordset** excepto a los desplazamientos hacia delante, o sobre un objeto **Recordset** abierto desde una consulta de paso a través desde una base de datos remota.

Puede utilizar la propiedad **PercentPosition** con una barra de desplazamiento sobre un formulario o cuadro de texto para indicar la ubicación del registro activo en un **Recordset**.

Permissions (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPermissionsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPermissionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPermissionsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPermissionsS"}

Establece o devuelve un valor que establece las autorizaciones del usuario o grupo de usuarios identificado por la propiedad **UserName** del objeto **Container** o **Document** (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es una constante de tipo **Long** que establece las autorizaciones. La siguiente tabla enumera los valores posibles para la propiedad **Permissions** de distintos objetos DAO. A menos que se indique lo contrario, todos los valores también son válidos para los objetos **Document**.

La siguiente tabla enumera los posibles valores de los objetos **Container** distintos de los contenedores Tables y Databases.

Constante	Descripción
dbSecNoAccess	Sin acceso al objeto (no es válido para objetos **Document**).
dbSecFullAccess	Acceso completo al objeto.
dbSecDelete	Puede eliminar el objeto.
dbSecReadSec	Puede leer la información relacionada con la seguridad del objeto.
dbSecWriteSec	Puede alterar los permisos de acceso.
dbSecWriteOwner	Puede cambiar la configuración de la propiedad **Owner**.

Para el contenedor Tables, los valores que se pueden establecer u obtener son:

Constante	Descripción
dbSecCreate	Puede crear nuevos documentos (válido sólo con un objeto **Document**).
dbSecReadDef	Puede leer la definición de tabla, incluyendo la información de columna e índice.
dbSecWriteDef	Puede modificar o eliminar la definición de tabla, incluyendo la información de columna e índice.
dbSecRetrieveData	Puede recuperar datos del **Document**.
dbSecInsertData	Puede agregar registros.
dbSecReplaceData	Puede modificar registros.
dbSecDeleteData	Puede eliminar registros.

Para el contenedor Databases, los valores que se pueden establecer u obtener son:

Constante	Descripción
dbSecDBAdmin	El usuario puede replicar una base de datos y cambiar la contraseña de la base de datos (no es válido para objetos **Document**).
dbSecDBCreate	El usuario puede crear bases de datos nuevas. Esta opción sólo es válida en el contenedor Databases del archivo de información del grupo de trabajo (Systen.mdw). Esta constante no es válida en objetos **Document**.

dbSecDBExclusive	Acceso exclusivo a la base de datos.
dbSecDBOpen	Puede abrir la base de datos.

Comentarios

Utilice esta propiedad para establecer o determinar el tipo de autorizaciones de lectura/escritura/modificación que tiene el usuario para un objeto **Container** o **Document**.

Un objeto **Document** hereda los permisos de los usuarios de su **Container**, asumiendo que la propiedad **Inherit** del **Container** está establecida para esos usuarios o para un grupo al cual pertenezcan los mismos. Configurando más tarde las propiedades **Permission** y **UserName** de un objeto **Document**, puede refinar mejor el control del acceso de su objeto.

Si desea establecer o devolver los permisos para un usuario que incluye los permisos heredados de cualquier grupo al que el usuario pertenece, utilice la propiedad **AllPermissions**.

PID (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPIDC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPIDX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPIDA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPIDS"}

Establece el identificador personal (PID) para un grupo o para una cuenta de usuario (sólo espacios de trabajo Microsoft Jet).

Valores

El valor es un tipo de datos **String** que contiene de 4 a 20 caracteres alfanuméricos. El valor de esta propiedad es de sólo lectura para los objetos nuevos todavía no anexados a una colección y no está disponible para objetos existentes.

Comentarios

Establezca la propiedad **PID** junto con la propiedad **Name** cuando cree un nuevo objeto **Group**. A su vez, establezca la propiedad **PID** junto con las propiedades **Name** y **Password** cuando cree un nuevo objeto **User**.

Primary (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPrimaryC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPrimaryX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPrimaryA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPrimaryS"}

Establece o devuelve un valor que indica si un objeto **Index** representa un índice principal para una tabla (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean**. **True** indica si el objeto **Index** representa un índice principal.

El valor de la propiedad **Primary** es de lectura/escritura para un objeto **Index** nuevo todavía no anexados a una colección y de sólo lectura para objeto **Index** existentes en una colección **Indexes**. Si el objeto **Index** está anexado al objeto **TableDef**, pero el objeto **TableDef** no está anexado a la colección **TableDefs**, la propiedad **Index** es de lectura/escritura.

Comentarios

Un índice principal contiene uno o más campos que identifican de forma única a todos los registros de la tabla en un orden predefinido. Puesto que el campo índice debe ser único, la propiedad **Unique** del objeto **Index** se establece como **True**. Si el índice principal contiene más de un campo, cada campo puede contener valores repetidos, pero cada combinación de valores de todos los campos indexados debe ser única. Un índice principal contiene una clave para la tabla y normalmente contiene los mismos campos que la clave principal.

Nota No tiene que crear obligatoriamente índices para las tablas, pero en tablas sin índices demasiado largas, el acceso a un registro específico puede llevar mucho tiempo. La propiedad **Attributes** de cada objeto **Field** en el **Index** determina el orden de los registros y consecuentemente determina las técnicas de acceso a utilizar para ese índice. Cuando cree una nueva tabla, es una buena idea crear un índice sobre uno o más campos que identifiquen cada registro y después establecer la propiedad **Primary** del objeto **Index** como **True**.

Cuando se establece una clave principal para una tabla, la clave principal se define automáticamente como el índice principal para la tabla.

QueryTimeout (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproQueryTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproQueryTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproQueryTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproQueryTimeoutS"}

Establece o devuelve un valor que especifica los segundos que espera antes de que ocurra un error de tiempo de espera cuando se ejecuta una consulta a un origen de datos ODBC.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Integer** que representa el número de segundos en espera. El valor predeterminado es 60.

Comentarios

Cuando utiliza una base de datos de ODBC, como SQL Server, puede haber retardos debidos al tráfico de la red o a un uso intensivo del servidor de ODBC. Mejor que esperar indefinidamente, puede especificar cuánto tiempo espera.

Cuando se utiliza **QueryTimeout** con un objeto **Connection** o **Database**, especifica un valor global para todas las consultas asociadas con la base de datos. Puede suplantar este valor para una consulta específica estableciendo la propiedad **ODBCTimeout** del objeto **QueryDef** concreto.

En un espacio de trabajo Microsoft Jet, puede suplantar el valor predeterminado creando una clave "ODBC" nueva en la ruta de acceso del Registro **\HKEY_LOCAL_MACHINE\SOFTWARE\Jet\3.5**, creando un parámetro **QueryTimeout** es esta clave y estableciendo el valor deseado.

RecordCount (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproRecordCountC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproRecordCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproRecordCountA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproRecordCountS"}

Devuelve el número de registros accedidos en un objeto **Recordset** o el número total de registros de un objeto **Recordset** o **TableDef**.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Long**.

Comentarios

Utilice la propiedad **RecordCount** para determinar a cuántos registros se ha accedido en un objeto **Recordset** o **TableDef**. La propiedad **RecordCount** no indica cuantos registros están contenidos en un objeto **Recordset** de tipo Dynaset, Snapshot o Forward-only hasta que se tenga acceso a todos los registros. Una vez que se haya accedido al último registro, la propiedad **RecordCount** indica el número total de registros no eliminados en el objeto **Recordset** o **TableDef**. Para forzar que se accede al último registro, utilice el método **MoveLast** en el objeto **Recordset**. También puede utilizar la función **Count** de SQL para determinar el número aproximado de registros que devolverá su consulta.

Nota El uso del método **MoveLast** para llenar un **Recordset** abierto de nuevo influye negativamente en el rendimiento. A menos que sea necesario tener un **RecordCount** exacto en cuanto abre el **Recordset**, es mejor esperar hasta que llene el **Recordset** con otros fragmentos de código antes de comprobar la propiedad **RecordCount**.

Si su aplicación elimina registros en un objeto **Recordset** de tipo Dynaset, el valor de la propiedad **RecordCount** disminuye. Sin embargo, los registros eliminados por otros usuarios no se reflejan en la propiedad **RecordCount** hasta que el registro activo se coloca en un registro eliminado. Si ejecuta una transacción que afecta al valor de la propiedad **RecordCount** y a continuación deshace los cambios en la transacción, la propiedad **RecordCount** no reflejará el número actual de registros restantes.

La propiedad **RecordCount** de un objeto **Recordset** de tipo Snapshot o Forward-only no está afectada por los cambios realizados en las tablas base.

Un objeto **Recordset** o **TableDef** sin registros tiene una valor de propiedad **RecordCount** de 0.

Cuando trabaja con objetos **TableDef** vinculados, el valor de la propiedad **RecordCount** es siempre –1.

La utilización del método **Requery** en un **Recordset** restablece la propiedad **RecordCount** exactamente como si la consulta se hubiera vuelto a ejecutar.

RecordsAffected (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproRecordsAffectedC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproRecordsAffectedX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproRecordsAffectedA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproRecordsAffectedS"}

Devuelve el número de registros afectados por el último método **Execute**.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Long** desde 0 hasta el número de registros afectados por el método **Execute** más recientemente llamado por un objeto **Database** o **QueryDef**.

Comentarios

Cuando utiliza el método **Execute** para ejecutar una consulta de acciones desde un objeto **QueryDef**, la propiedad **RecordsAffected** contendrá el número de registros eliminados, actualizados o insertados.

Cuando utiliza la propiedad **RecordsAffected** en un espacio de trabajo ODBC, no devolverá un valor útil de una consulta de acciones DROP TABLE de SQL.

Required (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproRequiredC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproRequiredX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproRequiredA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproRequiredS"}

Establece o devuelve un valor que indica si un objeto **Field** requiere un valor no **Null** o si todos los campos de un objeto **Index** deben rellenarse.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean**. **True** indica que no se permite un valor **Null**.

Para un objeto todavía no anexado a una colección, esta propiedad es de lectura/escritura. Para un objeto **Index**, el valor de esta propiedad es de sólo lectura sólo para los objetos anexados a las colecciones **Indexes** en unos objetos **Recordset** y **TableDef**.

Comentarios

La disponibilidad de la propiedad **Required** depende del objeto que contiene la colección **Fields**, como se muestra en la siguiente tabla.

Si la colección Fields pertenece a un	Required es
Index (Objeto)	No se admite
QueryDef (Objeto)	Sólo-lectura
Recordset (Objeto)	Sólo-lectura
Relation (Objeto)	No se admite
TableDef (Objeto)	Lectura/escritura

Para un objeto **Field**, puede utilizar la propiedad **Required** junto con la propiedad **AllowZeroLength**, **ValidateOnSet** o **ValidationRule** para determinar la validez de la configuración de la propiedad **Value** para ese objeto **Field**. Si **Required** se establece como **False**, el campo puede contener valores **Null**, así como también valores que cumplan con las condiciones especificadas en la configuración de las propiedades **AllowZeroLength** y **ValidationRule**.

Nota Cuando pueda establecer esta propiedad tanto para un objeto **Index** como para un objeto **Field**, establézcala para el objeto **Field**. La validez de la configuración de la propiedad para un objeto **Field** se comprueba antes que para un objeto **Index**.

Restartable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproRestartableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproRestartableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproRestartableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproRestartableS"}

Devuelve un valor que indica si un objeto **Recordset** soporta el método **Requery** el cual vuelve a ejecutar la consulta sobre la cual se basa el **Recordset**.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que es **True** si el objeto **Recordset** admite el método **Requery**. Los objetos **Recordset** de tipo Table siempre devuelven **False**.

Comentarios

Compruebe la propiedad **Restartable** antes de utilizar el método **Requery** en un **Recordset**. Si la propiedad **Restartable** del objeto está establecida como **False**, utilice el método **OpenRecordset** en la **QueryDef** subyacente para volver a ejecutar la consulta.

ReturnsRecords (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproReturnsRecordsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproReturnsRecordsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproReturnsRecordsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproReturnsRecordsS"}

Establece o devuelve un valor que indica si una consulta de paso a través de SQL a una base de datos externa devuelve registros.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** (predeterminado) si una consulta de paso a través devuelve registros.

Comentarios

No todas las consultas de paso a través de SQL a bases de datos externas devuelven registros. Por ejemplo, una instrucción SQL UPDATE actualiza registros sin devolver ninguno, mientras que una instrucción SQL SELECT devuelve registros. Si la consulta devuelve registros, se establece **ReturnsRecords** como **True**; si la consulta no devuelve registros, se establece **ReturnsRecords** a **False**.

Nota La propiedad **Connect** se debe establecer antes que la propiedad **ReturnsRecords**.

Size (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSizeS"}

Devuelve un valor que indica el tamaño máximo, en bytes, de un objeto **Field**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es una constante que indica el tamaño máximo de un objeto **Field**. Esta propiedad es de sólo lectura en objetos que todavía no se han añadido a la colección **Fields**. El valor depende de el valor de la propiedad **Type** del objeto **Field** como se explica en Comentarios.

Comentarios

En campos (distintos del tipo de campo Memo) que contengan caracteres, la propiedad **Size** indica el número máximo de caracteres que puede contener el campo. En campos numéricos, la propiedad **Size** indica cuantos bytes se necesitan para almacenarlo.

La utilización de la propiedad **Size** depende del objeto que contiene la colección **Fields** a la que se agrega el objeto **Field**, tal y como se muestra en la tabla siguiente.

Objeto agregado a	Utilización
Index	No se admite
QueryDef	Sólo lectura
Recordset	Sólo lectura
Relation	No se admite
TableDef	Sólo lectura

Cuando crea un objeto **Field** con un tipo de dato distinto de Text, el valor de la propiedad **Type** determina automáticamente el valor de la propiedad **Size** y no necesita configurar dicha propiedad. Para un objeto **Field** con un tipo de datos Text, **Size** se puede establecer con cualquier entero hasta el máximo de tamaño del texto (para bases de datos Microsoft Jet el máximo es 255.) Si no establece el tamaño, el campo será tan largo como permita la base de datos.

Para objetos **Field** de tipo Objeto OLE y Memo, **Size** se establece siempre como 0. Utilice el método **FieldSize** del objeto **Field** para determinar el tamaño del dato en un registro específico. El tamaño máximo de un campo de tipo Objeto OLE o Memo está limitado por los recursos del sistema o el tamaño máximo que permita la base de datos.

Sort (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSortC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSortX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSortA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSortS"}

Establece o devuelve el orden de los registros de un objeto **Recordset** (sólo espacio de trabajo Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que contiene la cláusula ORDER BY de una instrucción SQL, sin las palabras reservadas ORDER BY.

Comentarios

Puede utilizar la propiedad **Sort** con objetos **Recordset** de tipo Dynaset y Snapshot.

Cuando se establece esta propiedad para un objeto, la ordenación ocurre cuando se crea un **Recordset** subsiguiente a ese objeto. El valor de la propiedad **Sort** tiene preferencia sobre cualquier ordenación especificada para el objeto **QueryDef**.

La ordenación predeterminada es ascendente (A-Z o 0-100.)

La propiedad **Sort** no se aplica a objetos **Recordset** de tipo Table. Para ordenar un **Recordset** de tipo Table se utiliza la propiedad **Index**.

Nota En muchos casos, es más rápido abrir un nuevo **Recordset** utilizando una instrucción SQL que ya incluya el criterio de ordenación.

Source (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSourceA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSourceS"}

Devuelve el nombre del objeto o aplicación que originalmente generó el error.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **String** que representa el objeto o aplicación que generó el error.

Comentarios

El valor de la propiedad **Source** es normalmente el nombre de la clase del objeto o el Id de programa. Utilice **Source** para proporcionar a sus usuarios información acerca de porqué su código no es capaz de controlar un error generado en un objeto de otra aplicación.

Por ejemplo, si accede a Microsoft Excel y genera un error División por cero, Microsoft Excel establece **Error.Number** con su código de error correspondiente y establece **Source** como `Excel.Application`. Tenga en cuenta de que si el error se genera en otro objeto llamado por Microsoft Excel, será Excel la aplicación que intercepte el error y establecerá **Error.Number** de acuerdo a su propio código. Sin embargo, deja las demás propiedades del objeto **Error** (incluida **Source)** tal y como las estableció el objeto que generó el error. **Source** siempre contiene el nombre del objeto que generó originalmente el error.

Basándose en la información del error, puede escribir código para controlar de forma apropiada el error. Si su controlador de error falla, puede utilizar la información del objeto **Error** para describir el error al usuario, utilizando **Source** y las demás propiedades de **Error** para informar al usuario qué objeto causó originalmente el error, la descripción del error, etc.

Nota La construcción **On Error Resume Next** puede ser preferible a **On Error GoTo** cuando se encuentre con errores generados durante el acceso a otros objetos. La comprobación de **Error** tras cada interacción con un objeto elimina la ambigüedad acerca de a qué objeto se accedió cuando ocurrió el error. Así, puede estar seguro de qué objeto colocó el código de error en **Error.Number** y también qué objeto generó originalmente el error (el que se especifica en **Error.Source**.)

SourceField, SourceTable (Propiedades)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSourceFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSourceFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSourceFieldA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSourceFieldS"}

- **SourceField** - devuelve un valor que indica el nombre del campo que es la fuente original del dato de un objeto **Field**.
- **SourceTable** - devuelve un valor que indica el nombre de la tabla que es la fuente original del dato de un objeto **Field**.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **String** que especifica el nombre del campo o tabla que es el origen del dato.

Comentarios

Para un objeto **Field**, la utilización de las propiedades **SourceField** y **SourceTable** depende del objeto que contiene la colección **Fields** a la que está agregada el objeto **Field**, tal y como se muestra en la tabla siguiente:

Objeto agregado a	Utilización
Index	No se admite
QueryDef	Sólo lectura
Recordset	Sólo lectura
Relation	No se admite
TableDef	Sólo lectura

Estas propiedades indican los nombres del campo y tabla originales asociadas con el objeto **Field**. Por ejemplo, podría utilizar estas propiedades para determinar la fuente original de datos en un campo de consulta cuyo nombre no esté relacionado con el nombre del campo en la tabla subyacente.

Nota La propiedad **SourceTable** no devolverá un nombre de tabla significativo si utilizó un objeto **Field** de la colección **Fields** para un objeto **Recordset** de tipo Table.

SourceTableName (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSourceTableNameC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSourceTableNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSourceTableNameA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSourceTableNameS"}

Establece o devuelve un valor que especifica el nombre de una tabla vinculada o el nombre de la tabla base (sólo espacio de trabajo Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que especifica un nombre de tabla. Para una tabla base, el valor es una cadena de longitud cero (""). Esta propiedad es de sólo lectura para una tabla base y de lectura/escritura para tablas vinculadas o para objetos no agregados a la colección.

SQL (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSQLC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSQLX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSQLA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSQLS"}

Establece o devuelve la instrucción SQL que define la consulta ejecutada por un objeto **QueryDef**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que contiene una instrucción SQL.

Comentarios

La propiedad **SQL** contiene la instrucción SQL que determina cómo se seleccionan, agrupan y ordenan los registros cuando se ejecuta la consulta. Puede utilizar la consulta para seleccionar registros a incluir en un objeto **Recordset** de tipo Dynaset o Snapshot. También puede definir consultas para modificar datos sin devolver registros.

La sintaxis SQL utilizada en una consulta debe ser acorde con el dialecto SQL del motor de consulta, que está determinado por el tipo de espacio de trabajo. En un espacio de trabajo Microsoft Jet, utilice el lenguaje SQL de Microsoft Jet a no ser que utilice consultas SQL de paso a través, en cuyo caso debe utilizar el dialecto del servidor. En un espacio de trabajo ODBCDirect utilice el dialecto SQL del servidor.

Nota Puede enviar consultas DAO a una gran variedad de servidores de bases de datos diferentes con ODBCDirect y diferentes servidores reconocerán dialectos de SQL con pequeñas diferencias. Por lo tanto, no se proporciona Ayuda interactiva para Microsoft Jet SQL, aunque todavía se incluye Ayuda en pantalla para Microsoft Jet SQL mediante el menú Ayuda. Asegúrese de comprobar la documentación de referencia adecuada al dialecto SQL del servidor de base de datos cuando utilice conexiones ODBCDirect o consulta de paso a través en aplicaciones cliente/servidor conectadas a Microsoft Jet.

Si la instrucción SQL incluye parámetros para la consulta, debe configurarlos antes de ejecutar la consulta. Hasta que restaure los parámetros, se aplicarán los mismos valores a los parámetros cada vez que ejecute la consulta.

En un espacio de trabajo ODBCDirect, también puede usar la propiedad **SQL** para ejecutar instrucciones preparadas en el servidor. Por ejemplo, estableciendo la propiedad **SQL** con la siguiente cadena se ejecutará una instrucción ya preparada llamada "ObtenerDatos", con un parámetro en un servidor Microsoft SQL Server.

```
"{call ObtenerDatos (?)}"
```

En un espacio de trabajo Microsoft Jet, la mejor forma de realizar operaciones de consulta SQL de paso a través en orígenes de datos ODBC conectados a Microsoft Jet es utilizar el objeto **QueryDef**. Estableciendo la propiedad **Connect** del objeto **QueryDef** a un origen de datos ODBC, puede usar SQL en bases de datos que no sean Microsoft Jet en consultas que pasen al servidor externo. Por ejemplo, puede usar la instrucción TRANSACT SQL (con bases de datos Microsoft SQL Server o Sybase SQL Server) que el motor de base de datos Microsoft Jet no podría ejecutar de otra forma.

Nota Si establece la propiedad a una cadena de caracteres seguida de un valor no entero y los parámetros del sistema especifican un carácter decimal que no sea de EE.UU., como una coma (por ejemplo: `strSQL = "PRECIO > " & lngPrecio` y `lngPrecio = 125,50)`, se produce un error al intentar ejecutar el objeto **QueryDef** en una base de datos Microsoft Jet. Esto sucede porque durante la concatenación del carácter, el número que se va a convertir a una cadena usando el carácter decimal predeterminado por el sistema y Microsoft SQL sólo aceptan los caracteres decimales EE.UU.

Table (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproTableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproTableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproTableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproTableS"}

Indica el nombre de un objeto **Relation** de una tabla principal. Este nombre debe ser igual que la propiedad **Name** establecida en una **TableDef** u objeto **QueryDef** (sólo espacios de trabajo Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que define el nombre de la tabla de la colección **TableDefs** o la consulta de la colección **QueryDefs**. La propiedad **Table** es de lectura/escritura en objetos **Relation** nuevos que no se hayan agregado todavía a la colección y de sólo lectura en objetos **Relation** existentes en una colección **Relations**.

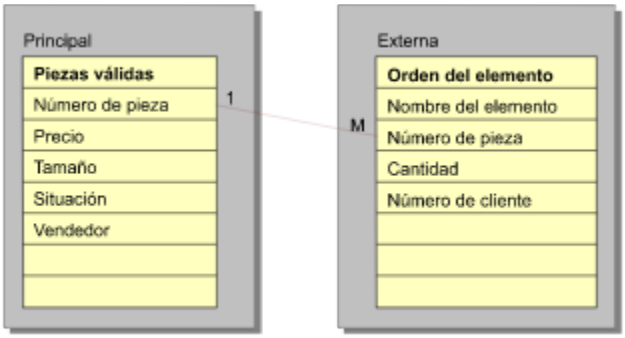
Comentarios

Utilice la propiedad **Table** con la propiedad **ForeignTable** para definir un objeto **Relation** que representa la relación entre campos de dos tablas o consultas. Establezca la propiedad **Table** con el valor de la propiedad **Name** del objeto **TableDef** o **QueryDef** primario y establezca la propiedad **ForeignTable** con el valor de la propiedad **Name** del objeto **TableDef** o **QueryDef** externo (al que se hace referencia.) La propiedad **Attributes** determina el tipo de relación entre los dos objetos.

Por ejemplo, si tiene una lista de códigos de piezas válidas (en un campo llamado NúmPieza) almacenada en una tabla PiezasVálidas, podría establecer una relación uno a varios con una tabla ElementosPedidos de forma que si un código de pieza se introduce en la tabla ElementosPedidos, tiene que existir también en la tabla PiezasVálidas. Si el código de pieza no existe en la tabla PiezasVálidas y no se ha establecido el valor **dbRelationDontEnforce** en la propiedad **Attributes** del objeto **Relation,** se generará un error interceptable.

En este caso, la tabla PiezasVálidas es la tabla principal, así la propiedad **Table** del objeto **Relation** se establecería como PiezasVálidas y la propiedad **ForeignTable** del objeto **Relation** se establecería como ElementosPedidos. La propiedad **Name** y la propiedad **ForeignName** del objeto **Field** en el objeto **Relation** de la colección **Fields** se configuraría como NúmPieza.

La siguiente figura muestra esta relación.



Transactions (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproTransactionsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproTransactionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproTransactionsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproTransactionsS"}

Devuelve un valor que indica si un objeto admite transacciones.

Valores que se pueden obtener

El valor que se obtiene es un tipo de datos **Boolean** que es **True** si el objeto admite transacciones.

Comentarios

En un espacio de trabajo ODBCDirect, la propiedad **Transactions** está activa en los objetos Connection y Database, e indica si se están utilizando transacciones o no.

En un espacio de trabajo de Microsoft Jet, puede también utilizar la propiedad **Transactions** como objetos **Recordset** de tipo Dinaset o Table. Los objetos **Recordset** de typo Snapshot y Forward-only siempre devuelven el valor **False**.

Si un **Recordset** de tipo Table o Dynaset se basa en una tabla del motor de base de datos Microsoft Jet, la propiedad **Transactions** es **True** y puede utilizar transacciones. Otros motores de bases de datos no pueden admitir transacciones. Por ejemplo, no se pueden utilizar transacciones en un **Recordset** de tipo Dynaset basado en una tabla de Paradox.

Compruebe la propiedad **Transactions** antes de utilizar el método **BeginTrans** sobre un objeto **Workspace** de un objeto **Recordset** para asegurarse de que se admiten las transacciones. La utilización de los métodos **BeginTrans**, **CommitTrans** o **Rollback** sobre un objeto no admitido no tiene ningún efecto.

Type (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproTypeS"}

Establece o devuelve un valor que indica el tipo operacional o el tipo de datos de un objeto.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es una constante que indica un tipo operacional o de datos. Para objetos **Field** o **Property** , esta propiedad es de lectura/escritura hasta que se agrega el objeto a la colección o a otro objeto, después será de sólo lectura. Para objetos **QueryDef**, **Recordset**, o **Workspace**, la propiedad es de sólo lectura. Para objetos **Parameter** en un espacio de trabajo de Microsoft Jet a propiedad es de sólo lectura, mientras en un espacio de trabajo ODBCDirect la propiedad es siempre de lectura/escritura.

Para un objeto **Field**, **Parameter** o **Property** los valores que se pueden establecer u obtener son:

Constante	Descripción
dbBigInt	Big Integer
dbBinary	Binary
dbBoolean	Boolean
dbByte	Byte
dbChar	Char
dbCurrency	Currency
dbDate	Date/Time
dbDecimal	Decimal
dbDouble	Double
dbFloat	Float
dbGUID	GUID
dbInteger	Integer
dbLong	Long
dbLongBinary	Long Binary (Objeto OLE)
dbMemo	Memo
dbNumeric	Numeric
dbSingle	Single
dbText	Text
dbTime	Time
dbTimeStamp	TimeStamp
dbVarBinary	VarBinary

Para un objeto **QueryDef**, los valores que se pueden establecer u obtener son:

Constante	Tipo de consulta
dbQAction	Acciones
dbQAppend	Datos agregados
dbQCompound	Compuesto
dbQCrosstab	Tablas de referencias cruzadas
dbQDDL	Definición de datos
dbQDelete	Eliminación
dbQMakeTable	Creación de tablas

dbQProcedure	Procedimiento (sólo espacios de trabajo ODBCDirect)
dbQSelect	Selección
dbQSetOperation	Unión
dbQSPTBulk	Se utiliza con **dbQSQLPassThrough** para especificar una consulta que no devuelva registros (sólo espacios de trabajo Microsoft Jet.)
dbQSQLPassThrough	Paso a través (sólo espacios de trabajo Microsoft Jet)
dbQUpdate	Actualización

Nota Para crear una consulta de paso a través de SQL en un espacio de trabajo Microsoft Jet, no necesita configurar explícitamente la propiedad **Type** a **dbQSQLPassThrough**. El motor de base de datos Microsoft Jet establece esto automáticamente al crear un objeto **QueryDef** y establece la propiedad **Connect**.

Para un objeto **Recordset**, los valores que se pueden establecer u obtener son:

Constante	Tipo de Recordset
dbOpenTable	Table (sólo espacios de trabajo Microsoft Jet)
dbOpenDynamic	Dynamic (sólo espacios de trabajo ODBCDirect)
dbOpenDynaset	Dynaset
dbOpenSnapshot	Snapshot
dbOpenForwardOnly	Forward-only

Para un objeto **Workspace**, los valores que se pueden establecer u obtener son:

Constante	Tipo de Workspace
dbUseJet	La **Workspace** es conectada al motor de base de datos de Microsoft Jet.
dbUseODBC	La **Workspace** es conectada a un origen de datos ODBC.

Comentarios

Cuando agrega un objeto **Field**, **Parameter** o **Property** nuevo a la colección de un objeto **Index**, **QueryDef**, **Recordset** o **TableDef**, se produce un error si la base de datos subyacente no admite el tipo de datos especificado para el nuevo objeto.

Unique (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproUniqueC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproUniqueX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproUniqueA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproUniqueS"}

Establece o devuelve un valor que indica si un objeto **Index** representa un índice único (clave) para una tabla (sólo espacios de trabajo Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** si el objeto **Index** representa a un índice único. Para un objeto **Index**, el valor de esta propiedad es de lectura/escritura hasta que el objeto se agregue a la colección, que será de sólo lectura.

Comentarios

Un índice único consta de uno o más campos que organizan lógicamente todos los registros de una tabla en un orden único predefinido. Si el índice consta de un campo, los valores de ese campo deben ser únicos en toda la tabla. Si el índice consta de más de un campo, cada campo puede contener valores duplicados, pero cada combinación de valores de todos los campos indexados debe ser única.

Si las dos propiedades **Unique** y **Primary** de un objeto **Index** se configuran como **True**, el índice es único y primario: Identifica de forma única a todos los registros de la tabla en un orden lógico predefinido. Si la propiedad **Primary** se configura como **False**, el índice es un índice secundario. Los índices secundarios distribuyen lógicamente los registros en un orden predefinido sin servir como identificadores de los registros de la tabla.

Notas

- No tiene porqué crear índices para las tablas, pero en tablas grandes no indexadas, el acceso a un registro específico puede llevar mucho tiempo.
- Los registros leídos de tablas sin índices se devuelve sin un orden particular.
- La propiedad **Attributes** de cada objeto **Field** en el objeto **Index** determina el orden de los registros y consecuentemente determina las técnicas de acceso a utilizar para ese **Index**.
- Un índice único ayuda a optimizar la búsqueda de registros.
- Los índices no afectan al orden físico de una tabla base, sino sólo a cómo se acceden los registros en el objeto **Recordset** de tipo Table cuando se elige un índice particular o cuando el motor de base de datos Microsoft Jet crea objetos **Recordset**.

Updatable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproUpdatableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproUpdatableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproUpdatableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproUpdatableS"}

Devuelve un valor que indica si se pueden hacer cambios a un objeto DAO

Valores que se pueden obtener

El valor que e obtiene es un tipo de datos **Boolean** que es **True** si el objeto se puede cambiar o actualizar. (Los objetos **Recordset** de tipo Snapshot y Forward-only devuelven siempre **False**.)

Comentarios

Dependiendo del objeto, si el valor de la propiedad **Updatable** es **True**, se cumple la instrucción asociada en la siguiente tabla:

Objeto	Indica que
Database	Se puede cambiar el objeto
QueryDef	Se puede cambiar la definición de la consulta
Recordset	Se puede actualizar el registro
TableDef	Se puede cambiar la definición de la tabla

El valor de la propiedad **Updatable** es siempre **True** para un objeto **TableDef** recién creado y **False** para un objeto adjunto **TableDef** vinculado. Un objeto **TableDef** nuevo puede ser agregado sólo a una base de datos para la cual el usuario actual tenga permiso de escritura.

Muchos tipos de objetos pueden contener campos que no pueden ser actualizados. Por ejemplo, puede crear un **Recordset** de tipo Dynaset en el cual sólo algunos campos se puedan cambiar. Esos campos se pueden fijar o contener datos que se incrementen automáticamente o el Dynaset puede resultar de una consulta que combine tablas que se pueden actualizar con tablas que no se pueden actualizar.

Si el objeto sólo contiene campos de sólo lectura, el valor de la propiedad **Updatable** es **False**. Cuando uno o más campos son que se pueden actualizar, el valor de la propiedad es **True**. Puede modificar sólo los campos que se pueden actualizar. Si intenta asignar un nuevo valor a un campo de sólo lectura se genera un error interceptable.

La propiedad **Updatable** de un objeto **QueryDef** se establece a **True** si la definición de consulta se puede actualizar, incluso si el **Recordset** resultante no se puede actualizar.

Dado que un objeto que se puede actualizar puede contener campos de sólo lectura, compruebe la propiedad **DataUpdatable** para cada campo de la colección **Fields** de un objeto **Recordset** antes de modificar un registro.

UserName (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproUserNameC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproUserNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproUserNameA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproUserNameS"}

Establece o devuelve un valor que representa un usuario o grupo de usuarios o el propietario de un objeto **Workspace**

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que define el nombre de un usuario. En un espacio de trabajo Microsoft Jet representa un objeto **User** de la colección **Users** o de un objeto **Group** en la colección **Groups**. Para objetos **Container** y **Document** de Microsoft Jet, esta propiedad es de lectura/escritura. Para todos los objetos **Workspace**, el valor de esta propiedad es de sólo lectura.

Comentarios

Dependiendo del tipo de objeto, la propiedad **UserName** representa

- El propietario de un objeto **Workspace**.
- Un usuario o grupo de usuarios cuando manipula las permisos de acceso de un objeto **Container** o de un objeto **Document** (sólo espacios de trabajo Microsoft Jet.)

Para encontrar o establecer permisos de un usuario particular o de un grupo de usuarios, establezca primero la propiedad **UserName** con el nombre de usuario o grupo que desea examinar. Después compruebe el valor de la propiedad **Permissions** para determinar qué permisos tiene el usuario o grupo de usuarios o establezca la propiedad **Permissions** para cambiar los permisos.

En el caso de un objeto **Workspace**, compruebe el valor de la propiedad **UserName** para determinar el propietario del objeto **Workspace**. Establezca la propiedad **UserName** para establecer el propietario del objeto **Workspace** antes de agregar el objeto a la colección **Workspaces**

V1xNullBehavior (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproV1xNullBehaviorC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproV1xNullBehaviorX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproV1xNullBehaviorA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproV1xNullBehaviorS"}

Indica si las cadenas de longitud cero ("") usadas en el código para rellenar campos Text o Memo se convierten a **Null**.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean**, que es **True** si las cadenas de longitud cero se convierten a **Null**.

Comentarios

Esta propiedad se aplica en bases de datos del motor de base de datos Microsoft Jet versión 1.x que se han convertido a bases de datos del motor de base de datos Microsoft Jet versiones 2.0 y 3.0.

Nota El motor de base de datos Microsoft Jet crea automáticamente esta propiedad cuando convierte una base de datos de una versión 1.x a una base de datos de las versiones 2.0 ó 3.0. Una base de datos de la versión 2.0 retiene esta propiedad cuando se convierte en una base de datos de la versión 3.0.

Si cambia el valor de esta propiedad, debe cerrar y después reabrir la bases de datos para que el cambio tenga efecto.

Para un rendimiento mayor, modifique el código que establezca cualquier campo Text o Memo como cadenas de longitud cero y en su lugar configúrelas como **Null** y elimine **V1xNullBehavior** de la colección **Properties**.

ValidateOnSet (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproValidateOnSetC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproValidateOnSetX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproValidateOnSetA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproValidateOnSetS"}

Establece o devuelve un valor que especifica si un valor de un objeto **Field** se valida inmediatamente o no cuando se establece la propiedad **Value** del objeto (sólo espacios de trabajo Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que puede tomar uno de estos valores:

Valor	Descripción
True	Se comprueba la regla de validación **Value** especificada por la propiedad **ValidationRule** establecida en el objeto **Field** cuando se establece la propiedad **Value** del objeto.
False	(predeterminado) Valida cuando actualiza el registro.

Sólo los objetos **Field** en objetos **Recordset** admiten la propiedad **ValidateOnSet** como de lectura/escritura.

Comentarios

Establecer la propiedad **ValidateOnSet** a **True** puede ser útil en situaciones en las que el usuario está introduciendo registros que incluyen datos Memo importantes. Esperar hasta la llamada **Update** para validar datos, puede dar como resultado un tiempo innecesario esperando a escribir los datos Memo en la base de datos y puede resultar que los datos sean inválidos de todas formas ya que se puede romper otra regla de validación en otro campo.

ValidationRule (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproValidationRuleC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproValidationRuleX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproValidationRuleA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproValidationRuleS"}

Establece o devuelve un valor que valida el dato de un campo cuando se cambia o se agrega a una tabla (sólo espacios de trabajo Microsoft Jet)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que describe una comparación en el formulario de una cláusula WHERE de SQL, sin la palabra reservada WHERE. Esta propiedad es de lectura/escritura en objetos que todavía no se han agregado a la colección **Fields**. Vea Comentarios para obtener mas información de las características de lectura/escritura de esta propiedad.

Comentarios

La propiedad **ValidationRule** determina si el dato es o no válido en el campo asociado. Si el dato no es legal, se genera un error de ejecución interceptable. El mensaje de error que devuelve es el texto de la propiedad **ValidationText**, si se especificó o el texto de la expresión especificada por **ValidationRule**.

Para un objeto **Field**, la utilización de la propiedad **ValidationRule** depende del objeto que contiene la colección **Fields** al que se ha agregado el objeto **Field**.

Objeto agregado a	Utilización
Index	No se admite
QueryDef	Sólo lectura
Recordset	Sólo lectura
Relation	No se admite
TableDef	Lectura/escritura

Para un objeto **Recordset**, la utilización de la propiedad **ValidationRule** es de sólo lectura. Para un objeto **TableDef**, la utilización de la propiedad **ValidationRule** depende del estado de la **TableDef**, tal y como muestra la tabla siguiente:

TableDef	Utilización
Tabla base	Lectura/escritura
Tabla vinculada	Sólo lectura

La validación está admitida sólo por bases de datos que utilizan el motor de base de datos Microsoft Jet.

La expresión de cadena especificada por la propiedad **ValidationRule** de un objeto **Field** se puede referir sólo a ese **Field**. La expresión no se puede referir a funciones definidas por el usuario, funciones de agrupamiento SQL o consultas. Para establecer la propiedad **ValidationRule** de un objeto **Field** cuando el valor de su propiedad **ValidateOnSet** es **True**, la expresión se debe analizar con éxito (con el nombre de campo como el operando implicado) y se establece como **True**. Si el valor de su propiedad **ValidateOnSet** es **False**, el valor de la propiedad **ValidationRule** se ignora.

La propiedad **ValidationRule** de un objeto **Recordset** o **TableDef** se puede referir a múltiples campos de ese objeto. Las restricciones anotadas anteriormente en este tema se aplican al objeto **Field**.

Para un objeto **Recordset** de tipo Table, **ValidationRule** toma el valor de la propiedad **ValidationRule** del objeto **TableDef** que se utiliza para crear el objeto **Recordset** de tipo Table.

Para un objeto **TableDef** basado en una tabla vinculada, la propiedad **ValidationRule** hereda el valor de la propiedad **ValidationRule** establecido de la tabla base subyacente. Si la tabla subyacente no admite una validación, el valor de esta propiedad es una cadena de longitud cero ("").

Nota Si establece la propiedad a una cadena de caracteres seguida de un valor no entero y los parámetros del sistema especifican un carácter decimal que no sea de EE.UU., como una coma (por ejemplo: `strRegla = "PRECIO > " & lngPrecio` y `lngPrecio = 125,50)`, se producirá un error cuando su código intente validar algún dato. Esto sucede porque durante la concatenación del carácter, el número que se va a convertir a una cadena usando el carácter decimal predeterminado por el sistema y Microsoft SQL sólo se aceptan los caracteres decimales EE.UU.

ValidationText (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproValidationTextC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproValidationTextX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproValidationTextA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproValidationTextS"}

Establece o devuelve un valor que especifica el texto del mensaje que su aplicación muestra, si el valor de un objeto **Field** no satisface la regla de validación especificada por el valor de la propiedad **ValidationRule** (sólo espacios de trabajo Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que especifica el texto que se muestra si el usuario intenta introducir un valor inválido. Para un objeto no agregado todavía a una colección, esta propiedad es de lectura/escritura. Para un objeto **Recordset**, el valor de esta propiedad es de sólo lectura. Para un objeto **TableDef**, el valor de esta propiedad es de sólo lectura para una tabla vinculada y de lectura/escritura para una tabla base.

Comentarios

Para un objeto **Field**, la utilización de la propiedad **ValidationText** depende del objeto que contenga la colección **Fields** a la que se ha agregado el objeto **Field**, tal y como muestra la tabla siguiente:

Objeto agregado a	Utilización
Index	No se admite
QueryDef	Sólo lectura
Recordset	Sólo lectura
Relation	No se admite
TableDef	Lectura/escritura

Value (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproValueC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproValueA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproValueS"}

Establece o devuelve el valor de un objeto.

Valores que se pueden establecer y obtener

El valor establecido o devuelto es una tipo de datos **Variant** que evalúa un valor apropiado para el tipo de datos, tal y como se especifica en la propiedad **Type** del objeto.

Comentarios

Generalmente, la propiedad **Value** se utiliza para recuperar y alterar datos en objetos **Recordset**.

La propiedad **Value** es la propiedad predeterminada de los objetos **Field**, **Parameter** y **Property**. Por lo tanto puede establecer o devolver valores de unos de estos objetos refiriéndose a ellos directamente en lugar de especificar la propiedad **Value**.

Si intenta establecer o devolver la propiedad **Value** en un contexto inapropiado (por ejemplo, la propiedad **Value** de un objeto **Field** en la colección **Fields** de un objeto **TableDef**) puede causar un error interceptable.

Notas

- En un espacio de trabajo ODBCDirect, no puede leer o establecer la propiedad **Value** de un campo **Recordset** mas de una vez sin actualizar el registro activo. Por ejemplo, para leer y establecer la propiedad **Value**, primero lea la propiedad, luego utilice el método **Move** para actualizar el registro activo y por último, escriba el nuevo valor.
- Cuando se leen valores decimales de una base de datos Microsoft SQL Server, debe darles formato usando la notación científica a través del espacio de trabajo Microsoft Jet, pero deben aparecer como valores decimales normales en espacios de trabajo ODBCDirect.

Version (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproVersionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproVersionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproVersionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproVersionS"}

- Espacio de trabajo de Microsoft Jet – En el objeto **DBEngine**, devuelve la versión actual de DAO en uso. En el objeto **Database**, devuelve la versión de Jet que creó el archivo.mdb.
- Espacio de trabajo ODBCDirect – En el objeto **DBEngine**, devuelve la versión actual de DAO en uso. En el objeto **Database**, devuelve la versión actual del controlador ODBC en uso.

Valores que se pueden obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que calcula el número de versión, con el siguiente formato:

- Espacio de trabajo Microsoft Jet representa el número de versión con el formato "*primario.secundario*". Por ejemplo, "3.0". El número de versión del producto consiste en el número primario (3), un punto y el número secundario.
- Espacio de trabajo ODBCDirect - representa el número de versión de DAO con el formato "*primario.secundario*" o representa el número de versión del controlador ODBC con el formato "*primario.secundario.revisión*". Por ejemplo, **DBEngine.Version** con un valor de "3.5" indica la versión 3.5 de DAO. Un objeto **Database** con un valor en **Version** de 2.50.1032 indica que la instancia actual de DAO está conectada con la versión 2.5 de ODBC, revisión 1032.

Comentarios

En un espacio de trabajo Microsoft Jet, la propiedad **Version** corresponde con la versión del motor de base de datos Microsoft Jet y no tiene que coincidir necesariamente con el número de versión del producto Microsoft con el que está incluido el motor de base de datos. Por ejemplo, la propiedad **Version** de un objeto de base de datos creado con Microsoft Visual Basic 3.0 será 1.1, no 3.0.

La siguiente tabla muestra qué versión del motor de base de datos se utiliza con las diversas versiones de los productos Microsoft que lo utilizan.

Motor Microsoft Jet	Microsoft Access	Visual Basic	Microsoft Excel	Visual C++
1.0 (1992)	1.0	-	-	-
1.1 (1993)	1.1	3.0	-	-
2.0 (1994)	2.0	-	-	-
2.5 (1995)	-	4.0 (16 bits)	-	-
3.0 (1995)	‘95 (7.0)	4.0 (32 bits)	‘95 (7.0)	4.0, 4.1, 4.2
3.5 (1996)	‘97 (8.0)	5.0	‘97 (8.0)	5.0

DesignMasterID (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproDesignMasterIDC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDesignMasterIDX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDesignMasterIDA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDesignMasterIDS"}

Establece o devuelve un valor de 16 bytes que únicamente identifica el Diseño principal en un conjunto de réplicas (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **GUID** que únicamente identifica el Diseño principal.

Comentarios

Debe establecer la propiedad **DesignMasterID** sólo si necesita mover el Diseño principal activo. Establecer esta propiedad hace una réplica específica en el conjunto de réplicas dl Diseño principal.

Precaución No cree nunca un segundo Diseño principal en un conjunto de réplicas. La existencia de un segundo Diseño principal puede provocar la pérdida de datos.

Bajo circunstancias extremas, por ejemplo, si el Diseño principal se ha borrado o está dañado, puede establecer esta propiedad en la réplica activa. Sin embargo, establecer esta propiedad en una réplica donde ya hay otro Diseño principal podría dividir su conjunto de réplicas en dos conjuntos irreconciliables e impedir cualquier sincronización posterior de datos.

Si decide hacer una réplica del nuevo Diseño principal para el conjunto, sincronice con todas las réplicas del conjunto antes de establecer la propiedad **DesignMasterID**. La réplica debe estar abierta en modo exclusivo para hacerla el Diseño principal.

Si hace una réplica de sólo lectura en el Diseño principal, la réplica destino es de lectura/escritura; el Diseño principal antiguo también es de lectura/escritura.

La propiedad **DesignMasterID** establecida es almacenada en la tabla del sistema MSysRepInfo.

KeepLocal (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproKeepLocalC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproKeepLocalX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproKeepLocalA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproKeepLocalS"}

Establece o devuelve un valor en una tabla, consulta, formulario, informe, macro o módulo que no quiera replicar cuando la base de datos sea replicada (sólo espacios de trabajo Microsoft Jet).

Nota Antes de obtener o configurar la propiedad **KeepLocal** en un objeto **TableDef**, o **QueryDef**, debe crearla utilizando el método **CreateProperty** y abriendo la colección **Properties** para el objeto.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Text**. Si establece esta propiedad como `"T"`, el objeto permanece local cuando se replica la base de datos. Después de que se han replicado no puede utilizar la propiedad **KeepLocal** en objetos.

Comentarios

Una vez establecida la propiedad **KeepLocal**, aparecerá en la colección **Properties** para el objeto **Document** representando el objeto host.

Antes de configurar la propiedad **KeepLocal**, debe comprobar el valor de la propiedad **Replicable**.

Después de replicar una base de datos, todos los objetos nuevos creados dentro del Diseño principal o en cualquier otra réplica en el conjunto de réplicas, son objetos locales. Los objetos locales permanecen en la réplica que se ha creado y no se copiarán en el conjunto de réplicas. Cada vez que crea una nueva réplica en el conjunto, la nueva réplica contiene todos los objetos replicables de la réplica origen, pero ninguno de los objetos locales de la réplica origen.

Si crea un nuevo objeto en una réplica y quiere cambiarlo de local a replicable para que todos los usuarios que lo utilicen puedan replicarlo, puede crear cada objeto o importarlo al Diseño principal. Asegúrese de eliminar el objeto local de cualquier réplica, en otro caso, encontrará un error de diseño. Una vez que el objeto forme parte del Diseño principal, establezca la propiedad **Replicable** de los objetos a **True**.

El objeto establecido con la propiedad **KeepLocal** puede tener heredada esa propiedad de otro objeto. Sin embargo, el valor establecido por el otro objeto no tiene efecto sobre el comportamiento del objeto que quiere conservar como local. Debe establecer explícitamente la propiedad para cada objeto.

Replicable (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproReplicableC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproReplicableX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproReplicableA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproReplicableS"}

Establece o devuelve un valor que determina si una base de datos o un objeto en una base de datos se puede replicar (sólo espacios de trabajo Microsoft Jet).

Nota Antes de obtener o establecer la propiedad **Replicable** en una **Database**, **TableDef** u objeto **QueryDef**, debe crearla utilizando el método **CreateProperty** y abrir la colección **Properties** para el objeto.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Text**.

En un objeto **Database**, establecer esta propiedad a `"T"` hace replicable la base de datos. Una vez establecida la propiedad a `"T"`, no puede cambiarla; establecer la propiedad a `"F"` (o cualquier otro valor distinto de `"T"`) produce un error.

En un objeto en una base de datos, establecer esta propiedad a `"T"` replica el objeto (y cambios posteriores del objeto) a todas las réplicas del conjunto de réplicas. También puede establecer esta propiedad en la hoja de propiedades del objeto en Microsoft Access.

Nota Microsoft Jet 3.5 también admite la propiedad **ReplicableBool** de **Boolean**. Esta funcionalidad es idéntica a la de la propiedad **Replicable**. Establecer esta propiedad a **True** hace replicable la base de datos.

Comentarios

Antes de establecer la propiedad **Replicable** en una base de datos, realice una copiar de seguridad de la base de datos. Si se produce un fallo al establecer la propiedad **Replicable**, deberá eliminar la base de datos parcialmente replicada, realizar una nueva copia a partir de la copia de seguridad e intentarlo de nuevo.

Cuando establece esta propiedad en un objeto **Database**, Microsoft Jet agrega campos, tablas y propiedades a objetos de la base de datos. Microsoft Jet utiliza estos campos, tablas y propiedades para sincronizar los objetos de la base de datos. Por ejemplo, todas las tablas existentes tienen tres nuevos campos agregados que ayudan a identificar los registros que han cambiado. La adición de estos campos y otros objetos incrementa el tamaño de su base de datos.

En formularios, informes, macros y módulos definidos por una aplicación principal (como Microsoft Access), establezca la propiedad en el objeto definido como principal a través de la interface principal. Una vez establecida, la propiedad **Replicable** aparecerá en la colección **Properties** para el objeto **Document** que representa el objeto principal.

Si ya estableció la propiedad **Replicable** en un objeto utilizando la casilla de verificación **Replicated** en la hoja de propiedades del objeto, no puede establecer la propiedad **Replicable** mediante código.

Cuando crea una nueva tabla, consulta, formulario, informe, macro o módulo en la réplica, el objeto es considerado como local y es almacenado sólo esa. Si quiere que usuarios de otras réplicas puedan utilizar el objeto, debe cambiar el objeto de local a replicable. Cree cada objeto o impórtelo al Diseño principal y establezca la propiedad **Replicable** a `"T"`.

El objeto que establece con la propiedad **Replicable** puede heredar la propiedad de otro objeto. Sin embargo, el valor establecido por el otro objeto no tiene efecto sobre el comportamiento del objeto que quiere replicar. Debe establecer explícitamente la propiedad para cada objeto.

ReplicaID (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproReplicaIDC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproReplicaIDX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproReplicaIDA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproReplicaIDS"}

Devuelve un valor de 16 bytes que únicamente identifica una réplica de una base de datos (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden obtener

El valor devuelto es un valor **GUID** que únicamente identifica la réplica o el Diseño principal.

Comentarios

El motor de base de datos Microsoft Jet genera automáticamente este valor cuando crea una nueva réplica.

La propiedad **ReplicaID** de cada réplica (y el Diseño principal) se almacena en la tabla del sistema MSysReplicas.

SystemDB (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproSystemDBC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproSystemDBX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproSystemDBA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproSystemDBS"}

Establece o devuelve la ruta de acceso de la posición activa del archivo del archivo de información del grupo de trabajo (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un valor de tipo **String** que describe la ruta de acceso completa.

Comentarios

El motor de base de datos Microsoft Jet le permite definir un grupo de trabajo y establecer permisos de acceso diferentes a cada objeto en la base de datos para cada usuario del grupo de trabajo. El grupo de trabajo se define en el archivo de información del grupo de trabajo, normalmente llamado "system.mda". Para permitir el acceso de usuarios a objetos asegurados de la base de datos, DAO debe tener la ubicación de este archivo de información del grupo de trabajo. Se puede informar a DAO de esta ubicación especificándola en el Registro de Windows o estableciendo la propiedad **SystemDB**. En la instalación, el valor predeterminado es simplemente "system.mda", sin ruta de acceso.

Para que esta opción tenga algún efecto, la propiedad **SystemDB** se debe establecer antes de que la aplicación inicialice el objeto **DBEngine** (por ejemplo, creando una instancia de cualquier otro objeto de acceso a datos). El alcance de este establecimiento está limitado por su aplicación y no se puede cambiar sin reiniciar su aplicación.

PartialReplica (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPartialReplicaC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPartialReplicaX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPartialReplicaA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPartialReplicaS"}

Establece o devuelve un valor en un objeto **Relation** que indica si se debería considerar la relación cuando se rellena una réplica parcial de una réplica completa. (sólo bases de datos Microsoft Jet.)

Valores que se pueden establecer y obtener

El valor que se puede establecer o devolver es un tipo de datos **Boolean** que es **True** cuando se obliga la relación durante la sincronización.

Comentarios

Esta propiedad le permite replicar datos desde la réplica completa a la réplica parcial basados en relaciones entre tablas. Puede usar la propiedad **PartialReplica** cuando al establecer la propiedad **ReplicaFilter**, ésta por sí sola no puede especificar adecuadamente qué datos se deberían replicar en la parcial. Por ejemplo, suponga que tiene una base de datos en la que la tabla Clientes tiene una relación de uno a varios con la tabla Pedidos, y quiere configurar una réplica parcial que sólo replique los pedidos de clientes en la región de Lara (en vez de todos los pedidos). No es posible establecer la propiedad **ReplicaFilter** en la tabla Pedidos a `Región = 'Lara'` porque el campo Región está en la tabla Clientes, no en la tabla Pedidos.

Para replicar todos los pedidos de la región de Lara, debe indicar que la relación entre las tablas Pedidos y Clientes estará activa durante la réplica. Cuando haya creado la réplica, los siguientes pasos la rellenarán con todos los pedidos de la región de Lara:

1. Establezca la propiedad **ReplicaFilter** en el objeto **TableDef** Clientes a `"Región = 'Lara'"`.
2. Establezca el valor de la propiedad **PartialReplica** a **True** en el objeto **Relation** que corresponda a la relación entre Pedidos y Clientes.
3. Llame al método **PopulatePartial**.

Precaución Cuando establece un filtro de réplica o una relación de réplica, tenga en cuenta que los registros en la réplica parcial que no satisfacen los criterios de restricción se eliminan de la réplica parcial, pero no de la réplica completa. Por ejemplo, suponga que establece la propiedad **ReplicaFilter** en la **TableDef** Clientes a `"Región = 'Lara'"` y luego vuelve a rellenar la base de datos. Esto insertará o actualizará todos los registros de clientes en Lara. Si restablece la propiedad **ReplicaFilter** a `"Región = 'Lera'"` y vuelve a rellenar la base de datos, se eliminarán todos los registros de la región de Lara en la réplica parcial y todos los registros de clientes en Lera se insertarán desde la réplica completa. No se eliminará ningún registro de la réplica completa.

Antes de establecer la propiedad **ReplicaFilter** o **PartialReplica**, es buena idea sincronizar la réplica parcial en la que está estableciendo estas propiedades con la réplica completa. Esto asegurará que los cambios pendientes en la réplica parcial se combinarán con la réplica completa antes de que se elimine ningún registro en la réplica parcial.

ReplicaFilter (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproReplicaFilterC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproReplicaFilterX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproReplicaFilterA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproReplicaFilterS"}

Establece o devuelve un valor en un objeto **TableDef** dentro de una réplica parcial, que indica qué subconjunto de registros de una réplica completa se replica en esa tabla. (sólo bases de datos Microsoft Jet.)

Valores que se pueden establecer y obtener

La configuración o el valor que devuelve es un tipo **String** o un tipo **Boolean** que indica qué subconjunto de registros se ha replicado, tal y como se especifica en la siguiente tabla:

Valor	Descripción
Una cadena	Un criterio que debe satisfacer un registro en la tabla de réplica parcial para que se replique desde la réplica completa
True	Replica todos los registros.
False	(Predeterminado) No replica ningún registros

Comentarios

Esta propiedad es similar a una cláusula WHERE de SQL (sin la palabra WHERE), pero dentro del criterio no se pueden especificar subconsultas, agregar funciones (como **Count**) o funciones definidas por el usuario.

Sólo puede sincronizar datos entre una réplica completa y una réplica parcial. No puede sincronizar datos entre dos réplicas parciales. Asimismo, con la réplica parcial puede establecer restricciones en los registros que se replicarán, pero no puede indicar qué campos se replicarán.

Normalmente, cuando quiere replicar un conjunto diferente de registros restablece un filtro de réplica. Por ejemplo, cuando un comercial se encarga temporalmente de la región de ventas de otro comercial, la aplicación de la base de datos puede replicar temporalmente los datos de ambas regiones y volver al filtro anterior. En este caso, la aplicación restablece la propiedad **ReplicaFilter** y luego vuelve a rellenar la réplica parcial.

Si su aplicación cambia filtros de réplica, debería seguir estos pasos:

1. Utilice el método **Synchronize** para sincronizar su réplica completa con la réplica parcial en la que se van a cambiar los filtros.
2. Utilice la propiedad **ReplicaFilter** para hacer los cambios deseados en el filtro de réplica.
3. Utilice el método **PopulatePartial** para eliminar todos los registros de la réplica parcial y transferir todos los registros de la réplica completa que cumplan con los nuevos criterios del filtro de réplica.

Para eliminar un filtro, establezca la propiedad **ReplicaFilter** a **False**. Si elimina todos los filtros y llama al método **PopulatePartial**, no aparecerá ningún registro en ninguna de las tablas de la réplica parcial.

Nota Si se cambia el filtro y se llama al método **Synchronize** sin haber llamado primero **PopulatePartial**, tendrá lugar un error interceptable.

BatchCollisionCount (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproBatchCollisionCountC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproBatchCollisionCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproBatchCollisionCountA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproBatchCollisionCountS"}

Devuelve el número de registros que no completaron la última actualización por lotes (sólo espacios de trabajo ODBCDirect).

Valor que se pueden obtener

El valor que se obtiene es un tipo de datos **Long** que indica el número de registros fallidos o 0 si todos los registros se actualizaron correctamente.

Comentarios

Esta propiedad indica cuántos registros se encontraron con colisiones o han fallado durante el último intento de actualización por lotes. El valor de esta propiedad se corresponde con el número de marcadores en la propiedad **BatchCollisions**.

Si establece en la matriz **BatchCollisions** la propiedad **Bookmark** del objeto **Recordset** a valores de marcador, puede moverse a cada registro que haya fallado al completar el método **Update** por lotes más reciente.

Después de que se corrijan los registros de colisión, se puede volver a llamar un método **Update** en modo por lotes. En este momento DAO intenta otra actualización por lotes y la propiedad **BatchCollisions** otra vez refleja el conjunto de registros que fallaron en el segundo intento. En el intento actual no se envían ninguno de los registros que tuvieron éxito en los intentos anteriores, porque ahora tienen una propiedad **RecordStatus** de **dbRecordUnmodified**. Este proceso puede continuar siempre que haya colisiones o hasta que usted abandone las actualizaciones y cierre el conjunto de resultados.

BatchCollisions (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproBatchCollisionsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproBatchCollisionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a ":"daproBatchCollisionsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproBatchCollisionsS"}

Devuelve una matriz de marcadores que indican las filas que generaron las colisiones en la última operación de actualización por lotes (sólo espacios de trabajo ODBCDirect).

Valor que se pueden obtener

El valor que se obtiene es una expresión de tipo variant que contiene una matriz de marcadores.

Comentarios

Esta propiedad contiene una matriz de marcadores de filas que se encontraron con una colisión durante el último intento de llamada a un método **Update** por lotes. La propiedad **BatchCollisionCount** indica el número de elementos en la matriz.

Si establece en la matriz **BatchCollisions** la propiedad **Bookmark** del objeto **Recordset** a valores de marcador, puede moverse a cada registro que haya fallado al completar el método **Update** de modo por lotes más reciente.

Después de que se corrijan los registros de colisión, se puede volver a llamar al método **Update** en modo por lotes. En este momento DAO intenta otra actualización por lotes y la propiedad **BatchCollisions** otra vez refleja el conjunto de registros que fallaron en el segundo intento. En el intento actual no se envían ninguno de los registros que tuvieron éxito en los intentos anteriores, porque ahora tienen una propiedad **RecordStatus** de **dbRecordUnmodified**. Este proceso puede continuar siempre que haya colisiones o hasta que usted abandone las actualizaciones y cierre el conjunto de resultados.

Esta matriz se vuelve a crear cada vez que ejecuta un método **Update** en modo por lotes.

BatchSize (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproBatchSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproBatchSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproBatchSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproBatchSizeS"}

Establece o devuelve el número de instrucciones devueltas al servidor en cada lote (sólo espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que indica el número de instrucciones incluidas en un lote que se enviaron en una actualización por lotes.sencilla. El valor predeterminado es 15.

Comentarios

La propiedad **BatchSize** determina el tamaño del lote que se utilizó cuando se enviaron las instrucciones al servidor en una actualización por lotes. El valor de la propiedad determina el número de instrucciones enviadas al servidor en un búfer de comandos. De manera predeterminada se envían al servidor 15 instrucciones en cada lote. Esta propiedad se puede cambiar en cualquier momento. Si un servidor de base de datos no admite lotes de instrucciones, puede establecer esta propiedad a 1 para que cada instrucción se envíe de forma separada.

CacheStart (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproCacheStartC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproCacheStartX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproCacheStartA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproCacheStartS"}

Establece o devuelve un valor que especifica el marcador del primer registro en un objeto **Recordset** de tipo Dynaset que contiene datos que se van a incluir en la memoria caché desde un origen de datos ODBC (sólo espacios de trabajo Microsoft Jet).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **String** que especifica un marcador.

Comentarios

La inclusión de datos en una memoria caché mejora el rendimiento de una aplicación que recupera datos de un servidor remoto a través de objetos **Recordset** de tipo de Dynaset. Una memoria caché es un espacio en la memoria local que mantiene los datos que se han recuperado más recientemente desde el servidor, en caso de que se pidan otra vez los datos mientras la aplicación se está ejecutando. Cuando se piden datos, el motor de base de datos Microsoft Jet busca en la memoria caché los datos solicitados, en vez de recuperarlos del servidor, lo que llevaría más tiempo. Sólo se pueden guardar en la memoria caché datos de un origen de datos ODBC.

Cualquier origen de datos ODBC conectado a Microsoft Jet, como una tabla vinculada, puede tener una memoria caché local. Para crear la memoria caché abra un objeto **Recordset** desde el origen de datos remoto, establezca las propiedades **CacheSize** y **CacheStart** y luego utilice el método **FillCache** o avance por los registros con los métodos Move.

La configuración de la propiedad **CacheStart** es el marcador del primer registro en el objeto **Recordset** que se va incluir en la memoria caché. Puede utilizar el marcador de cualquier registro para establecer adecuadamente la propiedad **CacheStart**. Haga que el registro desde el que quiere empezar la memoria caché sea el registro activo y establezca la propiedad **CacheStart** igual a la propiedad **Bookmark**.

El motor de base de datos Microsoft Jet solicita los registros dentro del alcance de la memoria caché a la memoria caché y solicita registros fuera del ámbito de la memoria caché al servidor.

Los registros recuperados de la memoria caché no reflejan los cambios hechos por otros usuarios al origen de los datos de manera concurrente.

Para forzar una actualización de todos los datos en la memoria caché, establezca la propiedad **CacheSize** del objeto **Recordset** a 0, establezca el tamaño de la memoria caché que solicitó en un principio y a continuación utilice el método **FillCache**.

Connection (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a ":"daproConnectionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproConnectionS"}

En un objeto **Database**, devuelve el objeto **Connection** que se corresponde con la base de datos (sólo espacios de trabajo ODBCDirect).

En un objeto **Recordset** devuelve el objeto **Connection** al que pertenece el **Recordset** (sólo espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es una variable de objeto que representa la conexión **Connection**. En un objeto **Database** la propiedad **Connection** es sólo de lectura, mientras que en un objeto **Recordset** la propiedad es de lectura/escritura.

Comentarios

Utilice la propiedad **Connection** en un objeto **Database** para obtener una referencia a un objeto **Connection** que se corresponda con la **Database**. En DAO, un objeto **Connection** y su objeto **Database** correspondiente son simplemente dos referencias diferentes de variables de objetos del mismo objeto. La propiedad **Database** de un objeto **Connection** y la propiedad **Connection** de un objeto **Database** facilitan el cambio de conexiones a un origen de datos ODBC a través del motor de base de datos Microsoft Jet para utilizar ODBCDirect.

Database (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDatabaseS"}

Devuelve el objeto **Database** que corresponde a esta conexión (sólo espacios de trabajo ODBCDirect).

Valores que se pueden obtener

El valor que se obtiene es una variable de objeto que representa un objeto **Database**.

Comentarios

En un objeto **Connection** , utilice la propiedad **Database** para obtener una referencia a un objeto **Database** que se corresponde con **Connection**. En DAO, un objeto **Connection** y su objeto **Database** correspondiente son simplemente dos referencias diferentes de variable de objetos de un mismo objeto. La propiedad **Database** de un objeto **Connection** y la propiedad **Connection** de un objeto **Database** facilitan el cambio de conexiones a un origen de datos ODBC a través del motor de base de datos Microsoft Jet para utilizar ODBCDirect.

DefaultCursorDriver (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproDefaultCursorTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDefaultCursorTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDefaultCursorTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDefaultCursorTypeS"}

Establece o devuelve el tipo de controlador de cursor que se utiliza en la conexión creada por los métodos **OpenConnection** u **OpenDatabase** (sólo espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que puede ser una de las siguientes constantes:

Constante	Descripción
dbUseDefaultCursor	(Predeterminada) Usa cursores de la parte del servidor, si el servidor los admite, en otro caso utiliza la Biblioteca de cursores de ODBC.
dbUseODBCCursor	Siempre utiliza la Biblioteca de cursores de ODBC. Esta opción proporciona un mejor rendimiento para conjuntos de resultados pequeños, pero se degrada rápidamente con conjuntos de resultados más grandes.
dbUseServerCursor	Utiliza siempre cursores de la parte del servidor. Para las operaciones más grandes esta opción proporciona un mejor rendimiento, pero puede causar más tráfico en la red.
dbUseClientBatchCursor	Utiliza siempre el cursor por lotes del cliente. Esta opción es necesaria para actualizaciones por lotes.
dbUseNoCursor	Abre todos los cursores (es decir, objetos **Recordset**) como un tipo sólo hacia delante sólo lectura y con un tamaño de conjunto de filas de 1. También conocido como "consultas sin cursor".

Comentarios

Este valor de propiedad sólo afecta a las conexiones establecidas después de haber establecido la propiedad. El cambio de la propiedad **DefaultCursorDriver** no tiene efecto en las conexiones existentes.

DefaultType (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproDefaultTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDefaultTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDefaultTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDefaultTypeS"}

Establece o devuelve un valor que indica qué tipo de espacio de trabajo (Microsoft Jet u ODBCDirect) utilizará el próximo objeto **Workspace** que se cree.

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que puede ser una de las siguientes constantes:

Constante	Descripción
dbUseJet	Crea objetos **Workspace** conectados al motor de base de datos Microsoft Jet.
dbUseODBC	Crea objetos **Workspace** conectados a un origen de datos ODBC.

Comentarios

Este valor se puede suplantar por un único **Workspace** estableciendo el argumento *tipo* al método **CreateWorkspace**.

Direction (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproDirectionC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproDirectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproDirectionA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproDirectionS"}

Establece y devuelve un valor que indica si un objeto **Parameter** representa un parámetro de entrada, un parámetro de salida o ambos o el valor que devuelve el procedimiento (sólo espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que puede ser una de las siguientes constantes:

Constante	Descripción
dbParamInput	(Predeterminada) Pasa información al procedimiento.
dbParamInputOutput	Pasa información desde y al procedimiento.
dbParamOutput	Devuelve información desde el procedimiento como en un parámetro de salida en SQL.
dbParamReturnValue	Pasa el valor que devuelto desde un procedimiento.

Comentarios

Utilice la propiedad **Direction** para determinar si el parámetro es un parámetro de entrada, un parámetro de salida o ambos o el valor devuelto desde el procedimiento. Algunos controladores de ODBC no proporcionan información a una instrucción SELECT o a un procedimiento de llamada sobre la dirección de los parámetros, en estos casos es necesario establecer la dirección antes de ejecutar la consulta.

Por ejemplo, el siguiente procedimiento devuelve un valor de un procedimiento almacenado llamado "obtener_empleados"

```
{? = call obtener_empleados}
```

Esta llamada produce un parámetro, el valor que devuelve. Necesita establecer la dirección de este parámetro en **dbParamOutput** o **dbParamReturnValue** antes de ejecutar la **QueryDef**.

Necesita establecer todas las direcciones de los parámetros, excepto **dbParamInput,** antes de tener acceso o de establecer los valores de los parámetros y antes de ejecutar la **QueryDef**.

Debería utilizar **dbParamReturnValue** para valores devueltos, pero en los casos en los que el servidor o el controlador no admitan esta opción, puede utilizar **dbParamOutput**.

Nota El controlador del Servidor de SQL 6.0 de Microsoft establece automáticamente la propiedad **Direction** para todos los parámetros de procedimiento. No todos los controladores de ODBC pueden determinar la dirección de una parámetro de consulta. En estos casos, es necesario establecer la dirección antes de ejecutar la consulta.

FieldSize (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproFieldSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproFieldSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproFieldSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproFieldSizeS"}

Devuelve el número de bytes utilizados en la base de datos (y no en la memoria) de un objeto **Field** de tipo Memo o Binary Long en la colección **Fields** de un objeto **Recordset**.

Valores que se obtienen

El valor que se obtiene es un tipo de datos **Long** que indica el número de caracteres (para un campo de tipo Memo) o el número de bytes (para un campo de tipo Binary Long).

Comentarios

Puede utilizar **FieldSize** con los métodos **AppendChunk** y **GetChunk** para manipular campos grandes.

Dado que el tamaño de un campo de tipo Memo o Binary Long puede sobrepasar los 64 KB, debería asignar el valor devuelto por **FieldSize** a una variable lo suficientemente grande como para almacenar una variable de tipo **Long**.

Para determinar el tamaño de un objeto **Field** y no de tipo Memo o Binary Long, utilice la propiedad **Size**.

Nota En un espacio de trabajo ODBCDirect, la propiedad **FieldSize** no está disponible en las siguientes situaciones:

- Si el servidor de bases de datos o el controlador de ODBC no admiten cursores del lado del servidor.
- Si está utilizando la biblioteca de cursores de ODBC (es decir, la propiedad **DefaultCursorDriver** está establecida a **dbUseODBC** o a **dbUseDefault** cuando el servidor no admite cursores de la parte del servidor).
- Si está utilizando una consulta sin cursores (es decir, la propiedad **DefaultCursorDriver** está establecida a **dbUseNoCursor**).

Por ejemplo, el Microsoft SQL Server versión 4.21 no admite cursores de la parte del servidor, por lo que la propiedad **FieldSize** no está disponible.

MaxRecords (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproMaxRecordsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproMaxRecordsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproMaxRecordsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalle":"daproMaxRecordsS"}

Establece o devuelve el número máximo de registros que se devuelven de una consulta..

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que representa el número de registros que se devolverán. El valor predeterminado es 0, que indica que no hay un límite en el número de registros que se devuelven.

Comentarios

Una vez que el número de filas que **MaxRecords** ha especificado, se han devuelto a su aplicación en un **Recordset**, el procesador de la consulta dejará de devolver registros adicionales, incluso si hubiera más registros que cumplieran los requisitos para estar incluidos en el **Recordset**. Esta propiedad es útil en situaciones donde los recursos limitados del cliente hacen que no sea posible la administración de un gran número de registros.

OriginalValue (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproOriginalValueC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproOriginalValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproOriginalValueA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproOriginalValueS"}

Devuelve el valor de un **Field** en la base de datos que existía cuando comenzó la última actualización por lotes (sólo espacios de trabajo ODBCDirect).

Valores que se pueden obtener

El valor que se obtiene es una expresión variant.

Comentarios

Durante una actualización por lotes optimista, puede tener lugar una colisión donde un segundo cliente modifica el mismo campo y registro en el tiempo entre el que el primer cliente recupera los datos y el primer intento de actualización por parte del primer cliente. La propiedad **OriginalValue** contiene el valor del campo en el momento en que comenzó el último método **Update** por lotes. Si este valor no coincide con el valor actualmente en la base de datos cuando la **Update** por lotes intenta escribir en la base de datos, tiene lugar una colisión. Cuando esto ocurre, el nuevo valor en la base de datos será accesible a través de la propiedad **VisibleValue**.

Prepare (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproPrepareC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproPrepareX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproPrepareA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproPrepareS"}

Establece o devuelve un valor que indica si la consulta debería estar preparada en el servidor como un procedimiento almacenado temporal, utilizando la función de API de ODBC **SQLPrepare** antes de la ejecución o simplemente ejecutada utilizando la función de API de ODBC **SQLExecDirect** (sólo espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que puede ser una de las siguientes constantes:

Constante	Descripción
dbQPrepare	(Predeterminada) La instrucción está preparada (es decir, se ha llamado a la API **SQLPrepare** de ODBC).
dbQUnprepare	La instrucción no está preparada (es decir, se ha llamado a la API **SQLExecDirect** de ODBC).

Comentarios

Puede utilizar la propiedad **Prepare** para que el servidor cree un procedimiento almacenado temporal desde su consulta y luego la ejecute o para que la consulta se ejecute directamente. De manera predeterminada, la propiedad **Prepare** está establecida a **dbQPrepare**. No obstante, puede establecer esta propiedad a **dbQUnprepare** para prohibir la preparación de la consulta. En este caso, la consulta se ejecuta utilizando la API **SQLExecDirect**.

Cuando se crea un procedimiento almacenado puede hacer que la operación inicial sea más lenta, pero mejora el rendimiento de todas las referencias subsecuentes a la consulta. No obstante, algunas consultas no se pueden ejecutar en forma de procedimientos almacenados. En estos casos, debe establecer la propiedad **Prepare** a **dbQUnprepare**.

Si **Prepare** está establecida a **dbQPrepare**, ésta se puede suplantar cuando la consulta se ejecute, estableciendo el argumento *opciones* del método **Execute** a **dbExecDirect**.

Nota Se llama a la API **SQLPrepare** de ODBC tan pronto como se establece la propiedad **SQL** de DAO. Por tanto, si quiere mejorar el rendimiento utilizando la opción **dbQUnprepare**, debe establecer la propiedad **Prepare** antes de establecer la propiedad **SQL**.

RecordStatus (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproRecordStatusC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproRecordStatusX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproRecordStatusA"} {ewc HLP95EN.DLL,DYNALINK,"Detalle":"daproRecordStatusS"}

Devuelve un valor que indica el estado de la actualización del registro actual, si éste es parte de una actualización por lotes (sólo espacios de trabajo ODBCDirect).

Valores que se pueden obtener

El valor que se puede obtener es un tipo de datos **Long** que puede ser cualquiera de las siguientes constantes:

Constante	Descripción
dbRecordUnmodified	(Predeterminada) El registro no se ha modificado o se ha actualizado con éxito.
dbRecordModified	El registro se ha modificado y no se actualizado en la base de datos.
dbRecordNew	El registro se ha insertado con un método **AddNew**, pero todavía no se ha insertado en la base datos.
dbDeleted	El registro se ha eliminado, pero todavía no se ha eliminado de la base de datos.
dbDBDeleted	El registro se ha eliminado localmente *y* de la base de datos..

Comentarios

El valor de la propiedad **RecordStatus** indica si el registro actual está implicado en la próxima actualización optimista por lotes y cómo lo estará.

Cuando un usuario cambia un registro, el **RecordStatus** para ese registro cambia automáticamente a **dbRecordModified**. De manera parecida, si se añade o elimina un registro **RecordStatus** refleja la constante apropiada. Cuando entonces utiliza un método **Update** en modo por lotes, DAO enviará para cada registro una operación apropiada al servidor remoto, basada en la propiedad **RecordStatus** del registro.

StillExecuting (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproStillExecutingC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproStillExecutingX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable":"daproStillExecutingA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproStillExecutingS"}

Indica si la operación asíncrona (eso es, un método llamado con la opción **dbRunAsync**) ha terminado o no de ejecutarse (sólo espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Boolean** que es **True** si la consulta está todavía ejecutando y **False** si la consulta se ha terminado.

Comentarios

Utilice la propiedad **StillExecuting** para determinar si el método asíncrono (es decir, un método ejecutado con la opción **dbRunAsync**) que se ha llamado más recientemente está completado. Mientras la propiedad **StillExecuting** sea **True**, no se puede tener acceso a ningún objeto devuelto.

La siguiente tabla muestra qué método se evalúa cuando utiliza **StillExecuting** en un tipo de objeto en particular.

Si StillExecuting se utiliza en	Se evalúa este método
Connection	Execute u OpenConnection
QueryDef	Execute
Recordset	MoveLast u OpenRecordset

Cuando la propiedad **StillExecuting** en un objeto **Connection** o **Recordset** devuelva **False**, se puede hacer referencia al objeto **Recordset** o **Connection** asociado. Cuando utiliza el método **NextRecordset** para completar el procesamiento de un **Recordset**, la propiedad **StillExecuting** se restablece a **True** mientras se recuperan los conjuntos de resultados subsecuentes.

Utilice el método **Cancel** para finalizar la ejecución o tarea en curso.

UpdateOptions (Propiedad)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproUpdateOptionsC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproUpdateOptionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproUpdateOptionsA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproUpdateOptionsS"}

Establece o devuelve un valor que indica cómo está construida la cláusula WHERE para cada registro durante una actualización por lotes y si la actualización por lotes debería utilizar una instrucción UPDATE o DELETE seguida de INSERT (sólo para espacios de trabajo ODBCDirect).

Valores que se pueden establecer y obtener

El valor que se puede establecer u obtener es un tipo de datos **Long** que puede ser cualquiera de las siguientes constantes:

Constante	Descripción
dbCriteriaKey	(Predeterminada) Utiliza sólo la(s) columna(s) clave en la cláusula WHERE.
dbCriteriaModValues	Utiliza la(s) columna(s) clave y todas las columnas actualizadas en la cláusula WHERE.
dbCriteriaAllCols	Utiliza la(s) columna(s) clave y todas las columnas en la cláusula WHERE.
dbCriteriaTimeStamp	Utiliza sólo la columna de fechas, si está disponible (si no hay una columna de fechas en el conjunto de resultados, generará un error de tiempo de ejecución).
dbCriteriaDeleteInsert	Utiliza un conjunto de instrucciones DELETE e INSERT para cada fila modificada.
dbCriteriaUpdate	(Predeterminada) Utiliza una instrucción UPDATE para cada fila modificada.

Comentarios

Cuando se ejecuta un método **Update** por lotes, DAO y la biblioteca de cursores por lotes del cliente crean una serie de instrucciones UPDATE de SQL que realizan los cambios necesarios. Para cada actualización se crea una cláusula WHERE de SQL para aislar los registros que están marcados como cambiados por la propiedad **RecordStatus** . Dado que algunos servidores remotos utilizan disparadores u otras formas de imponer integridad referencial, a veces es importante limitar los campos que se van a actualizar a solamente aquellos afectados por el cambio. Para hacer esto, establezca la propiedad **UpdateOptions** a una de las constantes **dbCriteriaKey**, **dbCriteriaModValues**, **dbCriteriaAllCols** o **dbCriteriaTimeStamp**. De este modo, sólo se ejecuta el mínimo absoluto de código de disparador. Como resultado, la operación de actualización se ejecuta más rápidamente y con un menor número de posibles errores.

También puede concatenar una de las constantes **dbCriteriaDeleteInsert** o **dbCriteriaUpdate** para determinar si utilizar un conjunto de instrucciones de SQL DELETE e INSERT o una instrucción UPDATE de SQL, para cada actualización cuando se envían de vuelta modificaciones en lotes al servidor. En el primer caso, se necesitan dos operaciones separadas para actualizar el registro. En algunos casos, especialmente cuando el sistema remoto implementa disparadores DELETE, INSERT y UPDATE, la elección de la configuración correcta de la propiedad **UpdateOptions** puede tener un impacto significante en el rendimiento.

Si no especifica ninguna constante, se utilizarán **dbCriteriaUpdate** y **dbCriteriaKey**.

Los registros agregados recientemente siempre generarán instrucciones INSERT y los registros eliminados siempre generarán instrucciones DELETE, por lo que esta propiedad sólo es aplicable a la manera en la que la biblioteca de cursores actualiza registros modificados.

VisibleValue (Propiedades)

{ewc HLP95EN.DLL,DYNALINK,"Vea tambiï¿½n":"daproVisibleValueC"} {ewc HLP95EN.DLL,DYNALINK,"Ejemplo":"daproVisibleValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Aplicable a":"daproVisibleValueA"} {ewc HLP95EN.DLL,DYNALINK,"Detalles":"daproVisibleValueS"}

Devuelve un valor en la base de datos que es más nuevo que la propiedad **OriginalValue** tal y como la determina un conflicto de actualización por lotes (sólo espacios de trabajo ODBCDirect).

Valores que se pueden obtener

El valor que se obtiene es una expresión variant.

Comentarios

Esta propiedad contiene el valor del campo que está actualmente en la base de datos en el servidor. Durante una actualización por lotes optimista, puede tener lugar una colisión donde un segundo cliente modificó el mismo campo y el mismo registro en el tiempo entre el que el primer cliente recuperó los datos y el primer intento de actualización del primer cliente. Cuando esto sucede, el valor que el segundo cliente establece será accesible a través de esta propiedad.

Objetos de acceso a datos (Constantes)

{ewc HLP95EN.dll, DYNALINK, "Vea tambiï¿½n":"damscDataAccessConstantsC "} {ewc HLP95EN.dll, DYNALINK, "Ejemplo":"damscDataAccessConstantsX ":1} {ewc HLP95EN.dll, DYNALINK, "Detalles":"damscDataAccessConstantsS "}

Los objetos de acceso a datos (DAO) proporcionan constantes incorporadas que puede utilizar con métodos o con propiedades. Estas constantes empiezan todas con las letras **db** y se documentan con el método o propiedad a la que aplican.

Leyenda:

Sólo lectura

Lectura/escritura

Constantes de la propiedad AllPermissions (Todas son)

Para cualquier objeto **Container** o **Document**:

Constante	Descripción
dbSecReadDef	Permite al usuario leer la definición de la tabla, incluyendo información de columna e índice.
DbSecWriteDef	Permite al usuario modificar o eliminar la definición de la tabla, incluyendo información de columna e índice.
DbSecRetrieveData	Permite al usuario recuperar datos del objeto **Documento**.
DbSecInsertData	Permite al usuario agregar registros.
DbSecReplaceData	Permite al usuario modificar registros.
dbSecDeleteData	Permite al usuario eliminar registros.

El contenedor **Databases** o cualquier objeto **Document** en una colección **Documents** puede incluir las siguientes:

Constante	Descripción
dbSecDeleteData	Permite al usuario eliminar registros.
dbSecDBAdmin	Permite al usuario hacer una réplica de la base de datos y cambiar la contraseña de la misma.
dbSecDBCreate	Permite al usuario crear nuevas bases de datos. Este valor sólo es válido en el contenedor Databases en el archivo de información del grupo de trabajo (System.mdw).
dbSecDBExclusive	Permite al usuario tener acceso exclusivo.
dbSecDBOpen	Permite al usuario abrir la base de datos.

Constantes de la propiedad Attributes

Para cualquier objeto **Field**, la propiedad **Attributes** puede incluir las siguientes:

Constante	Descripción

dbFixedField	Tamaño fijo de campo (predeterminado para campos Numéricos).
DbVariableField	Tamaño variable de campo (sólo para campos de Texto).
DbAutoIncrField	Nuevo valor de campo de registro aumentado a entero único **Long** (en un espacio de trabajo Microsoft Jet, disponible sólo para objetos **TableDef** abiertos desde archivos .mdb)
dbUpdatableField	El campo se puede actualizar
dbDescending	Campo clasificado en orden descendente (sólo espacios de trabajo Microsoft Jet)
dbHyperlinkField	El campo contiene información de vínculo (sólo campos de tipo Memo en espacios de trabajo Microsoft Jet)
dbSystemField	El campo es de réplica (sólo en un objeto **TableDef** en bases de datos Microsoft Jet)

Para cualquier objeto **Relation**, la propiedad **Attributes** puede incluir las siguientes:

Constante	Descripción
dbRelationUnique	Relación de uno a uno
dbRelationDontEnforce	Relación no impuesta (sin integridad referencial)
dbRelationInherited	Existe relación en la base de datos que contiene las dos tablas adjuntas
dbRelationUpdateCascade	Actualización en cascada
dbRelationDeleteCascade	Eliminación en cascada

Para cualquier objeto **TableDef**, la propiedad **Attributes** puede incluir las siguientes:

Constante	Descripción
dbAttachExclusive	Tabla de motor de la base de datos Microsoft Jet adjunta abierta en modo exclusivo.

dbAttachSavePWD	Guarda ID y contraseña del usuario para tabla remota adjunta.
DbSystemObject	Tabla del sistema
dbHiddenObject	Tabla oculta (para uso temporal).
DbAttachedTable	No hay tabla de base de datos ODBC adjunta.
DbAttachedODBC	Tabla de base de datos ODBC adjunta

Constantes de la propiedad CollatingOrder (Todas son)

Constante	Descripción
dbSortArabic	Secuencia de ordenación de arábico.
DbSortChineseSimplified	Secuencia de ordenación de chino simplificado.
DbSortChineseTraditional	Secuencia de ordenación de chino tradicional.
DbSortCyrillic	Secuencia de ordenación de ruso.
DbSortCzech	Secuencia de ordenación de checo.
DbSortDutch	Secuencia de ordenación de holandés.
DbSortGeneral	Secuencia de ordenación de inglés, alemán, francés y portugués.
DbSortGreek	Secuencia de ordenación de griego.
DbSortHebrew	Secuencia de ordenación de hebreo.
DbSortHungarian	Secuencia de ordenación de húngaro.
DbSortIcelandic	Secuencia de ordenación de islandés.
DbSortJapanese	Secuencia de ordenación de japonés.
DbSortKorean	Secuencia de ordenación de coreano.
DbSortNeutral	Secuencia de ordenación de neutral.
DbSortNorw	Secuencia de ordenación de noruego y danés.
DbSortPDXIntl	Secuencia de ordenación de internacional de Paradox.
DbSortPDXNor	Secuencia de ordenación de noruego y danés de Paradox.
DbSortPDXSwe	Secuencia de ordenación de sueco y finlandés de Paradox
dbSortPolish	Secuencia de ordenación de polaco

	de Paradox.
DbSortSlovenian	Secuencia de ordenación de esloveno.
DbSortSpanish	Secuencia de ordenación de español.
DbSortSwedFin	Secuencia de ordenación de sueco y finlandés.
DbSortThai	Secuencia de ordenación de tailandés.
DbSortTurkish	Secuencia de ordenación de turco.
DbSortUndefined	Secuencia de ordenación no definida o desconocido.

Propiedad DefaultCursorDriver (Todas son)

Constante	Descripción
dbUseDefaultCursor	(Predeterminada) Utiliza cursores del lado servidor si el servidor los admite; si no utiliza la Biblioteca de cursores ODBC.
DbUseODBCCursor	Siempre utiliza la Biblioteca de cursores ODBC. esta opción proporciona mejor rendimiento para conjuntos de resultado pequeños, pero se degrada rápidamente conjuntos de resultado grandes.
DbUseServerCursor	Siempre utiliza cursores del lado servidor. Para la mayoría de operaciones grandes proporciona mejor rendimiento, pero puede producir más tráfico en la red.
DbUseClientBatchCursor	Siempre utiliza la Biblioteca de cursores FoxPro. Se necesita esta opción para ejecutar actualizaciones por lotes.
DbUseNoCursor	Abre todos los cursores (esto es, objetos **Recordset**) como de tipo Forward-only, sólo lectura con un tamaño de conjunto de fila de 1. También se conoce como "consultas sin cursor."

Constantes de la propiedad Direction (Todas son)

Constante	Descripción
dbParamInput	(Predeterminada) Transfiere información al procedimiento.
DbParamInputOutput	Transfiere información al y desde el procedimiento.
DbParamOutput	Devuelve información desde el procedimiento como en un

	parámetro de salida en SQL.
DbParamReturnValue	Transfiere el valor devuelto por el procedimiento.

Constantes de la propiedad EditMode (Todas son)

Constante	Descripción
dbEditNone	No está en efecto ninguna operación de edición.
dbEditInProgress	Se solicitó el método **Edit**.
dbEditAdd	Se solicitó el método **AddNew**.

Constantes de la propiedad Permissions (Todas son)

Para cualquier objeto **Container**, la propiedad **Permissions** puede incluir las siguientes:

Constante	Descripción
dbSecNoAccess	Impide al usuario tener acceso al objeto.
dbSecFullAccess	Permite al usuario tener acceso total al objeto.
dbSecDelete	Permite al usuario eliminar el objeto.
dbSecReadSec	Permite al usuario leer información relacionada con la seguridad del objeto.
dbSecWriteSec	Permite al usuario alterar permisos de acceso.
dbSecWriteOwner	Permite al usuario cambiar el valor de la propiedad **Owner**.

Para cualquier base de datos **Container**, la propiedad **Permissions** puede incluir cualquier de las siguientes (Todas son):

Constante	Descripción
dbSecDBAdmin	Otorga permiso al usuario para hacer una replica de la base de datos y cambiar la contraseña de la misma.
dbSecDBCreate	Permite al usuario crear nuevas bases de datos (sólo válido en el objeto **Container** de la base de datos del sistema).
dbSecDBOpen	Permite al usuario abrir la base de datos.
dbSecDBExclusive	Permite al usuario tener acceso en modo exclusivo.

Para cualquier tabla **Container**, la propiedad **Permissions** puede incluir las siguientes (Todas son):

Constante	Descripción

dbSecCreate	Permite al usuario crear nuevas tablas (sólo válido con un objeto **Container** que representa una tabla o con el objeto **Container** de las bases de datos del sistema).
dbSecReadDef	Permite al usuario leer la definición de la tabla, incluyendo información de columna e índice.
dbSecWriteDef	Permite al usuario modificar o eliminar la definición de la tabla, incluyendo información de columna e índice.
dbSecRetrieveData	Permite al usuario recuperar datos del documento.
dbSecInsertData	Permite al usuario agregar registros.
dbSecReplaceData	Permite al usuario modificar registros.
dbSecDeleteData	Permite al usuario eliminar registros.

Para cualquier objeto **Document**, la propiedad **Permissions** puede incluir cualquiera de las siguientes (Todas son):

Constante	Descripción
dbSecCreate	Permite al usuario crear nuevas tablas (sólo válido con un objeto **Container** que representa una tabla).
dbSecDBCreate	Permite al usuario crear nuevas bases de datos (sólo válido en el objeto **Container** de las bases de datos del sistema).
dbSecDBOpen	Permite al usuario abrir la base de datos.
dbSecDBExclusive	Permite al usuario tener acceso en modo exclusivo.
dbSecDelete	Permite al usuario eliminar el objeto.
dbSecDeleteData	Permite al usuario eliminar registros.
dbSecFullAccess	Permite al usuario tener acceso total al objeto.
dbSecInsertData	Permite al usuario agregar registros.
dbSecReadDef	Permite al usuario leer la definición de tabla, incluyendo información de columna e índice.
dbSecReadSec	Permite al usuario leer información relacionada con la seguridad del objeto.
dbSecReplaceData	Permite al usuario modificar registros.
dbSecRetrieveData	Permite al usuario recuperar datos del documento.
dbSecWriteDef	Permite al usuario modificar o

	eliminar la definición de la tabla, incluyendo información de columna e índice.
dbSecWriteSec	Permite al usuario modificar los permisos de acceso.
dbSecWriteOwner	Permite al usuario cambiar el valor de la propiedad **Owner**.

Constantes de la propiedad Prepare (Todas son)

Constante	Descripción
dbQPrepare	(Predeterminado) Se prepara la instrucción (esto es, se llama a la API de ODBC **SQLPrepare**).
dbQUnprepare	No se prepara la instrucción (esto es, se llama a la API de ODBC **SQLExecDirect**).

Constantes de la propiedad RecordStatus (Todas son)

Constante	Descripción
dbDBDeleted	El registro se ha eliminado localmente y el la base de datos.
dbDeleted	El registro se ha eliminado, pero permanece todavía en la base de datos.
dbRecordModified	El registro se ha modificado y no se actualizó en la base de datos.
dbRecordNew	El registro se ha insertado con el método **AddNew**, pero todavía no se ha insertado en la base de datos.
dbRecordUnmodified	(Predeterminado) El registro no se ha modificado o se ha actualizado correctamente.

Constantes de la propiedad Type

Para cualquier objeto **Field**, **Parameter** o **Property**, la propiedad **Type** puede incluir cualquiera de la siguientes (Todas son):

Constante	Descripción
dbBigInt	Datos Big Integer (sólo ODBCDirect)
dbBinary	Datos Binary
dbBoolean	Datos Boolean (Verdadero/Falso)
dbByte	Datos Byte (8 bits)
dbChar	Datos Character (sólo ODBCDirect)
dbCurrency	Datos Currency
dbDate	Datos con valor Date
dbDecimal	Datos Decimal (sólo ODBCDirect)
dbDouble	Datos Double-precision floating-point

dbFloat	Datos Floating-point (sólo ODBCDirect)
dbGUID	Datos GUID
dbInteger	Datos Integer
dbLong	Datos Long Integer
dbLongBinary	Datos Binary (mapa de bits)
dbMemo	Datos Memo (texto extendido)
dbNumeric	Datos Numeric (sólo ODBCDirect)
dbSingle	Datos Single-precision floating-point data
dbText	Datos Text (tamaño variable)
dbTime	Datos en formato de hora (sólo ODBCDirect)
dbTimeStamp	Datos en formato de fecha y hora(sólo ODBCDirect)
dbVarBinary	Datos Variable Binary (sólo ODBCDirect)

Para cualquier objeto **QueryDef**, la propiedad **Type** puede incluir cualquiera de las siguientes (Todas son):

Constante	Descripción
dbQAction	Consulta de acciones
dbQAppend	Consulta de datos añadidos
dbQCompound	Consulta compuesta (sólo espacios de trabajo ODBCDirect)
dbQCrosstab	Consulta de tabla de referencias cruzadas
dbQDDL	Consulta de lenguaje de definición de datos (DDL)
dbQDelete	Consulta de eliminación
dbQMakeTable	Consulta de tabla
dbQProcedure	Procedimiento SQL que ejecuta un procedimiento almacenado (sólo espacios de trabajo ODBCDirect)
dbQSelect	Consulta de selección
dbQSetOperation	Consulta de operación establecer.
dbQSPTBulk	Consulta de operación de gran volumen.
dbQSQLPassThrough	Consulta de paso a través de SQL
dbQUpdate	Consulta de actualización

Para cualquier objeto **Recordset**, la propiedad **Type** puede incluir cualquiera de las siguientes (Todas son):

Constante	Descripción
dbOpenDynamic	Abre un objeto **Recordset** de tipo Dynamic (sólo espacios de trabajo ODBCDirect)
dbOpenDynaset	Abre un objeto **Recordset** de tipo Dynaset
dbOpenForwardOnly	Abre un objeto **Recordset** de tipo Forward-only
dbOpenSnapshot	Abre un objeto **Recordset** de tipo

	Snapshot
dbOpenTable	Abre un objeto **Recordset** de tipo Table (sólo espacios de trabajo Microsoft Jet)

Constantes de la propiedad UpdateOptions (Todas son)

Constante	Descripción
dbCriteriaKey	(Predeterminado) Utiliza sólo la columna o columnas de clave en la cláusula **Where**.
dbCriteriaModValues	Utiliza la columna o columnas de clave y todas las columnas actualizadas en la cláusula **Where**.
dbCriteriaAllCols	Utiliza la columna o columnas de clave y todas las columnas en la cláusula **Where**.
dbCriteriaTimeStamp	Utiliza sólo la columna de fecha si está disponible (generará un error en tiempo de ejecución si no hay columna de fecha en el conjunto de resultado).
dbCriteriaDeleteInsert	Utiliza la pareja de instrucciones **DELETE** y **INSERT** para cada fila modificada.
dbCriteriaUpdate	(Predeterminado) Utiliza la instrucción **UPDATE** para cada fila modificada.

Constantes de los argumentos configuración regional de los métodos CompactDatabase, CreateDatabase (Todas son)

Constante	Descripción
dbLangGeneral	Inglés, alemán, francés, portugués, italiano y español moderno
dbLangArabic	Arábico
dbLangChineseSimplified	Chino simplificado
dbLangChineseTraditional	Chino tradicional
dbLangCyrillic	Ruso
dbLangCzech	Checo
dbLangDutch	Holandés
dbLangGreek	Griego
dbLangHebrew	Hebreo
dbLangHungarian	Húngaro
dbLangIcelandic	Islandés
dbLangJapanese	Japón
dbLangKorean	Coreano
dbLangNordic	Nórdico
dbLangNorwdan	Noruego y danés

dbLangPolish	Polaco
dbLangSlovenian	Esloveno
dbLangSpanish	Español
dbLangSwedfin	Sueco y finlandés
dbLangThai	Tailandés
dbLangTurkish	Turco

Constantes del argumento opciones del método CompactDatabase (Todas son)

Constante	Descripción
dbDecrypt	Descodifica la base de datos mientras se compacta.
dbEncrypt	Codifica la base de datos
dbVersion10	motor de base de datos Microsoft Jet versión 1.0
dbVersion11	Motor de base de datos Microsoft Jet, versión 1.1.
dbVersion20	Motor de base de datos Microsoft Jet, versión 2.0.
dbVersion30	Motor de base de datos Microsoft Jet, versión 3.0.

Constantes del argumento opciones del método CreateDatabase (Todas son)

Constante	Description
dbEncrypt	Codifica la base de datos
dbVersion10	Motor de base de datos Microsoft Jet versión 1.0
dbVersion11	Motor de base de datos Microsoft Jet, versión 1.1.
dbVersion20	Motor de base de datos Microsoft Jet, versión 2.0.
dbVersion30	Motor de base de datos Microsoft Jet, versión 3.0.

Constantes del argumento tipo del método CreateWorkspace

Para cualquier objeto **Workspace**, la propiedad **Type**, el objeto **DBEngine** y la propiedad **DefaultType**, utilice cualquiera de las siguientes: (Todas son)

Constante	Description
dbUseODBC	El siguiente espacio de trabajo creado utilizará ODBCDirect.
dbUseJet	El siguiente espacio de trabajo creado utilizará el Motor de base de datos Microsoft Jet.

Constantes del argumento opciones del método Execute (Todas son)

Constante	Descripción
dbDenyWrite	Niega la autorización de escritura a otros usuarios (sólo espacios de trabajo Microsoft Jet).
dbInconsistent	Permite actualizaciones inconsistentes (sólo espacios de trabajo Microsoft Jet).
dbConsistent	Permite actualizaciones consistentes (sólo espacios de trabajo Microsoft Jet).
dbSQLPassThrough	El paso a través de SQL hace que la instrucción SQL pase a una base de datos ODBC para su procesamiento (sólo espacios de trabajo Microsoft Jet).
dbFailOnError	Deshace las actualizaciones si aparece un error (sólo espacios de trabajo Microsoft Jet).
dbSeeChanges	Genera un error de tiempo de ejecución si otro usuario trata de ver los datos que otro usuario está editando (sólo espacios de trabajo Microsoft Jet).
dbRunAsync	Ejecuta la consulta asíncronamente (sólo espacios de trabajo ODBCDirect).
dbExecDirect	Ejecuta la consulta sin llamar antes a la función ODBC **SQLPrepare** (sólo espacios de trabajo ODBCDirect).

Constantes del argumento opcional del método Idle (Esto es)

Constante	Descripción
dbRefreshCache	Graba las escrituras pendientes en disco y actualiza la memoria de los archivos de disco actuales.

Constantes del argumento opcional del método MakeReplica (Todas son)

Constante	Descripción
dbRepMakePartial	Crea una réplica parcial.
dbRepMakeReadOnly	Hace que los elementos replicables de la nueva base de datos sean de sólo lectura.

Constantes del argumento opciones de los métodos OpenConnection y OpenDatabase (Todas son)

Constante	Descripción
dbDriverNoPrompt	El administrador del controlador utiliza la cadena de conexión proporcionada en *conexión*. Si no se proporciona información suficiente, se devuelve un error interceptable.
dbDriverPrompt	El administrador del controlador muestra el cuadro de diálogo **Orígenes de datos** ODBC. La cadena de conexión utilizada para establecer la conexión se genera del nombre del origen de datos (DSN) seleccionado y completado por el usuario mediante los cuadros de diálogo.
dbDriverComplete	Si la cadena de conexión proporcionada incluye la palabra clave DSN, el administrador del controlador utiliza la cadena como se proporcionó en la conexión, en caso contrario se comporta como cuando se especifica **dbDriverPrompt**.
dbDriverCompleteRequired	(Predeterminado) Se comporta como **dbDriverComplete** excepto que el controlador desactiva los controles de cualquier información no necesaria para terminar la conexión.

Constantes del argumento tipo del método OpenRecordset (Todas son)

Constante	Descripción
dbOpenDynamic	Abre un objeto **Recordset** de tipo Dynamic (sólo espacios de trabajo ODBCDirect)
dbOpenDynaset	Abre el objeto **Recordset** de tipo Dynaset
dbOpenForwardOnly	Abre el objeto **Recordset** de tipo Forward-only
dbOpenSnapshot	Abre el objeto **Recordset** de tipo Snapshot.
dbOpenTable	Abre el **Recordset** de tipo Table (sólo espacios de trabajo Microsoft Jet)

Constantes del argumento bloquearmodificaciones del método OpenRecordset (Todas son)

Constante	Descripción
dbPessimistic	Concurrencia pesimista. El cursor utiliza el menor nivel de bloqueo

	suficiente para asegurar que el registro se puede actualizar.
dbReadOnly	El cursor es de sólo lectura. No se permiten actualizaciones.
dbOptimistic	Concurrencia optimista basada en el Id de registro. El cursor compara el Id de registro en los registros nuevo y viejo para determinar si se hicieron los cambios desde el último acceso al registro.
dbOptimisticValue	Concurrencia optimista basada en los valores del registro. El cursor compara los valores de datos en los registros nuevo y viejo para determinar si se hicieron los cambios desde el último acceso al registro (sólo espacios de trabajo ODBCDirect).
dbOptimisticBatch	Habilita las actualizaciones por lotes optimistas (sólo espacios de trabajo ODBCDirect).

Constantes del argumento opciones del método OpenRecordset (Todas son)

Constante	Descripción
dbDenyWrite	Impide a otros usuarios cambiar los registros del objeto **Recordset** (sólo espacios de trabajo Microsoft Jet).
dbDenyRead	Impide a otros usuarios leer los registros del objeto **Recordset** (de tipo Table sólo espacios de trabajo Microsoft Jet).
dbReadOnly	Abre el objeto **Recordset** como sólo lectura (sólo espacios de trabajo Microsoft Jet).
dbAppendOnly	Permite al usuario agregar nuevos registros a la hoja de respuestas dinámicas, pero el usuario no puede leer los registros existentes (de tipo Dynaset sólo espacios de trabajo Microsoft Jet).
dbInconsistent	Se aplica a todos los campos de la hoja de respuestas dinámicas aunque afecte a otros registros (de tipo Dynaset y Snapshot sólo espacios de trabajo Microsoft Jet).
dbConsistent	Sólo se aplica a los campos que no afectarán a otros registros de la hoja de respuestas dinámicas (de tipo Dynaset y Snapshot sólo en espacios de trabajo Microsoft Jet).
dbSQLPassThrough	Envía una instrucción SQL a una base de datos ODBC (de tipo

	Snapshot sólo espacios de trabajo Microsoft Jet).
dbForwardOnly	Crea un **Recordset** tipo Snapshot de movimiento progresivo (de tipo Snapshot sólo espacios de trabajo Microsoft Jet).
dbSeeChanges	Genera un error de tiempo de ejecución si otro usuario trata de ver los datos que otro usuario está editando (de tipo Dynaset sólo espacios de trabajo Microsoft Jet).
dbRunAsync	Ejecuta la consulta asíncronamente (sólo espacios de trabajo ODBCDirect).
dbExecDirect	Ejecuta la consulta sin antes llamar a la función ODBC **SQLPrepare** (sólo espacios de trabajo ODBCDirect).

Constantes de parámetro del método SetOption (Todas son)

Constante	Descripción
dbPageTimeout	La clave PageTimeout
dbSharedAsyncDelay	La clave SharedAsyncDelay
dbExclusiveAsyncDelay	La clave ExclusiveAsyncDelay
dbLockRetry	La clave LockRetry
dbUserCommitSync	La clave UserCommitSync
dbImplicitCommitSync	La clave ImplicitCommitSync
dbMaxBufferSize	La clave MaxBufferSize
dbMaxLocksPerFile	La clave MaxLocksPerFile
dbLockDelay	La clave LockDelay
dbRecycleLVs	La clave RecycleLVs
dbFlushTransactionTimeout	La clave FlushTransactionTimeout

Constantes del argumento interncambio del método Synchronize (Todas son)

Constante	Descripción
dbRepExportChanges	Envía los cambios de la base de datos activa a la base de datos de destino.
dbRepImportChanges	Recibe los cambios desde la base de datos de destino.
dbRepImpExpChanges	Envía y recibe datos en un intercambio bidireccional.
dbRepSyncInternet	Intercambia datos entre archivos conectados mediante una ruta de acceso Internet.

Constantes del argumento tipo del método Update (Todas son)

Constante	Descripción
dbUpdateRegular	(Predeterminado) Los cambios pendientes no se almacenan en memoria caché y se escriben en disco inmediatamente.
dbUpdateBatch	Todos los cambios pendientes en la memoria caché actualizada se escriben en disco.
dbUpdateCurrentRecord	Sólo los cambios pendientes en los registros activos se escriben en disco.

Constantes del argumento tipo del método CancelUpdate (Todas son)

Constante	Descripción
dbUpdateRegular	(Predeterminado) Los cambios pendientes no se almacenan en memoria caché y se escriben en disco inmediatamente.
dbUpdateBatch	Todos los cambios pendientes en la memoria caché actualizada se escriben en disco.

Ejemplo del objeto Container y la colección Containers

Este ejemplo enumera la colección **Containers** de la base de datos Neptuno y la colección **Properties** de cada objeto **Container** de la colección.

```
Sub ObjetoContainerX()

   Dim dbsNeptuno As Database
   Dim ctrBucle As Container
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno

      ' Enumera la colección Containers.
      For Each ctrBucle In .Containers
         Debug.Print "Propiedades de " & ctrBucle.Name _
            & " contenedor"

         ' Enumera la colección Properties de cada
         ' objeto Container.
         For Each prpBucle In ctrBucle.Properties
            Debug.Print "    " & prpBucle.Name _
               & " = " prpBucle
         Next prpBucle

      Next ctrBucle

      .Close
   End With

End Sub
```

Ejemplo del objeto DBEngine

Este ejemplo enumera las colecciones del objeto **DBEngine**. Para más ejemplos, vea los métodos y propiedades de **DBEngine**.

```
Sub DBEngineX()

   Dim wrkBucle As Workspace
   Dim prpBucle As Property

   With DBEngine
      Debug.Print "Propiedades de DBEngine "

      ' Enumera la colección Properties de DBEngine,
      ' para interceptar propiedades con valores no
      ' válidos en este contexto.
      For Each prpBucle In .Properties
         On Error Resume Next
         Debug.Print "    " & prpBucle.Name & " = " _
            & prpBucle
         On Error GoTo 0
      Next prpBucle

      Debug.Print "Colección Workspaces de DBEngine"

      ' Enumera la colección Workspaces de DBEngine.
      For Each wrkBucle In .Workspaces
         Debug.Print "    " & wrkBucle.Name

         ' Enumera la colección Properties de cada
         ' objeto Workspace, para interceptar
         ' propiedades con valores no válidos en este contexto.
         For Each prpBucle In wrkBucle.Properties
            On Error Resume Next
            Debug.Print "        " & prpBucle.Name & _
               " = " & prpBucle
            On Error GoTo 0
         Next prpBucle

      Next wrkBucle

   End With

End Sub
```

Ejemplo de un Recordset de tipo Dynaset

Este ejemplo abre un **Recordset** de tipo Dynaset y muestra el grado de actualización de los campos.

```
Sub dbOpenDynasetX()

   Dim dbsNeptuno As Database
   Dim rstFacturas As Recordset
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstFacturas = _
      dbsNeptuno.OpenRecordset("Facturas", dbOpenDynaset)

   With rstFacturas
      Debug.Print "Recordset de tipo Dynaset: " & .Name

      If .Updatable Then
         Debug.Print "    Campos Updatable :"

         ' Enumera la colección Fields del objeto
         ' Recordset de tipo Dynaset, imprimir sólo
         ' los campos actualizables.
         For Each fldBucle In .Fields
            If fldBucle.DataUpdatable Then
               Debug.Print "        " & fldBucle.Name
            End If
         Next fldBucle

      End If

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo del objeto Field y la colección Fields

Este ejemplo muestra las propiedades válidas de un objeto **Field** dependiendo de dónde resida el objeto **Field** (por ejemplo, la colección **Fields** de un **TableDef**, la colección **Fields** de un **QueryDef** y así sucesivamente). Se necesita el procedimiento SalidaField para poder ejecutar este procedimiento.

```
Sub FieldX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim fldDefTabla As Field
   Dim fldDefConsulta As Field
   Dim fldRecordset As Field
   Dim fldRelación As Field
   Dim fldÍndice As Field
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   ' Asigna un objeto Field de colecciones Fields
   ' diferentes a variables de objeto.
   Set fldDefTabla = _
      dbsNeptuno.TableDefs(0).Fields(0)
   Set fldDefConsulta =dbsNeptuno.QueryDefs(0).Fields(0)
   Set fldRecordset = rstEmpleados.Fields(0)
   Set fldRelación =dbsNeptuno.Relations(0).Fields(0)
   Set fldÍndice = _
      dbsNeptuno.TableDefs(0).Indexes(0).Fields(0)

   ' Imprime el informe.
   SalidaField "TableDef", fldDefTabla
   SalidaField "QueryDef", fldDefConsulta
   SalidaField "Recordset", fldRecordset
   SalidaField "Relation", fldRelación
   SalidaField "Index", fldÍndice

   rstEmpleados.Close
   dbsNeptuno.Close

End Sub

Sub SalidaField(strTemp As String, fldTemp As Field)
   ' Función de informe de FieldX.

   Dim prpBucle As Property

   Debug.Print "Propiedades de Field válidas en " & strTemp

   ' Enumera la colección Properties del objeto Field
   ' transferido.
   For Each prpBucle In fldTemp.Properties
      ' Algunas propiedades no son válidas en ciertos
      ' contextos (por ejemplo, la propiedad Value en
      ' la colección Fields de una TableDef).
```

```
      ' Cualquier intento de utilizar una propiedad no
      ' válida provocará un error.
      On Error Resume Next
      Debug.Print "    " & prpBucle.Name & " = " & _
         prpBucle.Value
      On Error GoTo 0
   Next prpBucle

End Sub
```

Ejemplo del objeto Group, la colección Groups, el objeto User y la colección Users

Este ejemplo muestra el uso de los objetos **Group** y **User** y las colecciones **Groups** y **Users**. En primer lugar, crea un nuevo objeto **User** y lo anexa a la colección **Users** del objeto **Workspace** predeterminado. A continuación, crea un nuevo objeto **Group** y lo anexa a la colección **Groups** del objeto **Workspace** predeterminado. Después el ejemplo agrega el usuario Patricia López al grupo Cuentas. Finalmente, enumera las colecciones **Users** y **Groups** del objeto **Workspace** predeterminado. Para más ejemplos, vea los métodos y propiedades enumerados en los temas resumen de **Group** y **User**.

```
Sub GroupX()

   Dim wrkPredeterminado As Workspace
   Dim usrNuevo As User
   Dim usrBucle As User
   Dim grpNuevo As Group
   Dim grpBucle As Group
   Dim grpMiembro As Group

   Set wrkPredeterminado = DBEngine.Workspaces(0)

   With wrkPredeterminado

      ' Crea y agrega un usuario nuevo.
      Set usrNuevo = .CreateUser("Patricia López", _
         "abc123DEF456", "Contraseña")
      .Users.Append usrNuevo

      ' Crea y agrega un grupo nuevo.
      Set grpNuevo = .CreateGroup("Cuentas", _
         "UVW987xyz654")
      .Groups.Append grpNuevo

      ' Hace que Patricia López sea un miembro del
      ' grupo Cuentas creando y agregando el objeto
      ' Group correspondiente a la colección Groups
      ' de usuarios. Si se crea y agrega un objeto
      ' User que representa a Patricia López a la
      ' Users del grupo Cuentas sucede
      ' lo mismo.
      Set grpMiembro = usrNuevo.CreateGroup("Cuentas")
      usrNuevo.Groups.Append grpMiembro

      Debug.Print "Colección Users:"

      ' Enumera todos los objetos User en el espacio
      ' de trabajo predeterminado de la colección Users.
      For Each usrBucle In .Users
         Debug.Print "    " & usrBucle.Name
         Debug.Print "        Pertenece a estos grupos:"

         ' Enumera todos los objetos Group en cada
         ' colección Groups del objeto User.
         If usrBucle.Groups.Count <> 0 Then
            For Each grpBucle In usrBucle.Groups
               Debug.Print "            " & _
```

```
                grpBucle.Name
            Next grpBucle
         Else
            Debug.Print "            [Ninguno]"
         End If

      Next usrBucle

      Debug.Print "Colección Groups:"

      ' Enumera todos los objetos Group en el espacio
      ' de trabajo predeterminado de la colección Groups.
      For Each grpBucle In .Groups
         Debug.Print "    " & grpBucle.Name
         Debug.Print "        tiene como miembros:"

         ' Enumera todos los objetos User en cada
         ' colección Users del objeto Group.
         If grpBucle.Users.Count <> 0 Then
            For Each usrBucle In grpBucle.Users
               Debug.Print "            " & _
                  usrBucle.Name
            Next usrBucle
         Else
            Debug.Print "            [Ninguno]"
         End If

      Next grpBucle

      ' Elimina los objetos User Group nuevos porque
      ' esto sólo es un ejemplo.
      .Users.Delete "Patricia López"
      .Groups.Delete "Cuentas"

   End With

End Sub
```

Ejemplo del objeto Index y la colección Indexes

Este ejemplo crea un objeto **Index** nuevo, los anexa a la colección **Indexes** de **TableDef** de Empleados y enumera la colección **Indexes** de **TableDef**. Finalmente, enumera un **Recordset**, primero utilizando el **Index** principal y después utilizando el **Index** nuevo. Se necesita el procedimiento SalidaÍndice para ejecutar este procedimiento.

```
Sub ObjetoIndexX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim idxNuevo As Index
   Dim idxBucle As Index
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno!Empleados

   With tdfEmpleados
      ' Crea un nuevo índice, crea y anexa objetos
      ' Field a la colección Fields.
      Set idxNuevo = .CreateIndex("NuevoÍndice")

      With idxNuevo
         .Fields.Append .CreateField("País")
         .Fields.Append .CreateField("Apellidos")
         .Fields.Append .CreateField("Nombre")
      End With

      ' Agrega un objeto Index nuevo a la colección
      ' Indexes de la colección de tabla Empleados.
      .Indexes.Append idxNuevo
      .Indexes.Refresh

      Debug.Print .Indexes.Count & " Indexado en " & _
         .Name & " TableDef"

      ' Enumera la colección Indexes de la tabla
      ' Empleados.
      For Each idxBucle In .Indexes
         Debug.Print "    " & idxBucle.Name
      Next idxBucle

      Set rstEmpleados = _
         dbsNeptuno.OpenRecordset("Empleados")

      ' Imprime un informe utilizando los índices nuevos y viejos.
      SalidaÍndice rstEmpleados, "ClavePpal"
      SalidaÍndice rstEmpleados, idxNuevo.Name
      rstEmpleados.Close

      ' Elimina el Index nuevo porque esto
      ' es un ejemplo.
      .Indexes.Delete idxNuevo.Name
   End With

   dbsNeptuno.Close
```

```
End Sub

Sub SalidaÍndice(rstTemp As Recordset, _
   strÍndice As String)
   ' Función de informe para FieldX.

   With rstTemp
      ' Establece el índice.
      .Index = strÍndice
      .MoveFirst
      Debug.Print "Recordset = " & .Name & _
         ", Índice = " & .Index
      Debug.Print "    IdEmpleado - País - Nombre"

      ' Enumera el recordset utilizando el índice
      ' especificado.
      Do While Not .EOF
         Debug.Print "    " & !IdEmpleado & " - " & _
            !País & " - " & !Apellidos & ", " & !Nombre
         .MoveNext
      Loop

   End With

End Sub
```

Ejemplo del objeto Parameter y la colección Parameters

Este ejemplo demuestra los objetos **Parameter** y la colección **Parameters** creando un **QueryDef** temporal y recuperando datos basados en cambios realizados al **QueryDef Parameters** de objetos. Se necesita el procedimiento CambioParámetros para ejecutar este procedimiento.

```
Sub ParameterX()

   Dim dbsNeptuno As Database
   Dim qdfInforme As QueryDef
   Dim prmInicio As Parameter
   Dim prmFin As Parameter

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Crea el objeto QueryDef temporal con dos
   ' parámetros.
   Set qdfInforme = dbsNeptuno.CreateQueryDef("", _
      "PARAMETERS FechaInicio DateTime, FechaFin DateTime; " & _
      "SELECT IdEmpleado, COUNT(IdPedido) AS NúmPedidos " & _
      "FROM Pedidos WHERE Fecha de envío BETWEEN " & _
      "[FechaInicio] AND [FechaFin] GROUP BY IdEmpleado " & _
      "ORDER BY IdEmpleado")
   Set prmInicio = qdfInforme.Parameters!FechaInicio
   Set prmFin = qdfInforme.Parameters!FechaFin

   ' Imprime el informe utilizando los valores de
   ' parámetros especificados.
   CambioParámetros qdfInforme, prmInicio, #1/1/95#, _
      prmFin, #6/30/95#
   CambioParámetros qdfInforme, prmInicio, #7/1/95#, _
      prmFin, #12/31/95#

   dbsNeptuno.Close

End Sub

Sub CambioParámetros(qdfTemp As QueryDef, _
   prmInicio As Parameter, dteInicio As Date, _
   prmFin As Parameter, dteFin As Date)
   ' Función de informe para ParameterX.

   Dim rstTemp As Recordset
   Dim fldBucle As Field

   ' Establece los valores de los parámetros y abre
   ' un recordset en el objeto QueryDef temporal.
   prmInicio = dteInicio
   prmFin = dteFin
   Set rstTemp = qdfTemp.OpenRecordset(dbOpenForwardOnly)
   Debug.Print "Desde " & dteInicio & " a " & dteFin

   ' Enumera el recordset.
   Do While Not rstTemp.EOF

      ' Enumera la colección Fields del recordset.
      For Each fldBucle In rstTemp.Fields
```

```
            Debug.Print " - " & fldBucle.Name & " = " & fldBucle;
        Next fldBucle

        Debug.Print
        rstTemp.MoveNext
    Loop

    rstTemp.Close

End Sub
```

Ejemplo del objeto Property y la colección Properties

Este ejemplo crea una propiedad definida por el usuario en la base de datos actual, establece las propiedades **Type** y **Value** y la anexa a la colección **Properties** de la base de datos. Después, el ejemplo enumera todas las propiedades en la base de datos. Para más ejemplos vea las propiedades enumeradas en el tema resumen **Property**.

```
Sub PropertyX()

   Dim dbsNeptuno As Database
   Dim prpNueva As Property
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Crea y anexa la propiedad definida por el usuario.
      Set prpNueva = .CreateProperty()
      prpNueva.Name = "DefinidaUsuario"
      prpNueva.Type = dbText
      prpNueva.Value = "Esta es una propiedad definida por el usuario."
      .Properties.Append prpNueva

      ' Enumera todas las propiedades de la base de datos actual.
      Debug.Print "Propiedades de " & .Name
      For Each prpBucle In .Properties
         With prpBucle
            Debug.Print "    " & .Name
            Debug.Print "        Tipo: " & .Type
            Debug.Print "        Valor: " & .Value
            Debug.Print "        Heredado: " & _
               .Inherited
         End With
      Next prpBucle

      ' Elimina la propiedad nueva porque
      ' esto es un ejemplo.
      .Properties.Delete "DefinidaUsuario"
   End With

End Sub
```

Ejemplo del objeto QueryDef y la colección QueryDefs

Este ejemplo crea un objeto **QueryDef** nuevo y lo anexa a la colección **QueryDefs** del objeto **Database** Neptuno. Después enumera la colección **QueryDefs** y la colección **Properties** del **QueryDef** nuevo.

```
Sub QueryDefX()

   Dim dbsNeptuno As Database
   Dim qdfNuevo As QueryDef
   Dim qdfBucle As QueryDef
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Crea un objeto QueryDef nuevo. Ya que tiene un
   ' nombre, se anexa automáticamente a la colección
   ' QueryDefs.
   Set qdfNuevo = dbsNeptuno.CreateQueryDef("NewQueryDef", _
         "SELECT * FROM Categorías")

   With dbsNeptuno
      Debug.Print .QueryDefs.Count & _
         "QueryDefs en " & .Name

      ' Enumera la colección QueryDefs.
      For Each qdfBucle In .QueryDefs
         Debug.Print "    " & qdfBucle.Name
      Next qdfBucle

      With qdfNuevo
         Debug.Print "Propiedades de " & .Name

         ' Enumera la colección Properties del objeto
         ' QueryDef nuevo.
         For Each prpBucle In .Properties
            On Error Resume Next
            Debug.Print "    " & prpBucle.Name & " - " & _
               IIf(prpBucle = "", "[vacío]", prpBucle)
            On Error Goto 0
         Next prpBucle
      End With

      ' Elimina el QueryDef nuevo porque
      ' esto es un ejemplo.
      .QueryDefs.Delete qdfNuevo.Name
      .Close
   End With

End Sub
```

Ejemplo del objeto Recordset y la colección Recordsets

Este ejemplo demuestra los objetos **Recordset** y la colección **Recordsets** abriendo cuatro tipos diferentes de **Recordsets**, enumerando la colección **Recordsets** de la **Database** actual y enumerando la colección **Properties** de cada **Recordset**.

```
Sub RecordsetX()

   Dim dbsNeptuno As Database
   Dim rstTable As Recordset
   Dim rstDynaset As Recordset
   Dim rstSnapshot As Recordset
   Dim rstForwardOnly As Recordset
   Dim rstBucle As Recordset
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno

      ' Abre cada tipo de objeto Recordset.
      Set rstTable = .OpenRecordset("Categorías", _
         dbOpenTable)
      Set rstDynaset = .OpenRecordset("Empleados", _
         dbOpenDynaset)
      Set rstSnapshot = .OpenRecordset("Compañías de envíos", _
         dbOpenSnapshot)
      Set rstForwardOnly = .OpenRecordset _
         ("Empleados", dbOpenForwardOnly)

      Debug.Print "Recordsets en la colección " & _
         " Recordsets de dbsNeptuno"

      ' Enumera la colección Recordsets.
      For Each rstBucle In .Recordsets

         With rstBucle
            Debug.Print "    " & .Name

            ' Enumera la colección Properties de cada
            ' objeto Recordset. Bloquea cualquier
            ' propiedad cuyo valor no es válido
            ' en este contexto.
            For Each prpBucle In .Properties
               On Error Resume Next
               If prpBucle <> "" Then Debug.Print _
                  "        " & prpBucle.Name & _
                  " = " & prpBucle
               On Error GoTo 0
            Next prpBucle

         End With

      Next rstBucle

      rstTable.Close
      rstDynaset.Close
```

```
      rstSnapshot.Close
      rstForwardOnly.Close

      .Close
   End With

End Sub
```

Ejemplo del objeto Relation y la colección Relations

Este ejemplo muestra cómo un objeto **Relation** existente puede controlar la entrada de datos. El procedimiento intenta agregar un registro con un IdCategoría deliberadamente incorrecta; esto desencadena una rutina de tratamiento de errores.

```
Sub RelationX()

   Dim dbsNeptuno As Database
   Dim rstProductos As Recordset
   Dim prpBucle As Property
   Dim fldBucle As Field
   Dim errBucle As Error

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstProductos = dbsNeptuno.OpenRecordset("Productos")

   ' Imprime un informe que muestra las diferentes
   ' partes de la relación y dónde se almacena cada fragmento.
   With dbsNeptuno.Relations!Categorías
      Debug.Print "Propiedades de " & .Name & " Relation"
      Debug.Print "    Tabla = " & .Table
      Debug.Print "    Tabla Externa = " & .ForeignTable
      Debug.Print "Campos de " & .Name & " Relación"
      With .Fields!IdCategoría
         Debug.Print "    " & .Name
         Debug.Print "        Nombre = " & .Name
         Debug.Print "        Nombre Externo = " & .ForeignName
      End With
   End With

   ' Intenta agregar un registro que infringe la relación.
   With rstProductos
      .AddNew
      !NombreProducto = "Trygve's Lutefisk"
      !IdCategoría = 10
      On Error GoTo Err_Relation
      .Update
      On Error GoTo 0
      .Close
   End With

   dbsNeptuno.Close

   Exit Sub

Err_Relation:

   ' Informa al usuario de cualquier error que
   ' provengan de datos no válidos.
   If DBEngine.Errors.Count > 0 Then
      For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & _
            vbCr & errBucle.Description
      Next errBucle
   End If
```

```
    Resume Next

End Sub
```

Ejemplo de un Recordset de tipo Snapshot

Este ejemplo abre un **Recordset** de tipo Snapshot y demuestra su características de sólo lectura.

```
Sub dbOpenSnapshotX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenSnapshot)

   With rstEmpleados
      Debug.Print "Recordset de tipo Snapshot: " & _
         .Name

      ' Enumera la colección Properties del objeto
      ' Recordset de tipo Snapshot, bloqueando
      ' las propiedades con valores no válidos
      ' en este contexto.
      For Each prpBucle In .Properties
         On Error Resume Next
         Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
         On Error Goto 0
      Next prpBucle

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de un Recordset de tipo Table

Este ejemplo abre una **Recordset** de tipo Table, establece la propiedad **Index** y enumera los registros.

```
Sub dbOpenTableX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   'Es predeterminado dbOpenTable.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      Debug.Print "Recordset de tipo Table: " & .Name

      ' Utiliza un índice predefinido.
      .Index = "Apellido"
      Debug.Print "    Índice = " & .Index

      ' Enumera los registros.
      Do While Not .EOF
         Debug.Print "        " & !Apellidos & ", " & _
            !Nombre
         .MoveNext
      Loop

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo del objeto TableDef y la colección TableDefs

Este ejemplo crea un **TableDef** nuevo y lo anexa a la colección **TableDefs** del objeto **Database** Neptuno. Después enumera la colección **TableDefs** y la colección **Properties** del **TableDef** nuevo.

```
Sub TableDefX()

   Dim dbsNeptuno As Database
   Dim tdfNuevo As TableDef
   Dim tdfBucle As TableDef
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Crea un objeto TableDef nuevo, anexa los objetos
   ' Field a la colección Fields y anexa el objeto
   ' TableDef a la colección TableDefs del objeto
   ' Database.
   Set tdfNuevo = dbsNeptuno.CreateTableDef("NuevoTableDef")
   tdfNuevo.Fields.Append tdfNuevo.CreateField("Fecha", dbDate)
   dbsNeptuno.TableDefs.Append tdfNuevo

   With dbsNeptuno
      Debug.Print .TableDefs.Count & _
         "TableDefs en " & .Name

      ' Enumera la colección TableDefs.
      For Each tdfBucle In .TableDefs
         Debug.Print "    " & tdfBucle.Name
      Next tdfBucle

      With tdfNuevo
         Debug.Print "Propiedades de " & .Name

         ' Enumera la colección Properties del objeto
         ' TableDef nuevo, sólo imprime las
         ' propiedades con valores no vacíos.
         For Each prpBucle In .Properties
            Debug.Print "    " & prpBucle.Name & " - " & _
               IIf(prpBucle = "", "[vacío]", prpBucle)
         Next prpBucle

      End With

      ' Elimina el TableDef nuevo ya que esto es un
      ' ejemplo.
      .TableDefs.Delete tdfNuevo.Name
      .Close
   End With

End Sub
```

Ejemplo del objeto Connection y de la colección Connections

Este ejemplo demuestra el objeto **Connection** y la colección **Connections** abriendo un objeto **Database** de Microsoft Jet y dos objetos de **Connection** de ODBCDirect, también muestra la lista de las propiedades disponibles para cada objeto.

```
Sub ObjetoConnectionX()

   Dim wrkJet as Workspace
   Dim dbsNeptuno As Database
   Dim wrkODBC As Workspace
   Dim conEditores As Connection
   Dim conEditores2 As Connection
   Dim conBucle As Connection
   Dim prpBucle As Property

   ' Abre objeto Database de Microsoft Jet.
   Set wrkJet = CreateWorkspace("NuevoWorkspaceJet", _
      "admin", "", dbUseJet)
   Set dbsNeptuno = wrkJet.OpenDatabase("Neptuno.mdb")

   ' Crea el objeto Workspace de ODBCDirect y abre los objetos
   ' Connection.
   Set wrkODBC = CreateWorkspace("NuevoWorkspaceODBC", _
      "admin", "", dbUseODBC)
   Set conEditores = wrkODBC.OpenConnection("Conexión1", , , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")
   Set conEditores2 = wrkODBC.OpenConnection("Conexión2", , _
      True, "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   Debug.Print "Propiedades de Database:"

   With dbsNeptuno
      ' Enumera la colección Properties del objeto Database.
      For Each prpBucle In .Properties
         On Error Resume Next
         Debug.Print "    " & prpBucle.Name & " = " & _
            prpBucle.Value
         On Error GoTo 0
      Next prpBucle
   End With

   ' Enumera la colección Connections.
   For Each conBucle In wrkODBC.Connections
      Debug.Print "Propiedades de Connection para " & _
         conBucle.Name & ":"

      With conBucle
         ' Imprime los valores de la propiedad llamando explícitamente
         ' a cada objeto Property; el objeto Connection no
         ' admite la colección Properties.
         Debug.Print "    Connect = " & .Connect
         ' Property devuelve un objeto Database
         Debug.Print "    Database[.Name] = " & _
            .Database.Name
         Debug.Print "    Name = " & .Name
         Debug.Print "    QueryTimeout = " & .QueryTimeout
```

```
            Debug.Print "    RecordsAffected = " & _
               .RecordsAffected
            Debug.Print "    StillExecuting = " & _
               .StillExecuting
            Debug.Print "    Transactions = " & .Transactions
            Debug.Print "    Updatable = " & .Updatable
         End With

      Next conBucle

      dbsNeptuno.Close
      conEditores.Close
      conEditores2.Close
      wrkJet.Close
      wrkODBC.Close

   End Sub
```

Ejemplo del objeto Database y de la colección Databases

Este ejemplo crea un objeto nuevo de **Database** y abre un objeto existente de **Database** en el objeto **Workspace** predeterminado. A continuación enumera la colección **Database** y la colección **Properties** de cada objeto **Database**. Para ver ejemplos adicionales, vea la lista de métodos y propiedades en el tema resumen de **Database.**

```
Sub ObjetoDatabaseX()

   Dim wrkJet As Workspace
   Dim dbsNeptuno As Database
   Dim dbsNueva As Database
   Dim dbsBucle As Database
   Dim prpBucle As Property

   Set wrkJet = CreateWorkspace("WorkspaceJet", "admin", _
      "", dbUseJet)

   ' Comprueba que no hay ya un archivo con el nombre de la
   ' base de datos nueva.
      If Dir("NuevaBD.mdb") <> "" Then Kill "NuevaBD.mdb"

   ' Crea una nueva base de datos con la secuencia
   ' de ordenación especificada'.
   Set dbsNueva = wrkJet.CreateDatabase("NuevaBD.mdb", _
      dbLangGeneral)
   Set dbsNeptuno = wrkJet.OpenDatabase("Neptuno.mdb")

   ' Enumera la colección Database.
   For each dbsBucle In wrkJet.Databases
      With dbsBucle
         Debug.Print "Properties de " & .Name
         ' Enumerar la colección Properties de
         ' cada objeto Database.
         For Each prpBucle In .Properties
            If prpBucle <> "" Then Debug.Print "    " & _
               prpBucle.Name & " = " & prpBucle
         Next prpBucle
      End With
   Next dbsBucle

   dbsNueva.Close
   dbsNeptuno.Close
   wrkJet.Close

End Sub
```

Ejemplo del objeto Document y de la colección Documents

Este ejemplo enumera la colección **Documents** del contenedor **Tables** y luego enumera la colección **Properties** del primer objeto **Document** en la colección.

```
Sub DocumentX()

   Dim dbsNeptuno As Database
   Dim docBucle As Document
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno.Containers!Tables
      Debug.Print "Documents en el contenedor " & .Name
      ' Enumera la colección Documents del contenedor
      ' Tables
      For Each docBucle In .Documents
         Debug.Print "    " & docBucle.Name
      Next docBucle
      With .Documents(0)
         ' Enumera la colección Properties del primer
         ' objeto Document del contenedor Tables.
         Debug.Print "Properties de Document " & .Name
         On Error Resume Next
         For Each prpBucle In .Properties
            Debug.Print "    " & prpBucle.Name & " = " & _
               prpBucle
         Next prpBucle
         On Error GoTo 0
      End With
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de Recordset de tipo Dynamic

Este ejemplo abre un objeto **Recordset** de tipo Dynamic y enumera sus registros.

```
Sub dbAbrirDynamicX()

   Dim wrkPrin As Workspace
   Dim conPrin As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset
   Dim strSQL As String
   Dim intBucl As Integer

   ' Crea un espacio de trabajo ODBC y abre la conexión a
   ' una base de datos del servidor SQL.
   Set wrkPrin = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conPrin = wrkPrin.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Editores")
   ' Open dynamic-type recordset.
   Set rstTemp = _
      conPrin.OpenRecordset("autores", _
      dbOpenDynamic)

   With rstTemp
      Debug.Print "Recorset de tipo Dynamic: " & .Name

      ' Enumera los registros.
      Do While Not .EOF
         Debug.Print "        " & !au_lname & ", " & _
            !au_fname
         .MoveNext
      Loop

      .Close
   End With

   conPrin.Close
   wrkPrin.Close

End Sub
```

Ejemplo del objeto Recordset de tipo Forward-only

Este ejemplo abre un **Recordset** de tipo Forward-only, demuestra sus características de sólo lectura y se mueve por el objeto **Recordset** con el método **MoveNext**.

```
Sub dbOpenForwardOnlyX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Abre un objeto Recordset de tipo Forward-only. Sólo se
   ' pueden utilizar los métodos MoveNext y Move para desplazarse
   ' por el Recordset.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenForwardOnly)

   With rstEmpleados
      Debug.Print "Recordset de tipo Forward-only: " & _
         .Name & ", Updatable = " & .Updatable

      Debug.Print "    Campo - DataUpdatable"
      ' Enumera la colección Fields imprimiendo las propiedades Name y
DataUpdatable de
      ' cada objeto Field.
      For Each fldBucle In .Fields
         Debug.Print "        " & _
            fldBucle.Name & " - " & fldBucle.DataUpdatable
      Next fldBucle

      Debug.Print "    Datos"
      ' Enumera el Recordset.
      Do While Not .EOF
         Debug.Print "        " & !Nombre & " " & _
            !Apellidos
         .MoveNext
      Loop

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo del objeto Workspace y de la colección Workspaces

Este ejemplo crea un nuevo objeto **Workspace** de Microsoft Jet y un objeto nuevo **Workspace** de ODBCDirect y los anexa a la colección **Workspaces**. A continuación enumera la colección **Workspaces** y **Properties** de cada objeto **Workspace**. Para más ejemplos, vea métodos y propiedades del objeto **Workspace** o la colección **Workspaces**.

```
Sub WorkspaceX()

   Dim wrkNuevoJet  As Workspace
   Dim wrkNuevoODBC As Workspace
   Dim wrkBucle As Workspace
   Dim prpBucle As Property

   ' Crea un nuevo Workspace Microsoft Jet.
   Set wrkNuevoJet  = CreateWorkspace("NuevoWorkspaceJet", _
      "admin", "", dbUseJet)
   Workspaces.Append wrkNuevoJet

   ' Crea un nuevo Workspace ODBCDirect.
   Set wrkNuevoODBC = CreateWorkspace("NuevoWorkspaceODBC", _
      "admin", "", dbUseODBC)
   Workspaces.Append wrkNuevoODBC

   ' Enumera la colección Workspaces.
   For Each wrkBucle In Workspaces
      With wrkBucle
         Debug.Print "Properties de " & .Name
         ' Enumera la colección Properties del nuevo
         ' objeto Workspace.
         For Each prpBucle In .Properties
            On Error Resume Next
            If prpBucle <> "" Then Debug.Print "    " & _
               prpBucle.Name & " = " & prpBucle
            On Error GoTo 0
         Next prpBucle
      End With
   Next wrkBucle

   wrkNuevoJet.Close
   wrkNuevoODBC.Close

End Sub
```

Ejemplo del método AddNew

Este ejemplo utiliza el método **AddNew** para crear un registro nuevo con el nombre especificado. Se necesita la función AgregarNombre para ejecutar este procedimiento.

```
Sub AddNewX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strNombre As String
   Dim strApellidos As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", dbOpenDynaset)

   ' Obtiene datos del usuario.
   strNombre = Trim(InputBox( _
      "Introduzca el nombre:"))
   strApellidos = Trim(InputBox( _
      "Introduzca los apellidos:"))

   ' Sólo se ejecuta si el usuario escribe algo
   ' en los dos campos.
   If strNombre <> "" and strApellidos <> "" Then

      ' Llama a la función que agrega el registro.
      AgregarNombre rstEmpleados, strNombre, strApellidos

      ' Muestra los datos agregados más recientemente.
      With rstEmpleados
         Debug.Print "Registro nuevo: " & !Nombre & _
            " " & !Apellidos
         ' Elimina el registro nuevo porque esto es un ejemplo.
         .Delete
      End With

   Else
      Debug.Print _
         "¡Debe escribir una cadena para el nombre y los apellidos!"
   End If

   rstEmpleados.Close
   dbsNeptuno.Close

End Sub

Function AgregarNombre(rstTemp As Recordset, _
   strNomrbe As String, strApellidos As String)

   ' Agrega un registro nuevo al Recordset utilizando
   ' datos transferidos del procedimiento que llama.
   ' El registro nuevo pasa a ser el registro actual.
   With rstTemp
      .AddNew
      !Nombre = strNomrbe
      !Apellidos = strApellidos
```

```
      .Update
      .Bookmark = .LastModified
   End With

End Function
```

Ejemplo de los métodos Append y Delete

Este ejemplo utiliza tanto el método **Append** como el método **Delete** para modificar la colección **Fields** de un objeto **TableDef**. Se necesita el procedimiento AnexarEliminarCampo para ejecutar este procedimiento.

```
Sub AppendX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno.TableDefs!Empleados

   ' Agrega tres campos nuevos.
   AnexarEliminarCampo tdfEmpleados, "APPEND", _
      "Correo electrónico", dbText, 50
   AnexarEliminarCampo tdfEmpleados, "APPEND", _
      "Http", dbText, 80
   AnexarEliminarCampo tdfEmpleados, "APPEND", _
      "Cuota", dbInteger, 5

   Debug.Print "Fields después de Append"
   Debug.Print , "Tipo", "Tamaño", "Nombre"

   ' Enumera la colección Fields para mostrar los campos nuevos.
   For Each fldBucle In tdfEmpleados.Fields
      Debug.Print , fldBucle.Type, fldBucle.Size, fldBucle.Name
   Next fldBucle

   ' Elimina los campos agregados más recientemente.
   AnexarEliminarCampo tdfEmpleados, "DELETE", "Correo electrónico"
   AnexarEliminarCampo tdfEmpleados, "DELETE", "Http"
   AnexarEliminarCampo tdfEmpleados, "DELETE", "Cuota"

   Debug.Print "Fields después de Delete"
   Debug.Print , "Tipo", "Tamaño", "Nombre"

   ' Enumera la colección Fields para mostrar que los
   ' campos nuevos se han eliminado.
   For Each fldBucle In tdfEmpleados.Fields
      Debug.Print , fldBucle.Type, fldBucle.Size, fldBucle.Name
   Next fldBucle

   dbsNeptuno.Close

End Sub

Sub AnexarEliminarCampo(tdfTemp As TableDef, _
   strComando As String, strNombre As String, _
   Optional varType, Optional varSize)

   With tdfTemp

      ' Comprueba primero para ver si el objeto
      ' TableDef se puede actualizar. Si no, el
```

```
      ' control vuelve al procedimiento que lo llama.
      If .Updatable = False Then
         MsgBox "¡TableDef no se puede actualizar! " & _
            "No se puede terminar la tarea."
         Exit Sub
      End If

      ' Dependiendo de los datos transferidos, se
      ' agrega o elimina un campo de la colección
      ' Fields del objeto TableDef especificado.
      If strComando = "APPEND" Then
         .Fields.Append .CreateField(strNombre, _
            varType, varSize)
      Else
         If strComando = "DELETE" Then .Fields.Delete _
            strNombre
      End If

   End With

End Sub
```

Ejemplo de los métodos AppendChunk y GetChunk

Este ejemplo utiliza los métodos **AppendChunk** y **GetChunk** para llenar un campo de tipo Objeto OLE con datos de otro registro**,** 32 KB de cada vez. En una aplicación real, podría utilizar un procedimiento como este para copiar un registro de empleado (incluyendo su foto) de una tabla a otra. En este ejemplo, el registro simplemente se copia de nuevo en la misma tabla. Observe que todo tratamiento de fragmentos tiene lugar dentro de una secuencia **AddNew**-**Update** sencilla.

```
Sub AppendChunkX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim rstEmpleados2 As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Abre dos recordsets de la tabla Empleados.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)
   Set rstEmpleados2 = rstEmpleados.Clone

   ' Agrega un registro nuevo al primer Recordset y
   ' copia los datos de un registro en el segundo Recordset.
   With rstEmpleados
      .AddNew
      !Nombre = rstEmpleados2!Nombre
      !Apellidos = rstEmpleados2!Apellidos
      CopiarCampoGrande rstEmpleados2!Foto, !Foto
      .Update

      ' Elimina el registro nuevo porque esto es un ejemplo.
      .Bookmark = .LastModified
      .Delete
      .Close
   End With

   rstEmpleados2.Close
   dbsNeptuno.Close

End Sub

Function CopiarCampoGrande(fldOrigen As Field, _
   fldDestino As Field)

   ' Establece el tamaño del fragmento en bytes.
   Const conChunkSize = 32768

   Dim lngCompensación As Long
   Dim lngTamañoTotal As Long
   Dim strChunk As String

   ' Copia la foto de un Recordset a otro en
   ' fragmentos de 32K hasta que se copia el campo en su totalidad.
   lngTamañoTotal = fldOrigen.FieldSize
   Do While lngCompensación < lngTamañoTotal
```

```
      strChunk = fldOrigen.GetChunk(lngCompensación, conChunkSize)
      fldDestino.AppendChunk strChunk
      lngCompensación = lngCompensación + conChunkSize
   Loop

End Function
```

Ejemplo de los métodos BeginTrans, CommitTrans y Rollback

Este ejemplo cambia el cargo de todos los representantes de ventas en la tabla Empleados de la base de datos. Después del método **BeginTrans** inicia una transacción que aisla todos los cambios realizados a la tabla Empleados, el método **CommitTrans** guarda los mismos. Observe que puede utilizar el método **Rollback** para deshacer los cambios que ha guardado utilizando el método **Update**. Además, la transacción principal se anida dentro de otra transacción que automáticamente deshace cualquier cambio realizado por el usuario durante este ejemplo.

Permanecen bloqueadas una o más páginas de la tabla mientras que el usuario decide si acepta o no los cambios. Por esta razón, esta técnica no se aconseja y sólo se muestra como un ejemplo.

```
Sub BeginTransX()

   Dim strNombre As String
   Dim strMensaje As String
   Dim wrkPredeterminado As Workspace
   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   ' Obtiene el Workspace predeterminado.
   Set wrkPredeterminado = DBEngine.Workspaces(0)
   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   ' Inicia la transacción de salida.
   wrkPredeterminado.BeginTrans
   ' Inicia la transacción principal.
   wrkPredeterminado.BeginTrans

   With rstEmpleados

      ' Hace un bucle en el Recordset y pregunta al
      ' usuario si quiere cambiar el cargo de un empleado específico.
      Do Until .EOF
         If !Cargo = "Representante de ventas" Then
            strNombre = !Apellidos & ", " & !Nombre
            strMensaje = "Empleado: " & strNombre & vbCr & _
               "¿Desea cambiar el cargo a Ejecutivo de cuentas?"

            ' Cambia el cargo del empleado especificado.
            If MsgBox(strMensaje, vbYesNo) = vbYes Then
               .Edit
               !Cargo = "Ejecutivo de cuentas"
               .Update
            End If
         End If

         .MoveNext
      Loop

      ' Pregunta si el usuario desea validar todos
      ' los cambios realizados anteriormente.
      If MsgBox("¿Desea guardar los cambios?", vbYesNo) = vbYes Then
         wrkPredeterminado.CommitTrans
      Else
```

```
            wrkPredeterminado.Rollback
        End If

        ' Imprime los datos actuales en el Recordset.
        .MoveFirst
        Do While Not .EOF
            Debug.Print !Apellidos & ", " & !Nombre & _
                " - " & !Cargo
            .MoveNext
        Loop

        ' Deshace cualquier cambio realizado por el
        ' usuario ya que esto es un ejemplo.
        wrkPredeterminado.Rollback
        .Close
    End With

    dbsNeptuno.Close

End Sub
```

Ejemplo del método Clone

Este ejemplo utiliza el método **Clone** para crear copias de un objeto **Recordset** y permite que el usuario sitúe independientemente el puntero del registro de cada copia.

```
Sub CloneX()

   Dim dbsNeptuno As Database
   Dim arstProductos(1 To 3) As Recordset
   Dim intBucle As Integer
   Dim strMensaje As String
   Dim strEncontrar As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Si la siguiente instrucción SQL se va a utilizar
   ' frecuentemente, será mejor crear un QueryDef
   ' permanente para mejorar el rendimiento.
   Set arstProductos(1) = dbsNeptuno.OpenRecordset( _
      "SELECT NombreProducto FROM Productos " & _
      "ORDER BY NombreProducto", dbOpenSnapshot)

   ' Crea dos copias del Recordset original.
   Set arstProductos(2) = arstProductos(1).Clone
   Set arstProductos(3) = arstProductos(1).Clone

   Do While True

      ' Hace un bucle en la matriz de modo que, el
      ' usuario en cada pasada busca una copia
      ' diferente del mismo Recordset.
      For intBucle = 1 To 3

         ' Pregunta por la cadena de búsqueda mientras
         ' muestra donde está el puntero del registro activo de cada
Recordset.
         strMensaje = _
            "Recordsets de la tabla Productos:" & vbCr & _
            "  1 - Original - Puntero del registro en " & _
            arstProductos(1)!NombreProducto & vbCr & _
            "  2 - Copia- Puntero del registro en " & _
            arstProductos(2)!NombreProducto & vbCr & _
            "  3 - Copia - Puntero del registro en " & _
            arstProductos(3)!NombreProducto & vbCr & _
            "Introduzca la cadena de búsqueda para #" & intBucle & ":"
         strEncontrar = Trim(InputBox(strMensaje))
         If strEncontrar = "" Then Exit Do

         ' Encuentra la cadena de búsqueda; si no hay
         ' coincidencia, salta al último registro.
         With arstProductos(intBucle)
            .FindFirst "NombreProducto >= '" & strEncontrar & "'"
            If .NoMatch Then .MoveLast
         End With

      Next intBucle
```

```
    Loop

    arstProductos(1).Close
    arstProductos(2).Close
    arstProductos(3).Close
    dbsNeptuno.Close

End Sub
```

Ejemplo del método Close

Este ejemplo utiliza el método **Close** en los objetos **Recordset** y **Database** que se han abierto. También demuestra cómo al cerrar un **Recordset** podrá perder los cambios no guardados.

```
Sub CloseX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   ' Realiza cambios a un registro pero cierra el
   ' Recordset antes de guardar los cambios.
   With rstEmpleados
      Debug.Print "Datos originales"
      Debug.Print "    Nombre - Extensión"
      Debug.Print "    " & !Nombre & " " & _
         !Apellidos & " - " & !Extensión
      .Edit
      !Extensión = "9999"
      .Close
   End With

   ' Vuelve a abrir el Recordset para mostrar que los
   ' datos no se cambiaron.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      Debug.Print "Datos después de Close"
      Debug.Print "    Nombre - Extensión"
      Debug.Print "    " & !Nombre & " " & _
         !Apellidos & " - " & !Extensión
      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo del método CompactDatabase

Este ejemplo utiliza el método **CompactDatabase** para cambiar el orden de una base de datos. No puede utilizar este código en un módulo que pertenezca a Neptuno.mdb.

```
Sub CompactDatabaseX()

   Dim dbsNeptuno As Database

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Muestra las propiedades de la base de datos original.
   With dbsNeptuno
      Debug.Print .Name & ", versión " & .Version
      Debug.Print "    Secuencia de ordenación = " & .CollatingOrder
      .Close
   End With

   ' Asegúrese de que no existe un archivo con el
   ' nombre de la base de datos compactada.
   If Dir("NeptunCo.mdb") <> "" Then _
      Kill "NeptunCo.mdb"

   ' Esta instrucción crea una versión compactada de
   ' la base de datos Neptuno que utiliza la secuencia
   ' de intercalación del lenguaje coreano.
   DBEngine.CompactDatabase "Neptuno.mdb", _
      "NeptunCo.mdb", dbLangKorean

   Set dbsNeptuno = OpenDatabase("NeptunCo.mdb")

   ' Muestra las propiedades de la base de datos compactada.
   With dbsNeptuno
      Debug.Print .Name & ", versión " & .Version
      Debug.Print "    Secuencia de ordenación = " & .CollatingOrder
      .Close
   End With

End Sub
```

Este ejemplo utiliza el método **CompactDatabase** para cambiar la versión de la base de datos. Para ejecutar este código, debe tener una base de datos Microsoft Jet versión 1.1 llamada Neptun11.mdb y no puede utilizar este código en un módulo que pertenezca a Neptun11.mdb.

```
Sub CompactDatabaseX2()

   Dim dbsNeptuno As Database
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptun11.mdb")

   ' Muestra las propiedades de la base de datos original.
   With dbsNeptuno
      Debug.Print .Name & ", versión " & .Version
      Debug.Print "    Secuencia de ordenación = " & .CollatingOrder
      .Close
   End With
```

```
    ' Asegúrese de que no existe un archivo con el
    ' nombre de la base de datos compactada.
    If Dir("Neptun20.mdb") <> "" Then _
       Kill "Neptun20.mdb"

    ' Este instrucción crea una base de datos
    ' Microsoft Jet versión 2.0 compactada y encriptada
    ' de la base de datos Microsoft Jet versión 1.1.
    DBEngine.CompactDatabase "Neptun11.mdb", _
       "Neptun20.mdb", , dbEncrypt + dbVersion20

    Set dbsNeptuno = OpenDatabase("Neptun20.mdb")

    ' Muestra las propiedades de la base de datos compactada.
    With dbsNeptuno
       Debug.Print .Name & ", versión " & .Version
       For Each prpBucle In .Properties
          On Error Resume Next
          If prpBucle <> "" Then Debug.Print "    " & _
             prpBucle.Name & " = " & prpBucle
          On Error GoTo 0
       Next prpBucle
       .Close
    End With

End Sub
```

Ejemplo del método CreateDatabase

Este ejemplo utiliza **CreateDatabase** para crear un objeto **Database** encriptado nuevo.

```
Sub CreateDatabaseX()

   Dim wrkPredeterminado As Workspace
   Dim dbsNueva As DATABASE
   Dim prpBucle As Property

   ' Obtiene el Workspace predeterminado.
   Set wrkPredeterminado = DBEngine.Workspaces(0)

   ' Asegúrese de que no existe un archivo con el
   ' nombre de la base de datos nueva.
   If Dir("BDNueva.mdb") <> "" Then Kill "BDNueva.mdb"

   ' Crea a una base de datos nueva encriptada con la
   ' secuencia de intercalación especificada.
   Set dbsNueva = wrkPredeterminado.CreateDatabase("BDNueva.mdb", _
      dbLangGeneral, dbEncrypt)

   With dbsNueva
      Debug.Print "Properties de " & .Name
      ' Enumera la colección Properties del objeto
      ' Database nuevo.
      For Each prpBucle In .Properties
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
      Next prpBucle
   End With

   dbsNueva.Close

End Sub
```

Ejemplo del método CreateField

Este ejemplo utiliza el método **CreateField** para crear tres **Fields** para una **TableDef** nueva. Después muestra las propiedades de estos objetos **Field** que establece automáticamente el método **CreateField**. (No se muestran las propiedades cuyos valores están vacíos en el momento de la creación de **Field**.)

```
Sub CreateFieldX()

   Dim dbsNeptuno As Database
   Dim tdfNuevo As TableDef
   Dim fldBucle As Field
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   Set tdfNuevo = dbsNeptuno.CreateTableDef("NuevoTableDef")

   ' Crea y agrega los objetos Field nuevos para el
   ' objeto TableDef nuevo.
   With tdfNuevo
      ' El método CreateField establecerá un tamaño
      ' predeterminado para un objeto Field nuevo si no se especifica
nada.
      .Fields.Append .CreateField("CampoTexto", dbText)
      .Fields.Append .CreateField("CampoEntero", dbInteger)
      .Fields.Append .CreateField("CampoFecha", dbDate)
   End With

   dbsNeptuno.TableDefs.Append tdfNuevo

   Debug.Print "Propiedades de Fields nuevo en " & tdfNuevo.Name

   ' Enumera la colección Fields para mostrar las
   ' propiedades de los objetos Field nuevos.
   For Each fldBucle In tdfNuevo.Fields
      Debug.Print "    " & fldBucle.Name

      For Each prpBucle In fldBucle.Properties
         ' Las propiedades no válidas en el contexto
         ' de TableDefs producirán un error y se
         ' intentan leer sus valores.
         On Error Resume Next
         Debug.Print "        " & prpBucle.Name & " - " & _
            IIf(prpBucle = "", "[vacía]", prpBucle)
         On Error GoTo 0
      Next prpBucle

   Next fldBucle

   ' Elimina el TableDef nuevo porque esto es un ejemplo.
   dbsNeptuno.TableDefs.Delete tdfNuevo.Name
   dbsNeptuno.Close

End Sub
```

Ejemplo del método CreateIndex

Este ejemplo utiliza el método **CreateIndex** para crear dos objetos **Index** nuevos y después los anexa a la colección **Indexes** de la tabla Empleados **objeto TableDef**. Enumera la colección **Indexes** del objeto **TableDef**, la colección **Fields** de los objetos **Index** nuevos y la colección **Properties** de los objetos **Index** nuevos. Se necesita la función CrearÍndiceSalida para ejecutar este procedimiento.

```
Sub CreateIndexX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim idxPaís As Index
   Dim idxNombre As Index
   Dim idxBucle As Index

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno!Empleados

   With tdfEmpleados
      ' Primero crea objeto Index, crea y agrega los
      ' objetos Field al objeto Index y después agrega
      ' el objeto Index a la colección Indexes de
      ' TableDef.
      Set idxPaís = .CreateIndex("ÍndicePaís")
      With idxPaís
         .Fields.Append .CreateField("País")
         .Fields.Append .CreateField("Apellidos")
         .Fields.Append .CreateField("Nombre")
      End With
      .Indexes.Append idxPaís

      ' Crea el segundo objeto Index, crea y agrega
      ' los objetos Field al objeto Index y después
      ' agrega el objeto Index a la colección Indexes
      ' de TableDef.
      Set idxNombre = .CreateIndex
      With idxNombre
         .Name = "ÍndiceNombre"
         .Fields.Append .CreateField("Nombre")
         .Fields.Append .CreateField("Apellidos")
      End With
      .Indexes.Append idxNombre

      ' Actualiza la colección para que pueda tener
      ' acceso a los objetos Index nuevos.
      .Indexes.Refresh

      Debug.Print .Indexes.Count & "Indexes en " & _
         .Name & " TableDef"

      ' Enumera la colección Indexes.
      For Each idxBucle In .Indexes
         Debug.Print "    " & idxBucle.Name
      Next idxBucle
```

```
      ' Imprime un informe.
      CrearÍndiceSalida idxPaís
      CrearÍndiceSalida idxNombre

      ' Elimina los objetos Index nuevos porque esto
      ' es un ejemplo.
      .Indexes.Delete idxPaís.Name
      .Indexes.Delete idxNombre.Name
   End With

   dbsNeptuno.Close

End Sub

Function CrearÍndiceSalida(idxTemp As Index)

   Dim fldBucle As Field
   Dim prpBucle As Property

   With idxTemp
      ' Enumera la colección Fields del objeto Index.
      Debug.Print "Fields en " & .Name
      For Each fldBucle In .Fields
         Debug.Print "    " & fldBucle.Name
      Next fldBucle

      ' Enumera la colección Properties del objeto Index.
      Debug.Print "Properties de " & .Name
      For Each prpBucle In .Properties
         Debug.Print "    " & prpBucle.Name & " - " & _
            IIf(prpBucle = "", "[vacía]", prpBucle)
      Next prpBucle
   End With

End Function
```

Ejemplo del método CreateProperty

Este ejemplo intenta establecer el valor de una propiedad definida por el usuario. Si la propiedad no existe, utiliza el método **CreateProperty** para crearla y establece el valor de la propiedad nueva. Se necesita la función EstablecerPropiedad para ejecutar este procedimiento.

```
Sub CreatePropertyX()

   Dim dbsNeptuno As Database
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Establece la propiedad Archive a True.
   EstablecerPropiedad dbsNeptuno, "Archive", True

   With dbsNeptuno
      Debug.Print "Properties de " & .Name

      ' Enumera la colección Properties de la base de
      ' datos Neptuno.
      For Each prpBucle In .Properties
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
      Next prpBucle

      ' Elimina la propiedad nueva porque esto es un
      ' ejemplo.
      .Properties.Delete "Archive"

      .Close
   End With

End Sub

Sub EstablecerPropiedad(dbsTemp As Database, strNombre As String, _
   booTemp As Boolean)

   Dim prpNueva As Property
   Dim errBucle As Error

   ' Intenta establecer la propiedad especificada.
   On Error GoTo Err_Propiedad
   dbsTemp.Properties("strNombre") = booTemp
   On Error GoTo 0

   Exit Sub

Err_Propiedad:

   ' El error 3270 quiere decir que no se encontró la propiedad.
   If DBEngine.Errors(0).Number = 3270 Then
      ' Crea la propiedad, establece su valor y la
      ' agrega a la colección Properties.
      Set prpNueva = dbsTemp.CreateProperty(strNombre, _
         dbBoolean, booTemp)
      dbsTemp.Properties.Append prpNueva
```

```
      Resume Next
   Else
      ' Si se produce un error diferente, muestra un mensaje.
      For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & vbCr & _
            errBucle.Description
      Next errBucle
      End
   End If

End Sub
```

Ejemplo del método CreateQueryDef

Este ejemplo utiliza el método **CreateQueryDef** para crear y ejecutar un **QueryDef** temporal y permanente. Se necesita la función ObtenerrstTemp para ejecutar este procedimiento.

```
Sub CreateQueryDefX()

   Dim dbsNeptuno As Database
   Dim qdfTemp As QueryDef
   Dim qdfNuevo As QueryDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Crea un QueryDef temporal.
      Set qdfTemp = .CreateQueryDef("", _
         "SELECT * FROM Empleados")
      ' Abre el Recordset e imprime un informe.
      ObtenerrstTemp qdfTemp
      ' Crea un QueryDef permanente.
      Set qdfNuevo = .CreateQueryDef("QueryDefNuevo", _
         "SELECT * FROM Categorías ")
      ' Abre el Recordset e imprime un informe.
      ObtenerrstTemp qdfNuevo
      ' Elimina el QueryDef nuevo porque esto es un ejemplo.
      .QueryDefs.Delete qdfNuevo.Name
      .Close
   End With

End Sub

Function ObtenerrstTemp(qdfTemp As QueryDef)

   Dim rstTemp As Recordset

   With qdfTemp
      Debug.Print .Name
      Debug.Print "    " & .SQL
      ' Abre el Recordset del QueryDef.
      Set rstTemp = .OpenRecordset(dbOpenSnapshot)

      With rstTemp
         ' Llena el Recordset e imprime el número de registros.
         .MoveLast
         Debug.Print "    Número de registros = " & _
            .RecordCount
         Debug.Print
         .Close
      End With

   End With

End Function
```

Ejemplo del método CreateRelation

Este ejemplo utiliza el método **CreateRelation** para crear un **Relation** entre el **TableDef** de Empleados y un **TableDef** nuevo llamado Departamentos. Este ejemplo demuestra también cómo crear un **Relation** nuevo y creará el objeto **Indexes** necesarios en la tabla externa (el **Index** EmpleadosDepartamentos en la tabla Empleados).

```
Sub CreateRelationX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim tdfNuevo As TableDef
   Dim idxNuevo As Index
   Dim relNuevo As Relation
   Dim idxBucle As Index

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Agrega un campo nuevo a la tabla Empleados.
      Set tdfEmpleados = .TableDefs!Empleados
      tdfEmpleados.Fields.Append _
         tdfEmpleados.CreateField("IdDpto", dbInteger, 2)

      ' Crea la tabla Departmentos nueva.
      Set tdfNuevo = .CreateTableDef("Departmentos")

      With tdfNuevo
         ' Crea y agrega los objetos Field a la
         ' colección Fields del objeto TableDef nuevo.
         .Fields.Append .CreateField("IdDpto", dbInteger, 2)
         .Fields.Append .CreateField("NombreDpto", dbText, 20)

         ' Crea el objeto Index en la tabla Departamentos.
         Set idxNuevo = .CreateIndex("ÍndiceIdDpto")
         ' Crea y agrega el objeto Field a la
         ' colección Fields del objeto Index nuevo.
         idxNuevo.Fields.Append idxNuevo.CreateField("IdDpto")
         ' El índice en la tabla principal debe ser
         ' Unique para formar parte de un Relation.
         idxNuevo.Unique = True
         .Indexes.Append idxNuevo
      End With

      .TableDefs.Append tdfNuevo

      ' Crea el objeto Relation EmpleadosDepartamentos, utilizando los
nombres
      ' de las dos tablas en la relación.
      Set relNuevo = .CreateRelation("EmpleadosDepartamentos", _
         tdfNuevo.Name, tdfEmpleados.Name, _
         dbRelationUpdateCascade)

      ' Crea el objeto Field para la colección Fields
      ' del objeto Relation nuevo. Establece las
      ' propiedades Name y ForeignName basadas en los
      ' campos que se van a utilizar en la relación.
```

```
      relNuevo.Fields.Append relNuevo.CreateField("IdDpto")
      relNuevo.Fields!IdDpto.ForeignName = "IdDpto"
      .Relations.Append relNuevo

      ' Imprime un informe.
      Debug.Print "Properties de" & relNuevo.Name & _
         " Relation"
      Debug.Print "    Tabla = " & relNuevo.Table
      Debug.Print "    TablaExterna = " & _
         relNuevo.ForeignTable
      Debug.Print "Fields de " & relNuevo.Name & " Relation"

      With relNuevo.Fields!IdDpto
         Debug.Print "    " & .Name
         Debug.Print "        Nombre = " & .Name
         Debug.Print "        TablaExterna = " & .ForeignName
      End With

      Debug.Print "Indexes en " & tdfEmpleados.Name & _
         " TableDef"
      For Each idxBucle In tdfEmpleados.Indexes
         Debug.Print "    " & idxBucle.Name & _
            ", Foreign = " & idxBucle.Foreign
      Next idxBucle

      ' Elimina los objetos nuevos porque estos es un ejemplo.
      .Relations.Delete relNuevo.Name
      .TableDefs.Delete tdfNuevo.Name
      tdfEmpleados.Fields.Delete "IdDpto"
      .Close
   End With

End Sub
```

Ejemplo del método CreateTableDef

Este ejemplo crea un objeto **TableDef** en la base de datos Neptuno.

```
Sub CreateTableDefX()

   Dim dbsNeptuno As Database
   Dim tdfNuevo As TableDef
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Crea un objeto TableDef nuevo.
   Set tdfNuevo = dbsNeptuno.CreateTableDef("Contactos")

   With tdfNuevo
      ' Crea los campos y los agrega al objeto
      ' TableDef nuevo. Debe hacer esto después de
      ' anexar el objeto TableDef a la colección
      ' TableDefs de la base de datos Neptuno.
      .Fields.Append .CreateField("Nombre", dbText)
      .Fields.Append .CreateField("Apellidos", dbText)
      .Fields.Append .CreateField("Teléfono", dbText)
      .Fields.Append .CreateField("Notas", dbMemo)

      Debug.Print "Propiedades del objeto TableDef nuevo " & _
         "antes de anexarlo a la colección :"

      ' Enumera la colección Properties del objeto
      ' TableDef nuevo.
      For Each prpBucle In .Properties
         On Error Resume Next
         If prpBucle <> "" Then Debug.Print "    " & _
             prpBucle.Name & " = " & prpBucle
         On Error GoTo 0
      Next prpBucle

      ' Anexa el objeto TableDef nuevo a la base de
      ' datos Neptuno.
      dbsNeptuno.TableDefs.Append tdfNuevo

      Debug.Print "Propiedades del objeto TableDef nuevo " & _
         "después de anexarlo a la colección:"

      ' Enumera la colección Properties del objeto
      ' TableDef nuevo.
      For Each prpBucle In .Properties
         On Error Resume Next
         If prpBucle <> "" Then Debug.Print "    " & _
             prpBucle.Name & " = " & prpBucle
         On Error GoTo 0
      Next prpBucle

   End With

   ' Elimina el objeto TableDef nuevo porque esto es
   ' un ejemplo.
```

```
    dbsNeptuno.TableDefs.Delete "Contactos"
    dbsNeptuno.Close
End Sub
```

Ejemplo del método Delete

Este ejemplo utiliza el método **Delete** para quitar un registro especificado de un objeto **Recordset**. Se necesita el procedimiento EliminarRegistro para ejecutar este procedimiento

```
Sub DeleteX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim lngID As Long

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   ' Agrega un registro temporal que se va a eliminar.
   With rstEmpleados
      .Index = "ClavePrincipal"
      .AddNew
      !Nombre = "Daniel"
      !Apellidos = "López Duque"
      .Update
      .Bookmark = .LastModified
      lngID = !IdEmpleado
   End With

   ' Elimina el registro de empleado con el Número de
   ' Id especificado.
   EliminarRegistro rstEmpleados, lngID

   rstEmpleados.Close
   dbsNeptuno.Close

End Sub

Sub EliminarRegistro(rstTemp As Recordset, _
   lngSeek As Long)

   With rstTemp
      .Seek "=", lngSeek
      If .NoMatch Then
         MsgBox "¡No existe el empleado #" & lngSeek & "en el archivo!"
      Else
         .Delete
         MsgBox "¡Registro del empleado #" & lngSeek & _
            "eliminado!"
      End If
   End With

End Sub
```

Ejemplo del método Edit

Este ejemplo utiliza el método **Edit** para reemplazar los datos actuales con el nombre especificado. Se necesita el procedimiento ModificarNombre para ejecutar este procedimiento.

```
Sub EditX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strAntiguoNombre As String
   Dim strAntiguosApellidos As String
   Dim strNombre As String
   Dim strApellidos As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   ' Almacena los datos originales.
   strAntiguoNombre = rstEmpleados!Nombre
   strAntiguosApellidos = rstEmpleados!Apellidos

   ' Obtiene los datos para el registro nuevo.
   strNombre = Trim(InputBox( _
         "Introduzca el nombre:"))
   strApellidos = Trim(InputBox( _
         "Introduzca los apellidos:"))

   ' Se ejecuta si el usuario introduce algo en los dos campos.
   If strNombre <> "" and strApellidos <> "" Then
      ' Actualiza el registro con los datos nuevos.
      ModificarNombre rstEmpleados, strNombre, strApellidos

      With rstEmpleados
         ' Muestra los datos viejos y nuevos.
         Debug.Print "Datos viejos: " & strAntiguoNombre & _
            " " & strAntiguosApellidos
         Debug.Print "Datos nuevos: " & !Nombre & _
            " " & !Apellidos
         ' Restaura los datos originales porque esto
         ' es un ejemplo.
         .Edit
         !Nombre = strAntiguoNombre
         !Apellidos = strAntiguosApellidos
         .Update
      End With

   Else
      Debug.Print _
         "¡Debe escribir una cadena para el nombre y los apellidos!"
   End If

   rstEmpleados.Close
   dbsNeptuno.Close

End Sub
```

```
Sub ModificarNombre(rstTemp As Recordset, _
   strNomrbe As String, strApellidos As String)

   ' Realiza los cambios en el registro y establece el
   ' marcador de posición para mantener el mismo registro actual.
   With rstTemp
      .Edit
      !Nombre = strNomrbe
      !Apellidos = strApellidos
      .Update
      .Bookmark = .LastModified
   End With

End Sub
```

Ejemplo del método Execute

Este ejemplo demuestra el método **Execute** cuando los ejecuta en un objeto **QueryDef** y en un objeto **Database**. Se necesitan los procedimientos EjecutarDefConsulta y ImprimirSalida para ejecutar este procedimiento.

```
Sub ExecuteX()

   Dim dbsNeptuno As Database
   Dim strSQLCambiar As String
   Dim strSQLRestaurar As String
   Dim qdfCambiar As QueryDef
   Dim rstEmpleados As Recordset
   Dim errBucle As Error

   ' Define dos instrucciones SQL para consultas de acciones.
   strSQLCambiar = "UPDATE Empleados SET País = " & _
      "'Estados Unidos' WHERE País = 'EE.UU.'"
   strSQLRestaurar = "UPDATE Empleados SET País = " & _
      "'EE.UU.' WHERE País = 'Estados Unidos'"

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Crea un objeto temporal QueryDef.
   Set qdfCambiar = dbsNeptuno.CreateQueryDef("", _
      strSQLCambiar)
   Set rstEmpleados = dbsNeptuno.OpenRecordset( _
      "SELECT Apellidos, País FROM Empleados", _
      dbOpenForwardOnly)

   ' Imprime un informe de los datos originales.
   Debug.Print _
      "Datos en la tabla Empleados antes de ejecutar la consulta "
   ImprimirSalida rstEmpleados

   ' Ejecuta el QueryDef temporal.
   EjecutarDefConsulta qdfCambiar, rstEmpleados

   ' Imprime un informe de los datos nuevos.
   Debug.Print _
      "Datos en la tabla Empleados después de ejecutar la consulta"
   ImprimirSalida rstEmpleados

   ' Ejecuta una consulta de acciones para restaurar
   ' los datos. Intercepta los errores y si es necesario comprueba la
colección Errors.
   On Error GoTo Err_Ejecutar
   dbsNeptuno.Execute strSQLRestaurar, dbFailOnError
   On Error GoTo 0

   ' Recupera los datos actuales volviendo a consultar el Recordset.
   rstEmpleados.Requery

   ' Imprime un informe de los datos restaurados.
   Debug.Print "Datos después de ejecutar la consulta " & _
      "para restaurar la información original "
   ImprimirSalida rstEmpleados
```

```
   rstEmpleados.Close

   Exit Sub

Err_Ejecutar:

   ' Notifica al usuario cualquier error resultante de
   ' la consulta.
   If DBEngine.Errors.Count > 0 Then
      For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & vbCr & _
            errBucle.Description
      Next errBucle
   End If

   Resume Next

End Sub

Sub EjecutarDefConsulta(qdfTemp As QueryDef, _
   rstTemp As Recordset)

   Dim errBucle As Error

   ' Ejecuta el objeto QueryDef especificado.
   ' Intercepta los errores y si es necesario comprueba la colección
Errors.
   On Error GoTo Err_Ejecutar
   qdfTemp.Execute dbFailOnError
   On Error GoTo 0

   ' Recupera los datos actuales volviendo a consultar el Recordset.
   rstTemp.Requery

   Exit Sub

Err_Ejecutar:

      ' Notifica al usuario cualquier error resultante
      ' de la consulta.
   If DBEngine.Errors.Count > 0 Then
      For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & vbCr & _
            errBucle.Description
      Next errBucle
   End If

   Resume Next

End Sub

Sub ImprimirSalida(rstTemp As Recordset)

   ' Enumera el Recordset.
   Do While Not rstTemp.EOF
      Debug.Print "    " & rstTemp!Apellidos & _
         ", " & rstTemp!País
```

```
      rstTemp.MoveNext
   Loop

End Sub
```

Ejemplo de los métodos FindFirst, FindLast, FindNext y FindPrevious

Este ejemplo utiliza los métodos **FindFirst**, **FindLast**, **FindNext** y **FindPrevious** para mover el puntero del registro de un objeto **Recordset** basado en una cadena de búsqueda suministrada y un comando. Se necesita la función EncontrarCualquiera para ejecutar este procedimiento.

```
Sub FindFirstX()

   Dim dbsNeptuno As Database
   Dim rstClientes As Recordset
   Dim strPaís As String
   Dim varMarcador As Variant
   Dim strMensaje As String
   Dim intComando As Integer

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstClientes = dbsNeptuno.OpenRecordset( _
      "SELECT NombreCompañía, Ciudad, País " & _
      "FROM Clientes ORDER BY NombreCompañía", _
      dbOpenSnapshot)

   Do While True
      ' Obtiene una entrada del usuario y construye la cadena de búsqueda.
      strPaís = _
         Trim(InputBox("Introduzca el país a buscar."))
      If strPaís = "" Then Exit Do
      strPaís = "País = '" & strPaís & "'"

      With rstClientes
         ' Llena el Recordset.
         .MoveLast
         ' Encuentra el primer registro que coincide
         ' con la cadena de búsqueda. Sale del bucle si no existe ningún
registro.
         .FindFirst strPaís
         If .NoMatch Then
            MsgBox "No se encontraron registros con " & _
               strPaís & "."
            Exit Do
         End If

         Do While True
            ' Almacena el marcador de posición del registro actual.
            varMarcador = .Bookmark
            ' Obtiene la elección del usuario del método a utilizar.
            strMensaje = "Compañía: " & !NombreCompañía & _
               vbCr & "Ciudad: " & !Ciudad & ", " & _
               !País & vbCr & vbCr & _
               strPaís & vbCr & vbCr & _
               "[1 - Primero, 2 - Último, " & _
               vbCr & "3 - Siguiente, " & _
               "4 - Anterior]"
            intComando = Val(Left(InputBox(strMensaje), 1))
            If intComando < 1 Or intComando > 4 Then Exit Do

            ' Utiliza el método Find seleccionado. Si
            ' falla Find, vuelve al último registro actual.
```

```
            If EncontrarCualquiera(intComando, rstClientes, _
                  strPaís) = False Then
               .Bookmark = varMarcador
               MsgBox "No hay coincidencias -volviendo al " & _
                  "registro actual."
            End If

         Loop

      End With

      Exit Do
   Loop

   rstClientes.Close
   dbsNeptuno.Close

End Sub

Function EncontrarCualquiera(intChoice As Integer, _
   rstTemp As Recordset, _
   strEncontrar As String) As Boolean

   ' Utiliza el método Find basado en la entrada del usuario.
   Select Case intChoice
      Case 1
         rstTemp.FindFirst strEncontrar
      Case 2
         rstTemp.FindLast strEncontrar
      Case 3
         rstTemp.FindNext strEncontrar
      Case 4
         rstTemp.FindPrevious strEncontrar
   End Select

   ' Establece el valor devuleto basado en la propiedad NoMatch.
   EncontrarCualquiera = IIf(rstTemp.NoMatch, False, True)

End Function
```

Ejemplo del método GetRows

Este ejemplo utiliza el método **GetRows** para recuperar un número de filas especificado de un objeto **Recordset** y llena una matriz con los datos resultantes. El método **GetRows** devolverá un número de filas menor que el deseado en dos casos: si se ha alcanzado el **EOF** o si **GetRows** intenta recuperar un registro que eliminó otro usuario. La función devuelve **False** sólo si se produce el segundo caso. Se necesita la función ObtenerFilasCorrecto para ejecutar este procedimiento.

```
Sub GetRowsX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strMensaje As String
   Dim intFilas As Integer
   Dim mvarRegistros As Variant
   Dim intRegistro As Integer

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = dbsNeptuno.OpenRecordset( _
      "SELECT Nombre, Apellidos, Cargo " & _
      "FROM Empleados ORDER BY Apellidos", dbOpenSnapshot)

   With rstEmpleados
      Do While True
         ' Obtiene una entrada del usuario para el número de filas.
         strMensaje = "Introduzca el número de filas a recuperar."
         intFilas = Val(InputBox(strMensaje))

         If intFilas <= 0 Then Exit Do

         ' Si ObtenerFilasCorrecto tiene éxito, imprime el
         ' resultado y nada si se alcanza el fin del archivo.
         If ObtenerFilasCorrecto(rstEmpleados, intFilas, _
               mvarRegistros) Then
            If intFilas > UBound(mvarRegistros, 2) + 1 Then
               Debug.Print "(No hay registros suficientes en " & _
                  "Recordset a recuperar " & intFilas & _
                  " filas .)"
            End If
            Debug.Print UBound(mvarRegistros, 2) + 1 & _
               " registros encontrados."

            ' Imprime los datos recuperados.
            For intRegistro = 0 To UBound(mvarRegistros, 2)
               Debug.Print "    " & _
                  mvarRegistros(0, intRegistro) & " " & _
                  mvarRegistros(1, intRegistro) & ", " & _
                  mvarRegistros(2, intRegistro)
            Next intRegistro
         Else
            ' Suponiendo que el error de GetRows se originó por
            ' los cambios realizados por otro usuario, utilice Requery
            ' para actualizar el Recordset y volver a empezar.
            If .Restartable Then
               If MsgBox("Falló GetRows - ¿Se vuelve a intentar?", _
                     vbYesNo) = vbYes Then
```

```
                .Requery
             Else
                Debug.Print "¡Falló GetRows!"
                Exit Do
             End If
          Else
             Debug.Print "¡Falló GetRows! " & _
                "¡El Recordset no se puede reiniciar!"
             Exit Do
          End If
       End If

       ' Como el uso de GetRows deja el puntero del registro actual
       ' en el último registro al que se tuvo acceso, mueva el
       ' puntero hacia atrás al principio del Recordset antes de
       ' volver a realizar otra búsqueda.
       .MoveFirst
    Loop
 End With

 rstEmpleados.Close
 dbsNeptuno.Close

End Sub

Function ObtenerFilasCorrecto(rstTemp As Recordset, _
 intNumber As Integer, avarData As Variant) As Boolean

 ' Almacena el resultado del método GetRows en una matriz.
 avarData = rstTemp.GetRows(intNumber)
 ' Devuelve False sólo si se devuelven menos
 ' registros de los deseados, pero no lo hace porque
 ' encuentra el final del Recordset.
 If intNumber > UBound(avarData, 2) + 1 And _
       Not rstTemp.EOF Then
    ObtenerFilasCorrecto = False
 Else
    ObtenerFilasCorrecto = True
 End If

End Function
```

Ejemplo del método Move

Este ejemplo utiliza el método **Move** para posicionar el puntero del registro basado en una entrada del usuario.

```
Sub MoveX()

   Dim dbsNeptuno As Database
   Dim rstProveedores As Recordset
   Dim varMarcador As Variant
   Dim strComando As String
   Dim lngMover As Long

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstProveedores = _
      dbsNeptuno.OpenRecordset("SELECT NombreCompañía, " & _
      "Ciudad, País FROM Proveedores ORDER BY NombreCompañía", _
      dbOpenDynaset)

   With rstProveedores
      ' Llena el Recordset.
      .MoveLast
      .MoveFirst

      Do While True
         ' Muestra la información acerca del registro
         ' actual y pregunta al usuario cuántos registros desea mover.
         strComando = InputBox( _
            "Registro " & (.AbsolutePosition + 1) & " de " & _
            .RecordCount & vbCr & "Compañía: " & _
            !NombreCompañía & vbCr & "Ciudad: " & !Ciudad & _
            ", " & !País & vbCr & vbCr & _
            "Introduzca el número de registros a mover " & _
            "(positivo o negativo).")

         If strComando = "" Then Exit Do

         ' Almacena el marcador de posición en caso de que no funciones
Move.
         varMarcador = .Bookmark

         ' El método Move necesita el parámetro del tipo de datos Long.
         lngMover = CLng(strComando)
         .Move lngMover

         ' Intercepta el BOF o el EOF.
         If .BOF Then
            MsgBox "¡Demasiado hacia atrás! " & _
               "Volviendo al registro actual."
            .Bookmark = varMarcador
         End If
         If .EOF Then
            MsgBox "¡Demasiado hacia adelante! " & _
               "Volviendo al registro actual."
            .Bookmark = varMarcador
         End If
      Loop
```

```
        .Close
    End With

    dbsNeptuno.Close

End Sub
```

Ejemplo de los métodos MoveFirst, MoveLast, MoveNext y MovePrevious

Este ejemplo utiliza los métodos **MoveFirst**, **MoveLast**, **MoveNext** y **MovePrevious** para mover el puntero del registro de un objeto **Recordset** basado en el comando suministrado. Se necesita el procedimiento MoverCualquiera para ejecutar este procedimiento.

```
Sub MoveFirstX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strMensaje As String
   Dim intComando As Integer

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = dbsNeptuno.OpenRecordset( _
      "SELECT Nombre, Apellidos FROM Empleados " & _
      "ORDER BY Apellidos", dbOpenSnapshot)

   With rstEmpleados
      ' Llena el Recordset.
      .MoveLast
      .MoveFirst
      Do While True
         ' Muestra información del registro actual y
         ' obtiene la elección del método del usuario.
         strMensaje = "Nombre: " & !Nombre & " " & _
            !Apellidos & vbCr & "Registro " & _
            (.AbsolutePosition + 1) & " de " & _
            .RecordCount & vbCr & vbCr & _
            "[1 - Primero, 2 - Último, " & vbCr & _
            "3 - Siguiente, 4 - Anterior]"
         intComando = Val(Left(InputBox(strMensaje), 1))
         If intComando < 1 Or intComando > 4 Then Exit Do

         ' Llama al método basado en la entrada del usuario.
         MoverCualquiera intComando, rstEmpleados
      Loop
      .Close
   End With

   dbsNeptuno.Close

End Sub

Sub MoverCualquiera(intChoice As Integer, _
   rstTemp As Recordset)

   ' Utiliza el método especificado, intercepta el BOF y el EOF.
   With rstTemp
      Select Case intChoice
         Case 1
            .MoveFirst
         Case 2
            .MoveLast
         Case 3
            .MoveNext
            If .EOF Then
```

```
                MsgBox "¡Se alcanzó el final del Recordset!"
                .MoveLast
            End If
         Case 4
            .MovePrevious
            If .BOF Then
                MsgBox "¡Se alcanzó el principio del Recordset!"
                .MoveFirst
            End If
      End Select
   End With

End Sub
```

Ejemplo del método NewPassword

Este ejemplo pregunta al usuario una contraseña nueva para el usuario Paula Pérez. Si la entrada es una cadena de caracteres entre 1 y 14 de longitud, el ejemplo utiliza el método **NewPassword** para cambiar la contraseña. El usuario se debe conectar como Paula Pérez o debe ser un miembro del grupo Administradores.

```
Sub NewPasswordX()

   Dim wrkPredeterminado As Workspace
   Dim usrNuevo As User
   Dim grpNuevo As Group
   Dim grpMiembro As Group
   Dim strContraseña As String

   ' Obtiene el espacio de trabajo predeterminado.
   Set wrkPredeterminado = DBEngine.Workspaces(0)

   With wrkPredeterminado

      ' Crea y agrega un usuario nuevo.
      Set usrNuevo = .CreateUser("Paula Pérez", _
         "abc123DEF456", "Contraseña1")
      .Users.Append usrNuevo

      ' Crea y agrega un grupo nuevo.
      Set grpNuevo = .CreateGroup("Cuentas", _
         "UVW987xyz654")
      .Groups.Append grpNuevo

      ' Hace al usuario nuevo miembro del grupo nuevo.
      Set grpMiembro = usrNuevo.CreateGroup("Cuentas")
      usrNuevo.Groups.Append grpMiembro

      ' Pregunta al usuario una contraseña nueva. Si
      ' la entrada es demasiado larga, vuelve a preguntar.
      Do While True
         strContraseña = InputBox("Introduzca la contraseña nueva:")
         Select Case Len(strContraseña)
            Case 1 To 14
               usrNuevo.NewPassword "Contraseña1", strContraseña
               MsgBox "¡Contraseña cambiada!"
               Exit Do
            Case Is > 14
               MsgBox "¡Contraseña demasiado larga!"
            Case 0
               Exit Do
         End Select
      Loop

      ' Elimina los objetos User y Group nuevos porque
      ' esto es un ejemplo.
      .Users.Delete "Paula Pérez
      .Groups.Delete "Cuentas"

   End With
```

```
End Sub
```

Ejemplo del método Refresh

Este ejemplo utiliza el método **Refresh** para actualizar la colección **Fields** de la tabla Categorías basada en los cambios de los datos de **OrdinalPosition**. El orden de **Fields** en la colección cambia sólo después de utilizar el método **Refresh**.

```
Sub RefreshX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim mintPosición() As Integer
   Dim mstrNombreCampo() As String
   Dim intTemp As Integer
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno.TableDefs("Categorías ")

   With tdfEmpleados
      ' Muestra los datos de OrdinalPosition
      ' originales y los almacena en una matriz.
      Debug.Print _
         "Datos originales de OrdinalPosition en TableDef."
      ReDim mintPosición(0 To .Fields.Count - 1) As Integer
      ReDim mstrNombreCampo(0 To .Fields.Count - 1) As String
      For intTemp = 0 To .Fields.Count - 1
         mintPosición(intTemp) = _
            .Fields(intTemp).OrdinalPosition
         mstrNombreCampo(intTemp) = .Fields(intTemp).Name
         Debug.Print , mintPosición(intTemp), _
            mstrNombreCampo(intTemp)
      Next intTemp

      ' Cambia los datos de OrdinalPosition.
      For Each fldBucle In .Fields
         fldBucle.OrdinalPosition = _
            100 - fldBucle.OrdinalPosition
      Next fldBucle
      Set fldBucle = Nothing

      ' Imprime los datos nuevos.
      Debug.Print "Datos de OrdinalPosition nuevos antes de Refresh."
      For Each fldBucle In .Fields
         Debug.Print , fldBucle.OrdinalPosition, fldBucle.Name
      Next fldBucle

      .Fields.Refresh

      ' Imprime los datos nuevos, mostrando cómo se ha
      ' cambiado el orden de los campos.
      Debug.Print " Datos de OrdinalPosition nuevos después de Refresh."
      For Each fldBucle In .Fields
         Debug.Print , fldBucle.OrdinalPosition, fldBucle.Name
      Next fldBucle

      ' Restaura los datos originales de OrdinalPosition.
      For intTemp = 0 To .Fields.Count - 1
```

```
            .Fields(mstrNombreCampo(intTemp)).OrdinalPosition = _
               mintPosición(intTemp)
         Next intTemp
      End With

      dbsNeptuno.Close

   End Sub
```

Ejemplo del método RepairDatabase

Este ejemplo intenta reparar la base de datos llamada Neptuno.mdb. No puede ejecutar este procedimiento desde un módulo de Neptuno.mdb.

```
Sub RepairDatabaseX()

   Dim errBucle As Error

   If MsgBox("¿Desea reparar la base de datos Neptuno?", _
         vbYesNo) = vbYes Then
      On Error GoTo Err_Reparar
      DBEngine.RepairDatabase "Neptuno.mdb"
      On Error GoTo 0
      MsgBox "¡Fin del procedimiento reparar!"
   End If

   Exit Sub

Err_Reparar:

   For Each errBucle In DBEngine.Errors
      MsgBox "¡Falló Repair!" & vbCr & _
         "Número de error: " & errBucle.Number & _
         vbCr & errBucle.Description
   Next errBucle

End Sub
```

Ejemplo del método Seek

Este ejemplo demuestra el método **Seek** permitiendo al usuario buscar un producto basado en un número de identificación.

```
Sub SeekX()

   Dim dbsNeptuno As Database
   Dim rstProductos As Recordset
   Dim intPrimero As Integer
   Dim intÚltimo As Integer
   Dim strMensaje As String
   Dim strBuscar As String
   Dim varMarcador As Variant

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Debe abrir un Recordset de tipo Table para
   ' utilizar un índice y el método Seek.
   Set rstProductos = _
      dbsNeptuno.OpenRecordset("Productos", dbOpenTable)

   With rstProductos
      ' Establece el índice.
      .Index = "ClavePrincipal"

      ' Obtiene el mayor y el menor Id de producto.
      .MoveLast
      intÚltimo = !IdProducto
      .MoveFirst
      intPrimero = !IdProducto

      Do While True
         ' Muestra la información del registro actual
         ' y pregunta el usuario un número de Id.
         strMensaje = "Id de producto: " & !IdProducto & vbCr & _
            "Nombre: " & !NombreProducto & vbCr & vbCr & _
            "Introduzca un Id de producto entre " & intPrimero & _
            " y " & intÚltimo & "."
         strBuscar = InputBox(strMensaje)

         If strBuscar = "" Then Exit Do

         ' Almacena el marcador de posición actual si falla Seek.
         varMarcador = .Bookmark

         .Seek "=", Val(strBuscar)

         ' Vuelve al registro actual si falla Seek.
         If .NoMatch Then
            MsgBox "¡No se encontró Id.!"
            .Bookmark = varMarcador
         End If
      Loop

      .Close
   End With
```

```
    dbsNeptuno.Close

End Sub
```

Ejemplo del método Update

Este ejemplo demuestra el método **Update** en unión con el método **Edit**.

```
Sub UpdateX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strAntiguoNombre As String
   Dim strAntiguosApellidos As String
   Dim strMensaje As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      .Edit
      ' Almacena los datos originales.
      strAntiguoNombre = !Nombre
      strAntiguosApellidos = !Apellidos
      ' Cambia los datos en el búfer de modificación.
      !Nombre = "María"
      !Apellidos = "Álvarez"

      ' Muestra el contenido del búfer y obtiene una entrada del usuario.
      strMensaje = "Modificación en progreso:" & vbCr & _
         "    Datos originales = " & strAntiguoNombre & " " & _
         strAntiguosApellidos & vbCr & "    Datos en el búfer = " & _
         !Nombre & " " & !Apellidos & vbCr & vbCr & _
         "¿Utilizar Update para reemplazar los datos originales con " & _
         "los datos del búfer en el Recordset?"

      If MsgBox(strMensaje, vbYesNo) = vbYes Then
         .Update
      Else
         .CancelUpdate
      End If

      ' Muestra los datos resultantes.
      MsgBox "Datos en el Recordset = " & !Nombre & " " & _
         !Apellidos

      ' Restaura los datos originales porque esto es un ejemplo.
      If Not (strAntiguoNombre = !Nombre And _
            strAntiguosApellidos = !Apellidos) Then
         .Edit
         !Nombre = strAntiguoNombre
         !Apellidos = strAntiguosApellidos
         .Update
      End If

      .Close
   End With

   dbsNeptuno.Close
```

```
End Sub
```

Este ejemplo demuestra el método **Update** en unión con el método **AddNew**.

```
Sub UpdateX2()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strAntiguoNombre As String
   Dim strAntiguosApellidos As String
   Dim strMensaje As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      .AddNew
      !Nombre = "Juan"
      !Apellidos = "Fernández"

      ' Muestra el contenido del búfer y obtiene una entrada del usuario.
      strMensaje = "AddNew en progreso:" & vbCr & _
         "    Datos en el búfer = " & !Nombre & " " & _
         !Apellidos & vbCr & vbCr & _
         "¿Utilizar Update para guardar el búfer en el Recordset?"

      If MsgBox(strMensaje, vbYesNoCancel) = vbYes Then
         .Update
         ' Va al registro nuevo y Muestra los datos resultantes.
         .Bookmark = .LastModified
         MsgBox "Datos en el Recordset = " & !Nombre & _
            " " & !Apellidos
         ' Elimina los datos nuevos porque esto es un ejemplo.
         .Delete
      Else
         .CancelUpdate
         MsgBox "No se agregó el registro nuevo."
      End If

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo del método CancelUpdate

Este ejemplo muestra cómo se utiliza el método **CancelUpdate** con el método **AddNew**.

```
Sub CancelUpdateX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim intComando As Integer

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = dbsNeptuno.OpenRecordset( _
      "Empleados", dbOpenDynaset)

   With rstEmpleados
      .AddNew
      !Nombre = "Carolina"
      !Apellidos = "Alonso"
      intComando = MsgBox("Agregar un nuevo registro para " & _
         !Nombre & " " & !Apellidos & "?", vbYesNo)
      If intComando = vbYes Then
         .Update
         MsgBox "Registro agregado."
         ' Elimina el nuevo registro porque esto es un
         ' ejemplo.
         .Bookmark = .LastModified
         .Delete
      Else
         .CancelUpdate
         MsgBox "Registro no agregado."
      End If
   End With

   dbsNeptuno.Close

End Sub
```

Este ejemplo muestra cómo se utiliza el método **CancelUpdate** con el método **Edit**.

```
Sub CancelUpdateX2()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strPrimero As String
   Dim strÚltimo As String
   Dim intComando As Integer

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   With rstEmpleados
      strPrimero = !Nombre
      strÚltimo = !Apellidos
      .Edit
      !Nombre = "Diana"
      !Apellidos = "Sainz"
      intComando = MsgBox("Reemplazar el nombre actual por " & _
```

```
            !Nombre & " " & !Apellidos & "?", vbYesNo)
         If intComando = vbYes Then
            .Update
            MsgBox "Registro modificado."
            ' Recupera datos porque esto es un ejemplo.
            .Bookmark = .LastModified
            .Edit
            !Nombre = strPrimero
            !Apellidos = strÚltimo
            .Update
         Else
            .CancelUpdate
            MsgBox " Registro no modificado."
         End If
         .Close
      End With

      dbsNeptuno.Close

End Sub
```

Ejemplo del método CopyQueryDef

Este ejemplo utiliza el método **CopyQueryDef** para crear una copia de un objeto **QueryDef** de un objeto **Recordset** existente y modifica la copia agregando una cláusula a la propiedad **SQL**. Cuando crea un objeto **QueryDef** permanente**,** los espacios en blanco, puntos y comas o avances de línea se pueden agregar a la propiedad **SQL**; estos caracteres adicionales se deben quitar antes de adjuntar cláusulas nuevas a la instrucción SQL.

```
Function CopiarNuevaConsulta(rstTemp As Recordset, _
   strAgregar As String) As QueryDef

   Dim strSQL As String
   Dim strDerechaSQL As String

   Set CopiarNuevaConsulta = rstTemp.CopyQueryDef
   With CopiarNuevaConsulta
      ' Quita caracteres adicionales.
      strSQL = .SQL
      strDerechaSQL = Right(strSQL, 1)
      Do While strDerechaSQL = " " Or strDerechaSQL = ";" Or _
            strDerechaSQL = Chr(10) Or strDerechaSQL = vbCr
         strSQL = Left(strSQL, Len(strSQL) - 1)
         strDerechaSQL = Right(strSQL, 1)
      Loop
      .SQL = strSQL & strAgregar
   End With

End Function
```

Este ejemplo muestra un posible uso de CopiarNuevaConsulta().

```
Sub CopiarDefConsulta()

   Dim dbsNeptuno As Database
   Dim qdfEmpleados As QueryDef
   Dim rstEmpleados As Recordset
   Dim intComando As Integer
   Dim strOrdenarPor As String
   Dim qdfCopiar As QueryDef
   Dim rstCopiar As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set qdfEmpleados = dbsNeptuno.CreateQueryDef("NuevoQueryDef", _
      "SELECT Nombre, Apellidos, FechaNacimiento FROM _
      Empleados")
   Set rstEmpleados = qdfEmpleados.OpenRecordset(dbOpenSnapshot, _
      dbForwardOnly)

   Do While True
      intComando = Val(InputBox( _
         "Elija el campo por el que ordenar un nuevo_
         "Recordset:" & vbCr & "1 - Nombre" & vbCr &_
         "2 - Apellidos" & vbCr & "3 - Fecha de _
         "nacimiento" & vbCr & "[Cancelar - salir]"))
      Select Case intComando
         Case 1
            strOrdenarPor = " ORDER BY Nombre"
```

```
         Case 2
            strOrdenarPor = " ORDER BY Apellidos"
         Case 3
            strOrdenarPor = " ORDER BY FechaNacimiento"
         Case Else
            Exit Do
      End Select
      Set qdfCopiar = CopiarNuevaConsulta(rstEmpleados, strOrdenarPor)
      Set rstCopiar = qdfCopiar.OpenRecordset(dbOpenSnapshot, _
         dbForwardOnly)
      With rstCopiar
         Do While Not .EOF
            Debug.Print !Apellidos & ", " & !Nombre & " - " & _
               !FechaNacimiento
            .MoveNext
         Loop
         .Close
      End With
      Exit Do
   Loop

   rstEmpleados.Close
   ' Elimina el nuevo QueryDef porque esto es un ejemplo.
   dbsNeptuno.QueryDefs.Delete qdfEmpleados.Name
   dbsNeptuno.Close

End Sub
```

Ejemplo del método CreateGroup

Este ejemplo utiliza el método **CreateGroup** para crear un nuevo objeto **Group**; después hace al usuario "admin" miembro del nuevo objeto **Group** y enumera sus propiedades y usuarios.

```
Sub CreateGroupX()

   Dim wrkPredeterminado As Workspace
   Dim grpNuevo As Group
   Dim grpTemp As Group
   Dim prpBucle As Property
   Dim usrBucle As User

   Set wrkPredeterminado = DBEngine.Workspaces(0)

   With wrkPredeterminado

      ' Crea y agrega el nuevo grupo.
      Set grpNuevo = .CreateGroup("NuevoGrupo", _
         "AAA123456789")
      .Groups.Append grpNuevo

      ' Hace al usuario "admin" miembro del
      ' grupo NuevoGrupo creando y agregando el
      ' objeto Group adecuado a la colección Groups
      ' del usuario.
      Set grpTemp = .Users("admin").CreateGroup("NuevoGrupo")
      .Users("admin").Groups.Append grpTemp

      Debug.Print "Propiedades de" & grpNuevo.Name

      ' Enumera la colección Properties de NuevoGrupo. La
      ' propiedad PID no es legible.
      For Each prpBucle In grpNuevo.Properties
         On Error Resume Next
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
         On Error GoTo 0
      Next prpBucle

      Debug.Print "Colección Users de " & grpNuevo.Name

      ' Enumera la colección Users de NuevoGrupo.
      For Each usrBucle In grpNuevo.Users
         Debug.Print "    " & _
            usrBucle.Name
      Next usrBucle

      ' Elimina el objeto Group porque esto
      ' es un ejemplo.
      .Groups.Delete "NuevoGrupo"

   End With

End Sub
```

Ejemplo del método CreateUser y de las propiedades Password y PID

Este ejemplo utiliza el método **CreateUser** y las propiedades **Password** y **PID** para crear un nuevo objeto **User**; que hace el nuevo objeto **User** miembro de objetos **Group** diferentes y enumera sus propiedades y grupos.

```
Sub CreateUserX()

   Dim wrkPredeterminado As Workspace
   Dim usrNuevo As User
   Dim grpNuevo As Group
   Dim usrTemp As User
   Dim prpBucle As Property
   Dim grpBucle As Group

   Set wrkPredeterminado = DBEngine.Workspaces(0)

   With wrkPredeterminado

      ' Crea y agrega un User nuevo.
      Set usrNuevo = .CreateUser("NuevoUsuario")
      usrNuevo.PID = "AAA123456789"
      usrNuevo.Password = "NuevaContraseña"
      .Users.Append usrNuevo

      ' Crea y agrega un Group nuevo.
      Set grpNuevo = .CreateGroup("NuevoGrupo", _
         "AAA123456789")
      .Groups.Append grpNuevo

      ' Hace al usuario "NuevoUsuario" miembro del
      ' grupo "NuevoGrupo" creando y agregando el objeto
      ' User adecuado para la colección Users
      ' del grupo.
      Set usrTemp = _
         .Groups("NuevoGrupo").CreateUser("NuevoUsuario")
      .Groups("NuevoGrupo").Users.Append usrTemp

      Debug.Print "Propiedades de" & usrNuevo.Name

      ' Enumera la colección Properties de NuevoUsuario. La
      ' propiedad PID no es legible.
      For Each prpBucle In usrNuevo.Properties
         On Error Resume Next
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
         On Error GoTo 0
      Next prpBucle

      Debug.Print "Colección Groups de " & usrNuevo.Name

      ' Enumera la colección Groups de NuevoUsuario.
      For Each grpBucle In usrNuevo.Groups
         Debug.Print "    " & _
            grpBucle.Name
      Next grpBucle
```

```
      ' Elimina los objetos User y Group nuevos porque esto
      ' es un ejemplo.
      .Users.Delete "NuevoUsuario"
      .Groups.Delete "NuevoGrupo"

   End With

End Sub
```

Ejemplo del método CreateWorkspace

Este ejemplo utiliza el método **CreateWorkspace** para crear un espacio de trabajo Microsoft Jet y un espacio de trabajo ODBCDirect. Después enumera las propiedades de los dos tipos de espacios de trabajo.

```
Sub CreateWorkspaceX()

   Dim wrkODBC As Workspace
   Dim wrkJet As Workspace
   Dim wrkBucle As Workspace
   Dim prpBucle As Property

   ' Crea un espacio de trabajo ODBCDirect. Hasta que cree
   ' espacio de trabajo Microsoft Jet, el motor de base de datos
   ' Microsoft Jet no se cargará en memoria.
   Set wrkODBC = CreateWorkspace("ODBCWorkspace", "admin", _
      "", dbUseODBC)
   Workspaces.Append wrkODBC

   DefaultType = dbUseJet
   ' Crea un objeto Workspace sin nombre del tipo
   ' especificado por la propiedad DefaultType de DBEngine
   ' (dbUseJet).
   Set wrkJet = CreateWorkspace("", "admin", "")

   ' Enumera la colección Workspaces.
   Debug.Print "Objetos Workspace en la colección Workspaces:"
   For Each wrkBucle In Workspaces
      Debug.Print "    " & wrkBucle.Name
   Next wrkBucle

   With wrkODBC
      ' Enumera la colección Properties del espacio de
      ' trabajo ODBCDirect.
      Debug.Print "Propiedades de" & .Name
      On Error Resume Next
      For Each prpBucle In .Properties
         Debug.Print "    " & prpBucle.Name & " = " & prpBucle
      Next prpBucle
      On Error GoTo 0
   End With

   With wrkJet
      ' Enumera la colección Properties del espacio de
      ' trabajo Microsoft Jet.
      Debug.Print _
         "Propiedades del espacio de trabajo Microsoft Jet sin nombre "
      On Error Resume Next
      For Each prpBucle In .Properties
         Debug.Print "    " & prpBucle.Name & " = " & prpBucle
      Next prpBucle
      On Error GoTo 0
   End With

   wrkODBC.Close
   wrkJet.Close
```

```
End Sub
```

Ejemplo del método Idle

Este ejemplo utiliza el método **Idle** para asegurar que un procedimiento de salida recupera los datos disponibles más recientes de la base de datos. El procedimiento SalidaIdle se necesita para ejecutar este procedimiento.

```
Sub IdleX()

   Dim dbsNeptuno As Database
   Dim strPaís As String
   Dim strSQL As String
   Dim rstPedidos As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Obtiene el nombre del país del usuario y crea una
   ' instrucción SQL con él.
   strPaís = Trim(InputBox("Escriba el país:"))
   strSQL = "SELECT * FROM Pedidos WHERE PaísDestinatario = '" & _
      strPaís & "' ORDER BY IdPedido"

   ' Abre el objeto Recordset con la instrucción SQL.
   Set rstPedidos = dbsNeptuno.OpenRecordset(strSQL)

   ' Muestra el contenido del objeto Recordset.
   SalidaIdle rstPedidos, strPaís

   rstPedidos.Close
   dbsNeptuno.Close

End Sub

Sub SalidaIdle(rstTemp As Recordset, strTemp As String)

   ' Llama al método Idle para liberar bloqueos no necesarios, obligar
   ' escrituras pendientes y actualizar la memoria con los datos
   ' actuales en el archivo .mdb.
   DBEngine.Idle (dbRefreshCache)

   ' Enumera el objeto Recordset.
   With rstTemp
      Debug.Print "Pedidos de " & strTemp & ":"
      Debug.Print , "IdPedido", "IdCliente", "FechaPedido"
      Do While Not .EOF
         Debug.Print , !IdPedido, !IdCliente, !FechaPedido
         .MoveNext
      Loop
   End With

End Sub
```

Ejemplo del método MakeReplica

Esta función utiliza el método **MakeReplica** para crear una replica adicional de un Diseño Principal. El argumento `intOpciones` puede ser una combinación de las constantes **dbCrearRépSóloLectura** y **dbCrearRépParcial**, o puede ser 0. Por ejemplo, para crear una réplica parcial de sólo lectura, debe transferir el valor `dbCrearRépSóloLectura + dbCrearRépParcial` como el valor de `intOpciones`.

```
Function CrearRéplicaAdicional(strBDReplicable As _
   String, strNuevaRéplica  As String, intOpciones As _
   Integer) As Integer

   Dim dbs As Database
   On Error Goto ErrorHandler

   Set dbs = OpenDatabase(strBDReplicable)

   ' Si no se transfieren opciones a
   ' CrearRéplicaAdicional, omita el
   ' argumento opciones, que de manera predeterminada
   ' crea una réplica completa de lectura/escritura.
   ' Por tanto, utilice el valor de intOpciones.

   If intOpciones = 0 Then
      dbs.CrearRéplica strNuevaRéplica , _
         "Réplica de " & strBDReplicable
   Else
      dbs.CrearRéplica strNuevaRéplica , _
         "Réplica de " & strBDReplicable, _
         intOpciones
   End If

   dbsCrearRéplica.Close

ErrorHandler:
   Select Case Err
      Case 0:
         CrearRéplicaAdicional = 0
         Exit Function
      Case Else
         MsgBox "Error " & Err & " : " & Error
         CrearRéplicaAdicional = Err
         Exit Function
   End Select

End Function
```

Ejemplo del método NextRecordset y de la propiedad DefaultCursorDriver

Este ejemplo utiliza el método **NextRecordset** para ver datos de una consulta SELECT compuesta. La propiedad **DefaultCursorDriver** se debe establecer a **dbUseODBCCursor** cuando ejecuta estas consultas. El método **NextRecordset** devolverá **True** incluso si alguna o todas las instrucciones SELECT no devuelven registros - devolverá **False** sólo después de que se hayan comprobado todas las cláusula SQL individuales.

```
Sub NextRecordsetX()

   Dim wrkODBC As Workspace
   Dim conEditores As Connection
   Dim rstTemp As Recordset
   Dim intContador As Integer
   Dim booSiguiente As Boolean

   ' Crea un objeto Workspace ODBCDirect y abre el objeto
   ' Connection. Se necesita el valor DefaultCursorDriver
   ' cuando utiliza instrucciones SQL compuestas.
   Set wrkODBC = CreateWorkspace("", _
      "admin", "", dbUseODBC)
   wrkODBC.DefaultCursorDriver = dbUseODBCCursor
   Set conEditores = wrkODBC.OpenConnection("Editores", , , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Genera la instrucción SELECT compuesta.
   Set rstTemp = conEditores.OpenRecordset("SELECT * " & _
      "FROM autores; " & _
      "SELECT * FROM almacenes; " & _
      "SELECT * FROM trabajos")

   ' Intenta imprimir los resultados de cada una de las tres
   ' instrucciones SELECT.
   booSiguiente = True
   intContador = 1
   With rstTemp
      Do While booSiguiente
         Debug.Print "Contenido del recordset número " & intContador
         Do While Not .EOF
            Debug.Print , .Fields(0), .Fields(1)
            .MoveNext
         Loop
         booSiguiente = .NextRecordset
         Debug.Print "    rstTemp.NextRecordset = " & _
            booSiguiente
         intContador = intContador + 1
      Loop
   End With

   rstTemp.Close
   conEditores.Close
   wrkODBC.Close

End Sub
```

Otra forma de llevar a cabo la misma tarea es crear una instrucción preparada que contenga la instrucción SQL compuesta. La propiedad **CacheSize** del objeto **QueryDef** debe ser establecida a 1,

y el objeto **Recordset** sólo debe estar primero y de sólo lectura.

```
Sub SiguienteRecordsetX2()

   Dim wrkODBC As Workspace
   Dim conPublicaciones As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset
   Dim intContador As Integer
   Dim booSiguiente As Boolean

   ' Crea un objeto de espacio de trabajo ODBCDirect y abre el objeto
Connection.
   ' El valor de DefaultCursorDriver es obligatorio
   ' cuando utilice instrucciones SQL compuestas.
   Set wrkODBC = CreateWorkspace("", _
      "admin", "", dbUseODBC)
   wrkODBC.DefaultCursorDriver = dbUseODBCCursor
   Set conPublicaciones = wrkODBC.OpenConnection("Publicaciones", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publicaciones")

   ' Crea un procedimiento temporal almacenado con una instrucción
   ' SELECT compuesta.
   Set qdfTemp = conPublicaciones.CreateQueryDef("", _
      "SELECT * FROM autores; " & _
      "SELECT * FROM almacenes; " & _
      "SELECT * FROM trabajos")
   ' Establece el tamaño de caché y abre el objeto Recordset con los
argumentos
   ' qe ermitirán tener acceso a múltiples recordsets.
   qdfTemp.CacheSize = 1
   Set rstTemp = qdfTemp.OpenRecordset(dbOpenForwardOnly, _
      dbReadOnly)

   ' Intente imprimir los resultados de cada una de las tres instrucciones
   ' SELECT.
   booSiguiente = True
   intContador = 1
   With rstTemp
      Do While booSiguiente
         Debug.Print "Contenido del recordset #" & intContador
         Do While Not .EOF
            Debug.Print , .Fields(0), .Fields(1)
            .MoveNext
         Loop
         booSiguiente = .NextRecordset
         Debug.Print "    rstTemp.NextRecordset = " & _
            booSiguiente
         intContador = intContador + 1
      Loop
   End With

   rstTemp.Close
   qdfTemp.Close
   conPublicaciones.Close
   wrkODBC.Close
```

```
End Sub
```

Ejemplo del método OpenConnection

Este ejemplo utiliza el método **OpenConnection** con parámetros diferentes para abrir tres objetos **Connection** diferentes.

```
Sub OpenConnectionX()

   Dim wrkODBC As Workspace
   Dim conEditores As Connection
   Dim conEditores2 As Connection
   Dim conEditores3 As Connection
   Dim conBucle As Connection

   ' Crea el objeto ODBCDirect Workspace.
   Set wrkODBC = CreateWorkspace("NuevoWorkspaceODBC", _
      "admin", "", dbUseODBC)

   ' Abre el objeto Connection utilizando la información proporcionada
   ' en la cadena de conexión. Si esta información no es suficiente,
   ' podría interceptar un error en vez de ir al cuadro de diálogo
   ' Administrador del controlador ODBC.
   MsgBox "Abriendo conexión1..."
   Set conEditores = wrkODBC.OpenConnection("conexión1", _
      dbDriverNoPrompt, , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Abre el objeto Connection de sólo lectura basado en información
   ' que escribió en el cuadro de diálogo Administrador del controlador
ODBC.
   MsgBox "Abriendo conexión2..."
   Set conEditores2 = wrkODBC.OpenConnection("conexión2", _
      dbDriverPrompt, True, "ODBC;DSN=Editores;")

   ' Abre el objeto Connection de sólo lectura escribiendo sólo la
   ' información que falta en el cuadro de diálogo Administrador del
   ' controlador ODBC.
   MsgBox "Abriendo conexión3..."
   Set conEditores3 = wrkODBC.OpenConnection("conexión3", _
      dbDriverCompleteRequired, True, _
      "ODBC;DATABASE=eds;DSN=Editores;")

   ' Enumera la colección Connections.
   For Each conBucle In wrkODBC.Connections
      Debug.Print "Propiedades Connection para " & _
         conBucle.Name & ":"

      With conBucle
         ' Imprime los valores de la propiedad llamando explícitamente
         ' al objeto Property; el objeto Connection no admite
         ' una colección Properties.
         Debug.Print "    Connect = " & .Connect
         ' La propiedad devuelve actualmente un objeto Database.
         Debug.Print "    Database[.Name] = " & _
            .Database.Name
         Debug.Print "    Name = " & .Name
         Debug.Print "    QueryTimeout = " & .QueryTimeout
         Debug.Print "    RecordsAffected = " & _
```

```
            .RecordsAffected
         Debug.Print "    StillExecuting = " & _
            .StillExecuting
         Debug.Print "    Transactions = " & .Transactions
         Debug.Print "    Updatable = " & .Updatable
      End With

   Next conBucle

   conEditores.Close
   conEditores2.Close
   conEditores3.Close
   wrkODBC.Close

End Sub
```

Ejemplo del método OpenDatabase

Este ejemplo utiliza el método **OpenDatabase** para abrir una base de datos Microsoft Jet y dos bases de datos ODBC conectadas a Microsoft Jet.

```
Sub OpenDatabaseX()

   Dim wrkJet As Workspace
   Dim dbsNeptuno As Database
   Dim dbsEditores As Database
   Dim dbsEditores2 As Database
   Dim dbsBucle As Database
   Dim prpBucle As Property

   ' Crea un objeto Workspace Microsoft Jet.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)

   ' Abre para uso en modo exclusivo el objeto Database de
   ' una base de datos Microsoft Jet guardada.
   MsgBox "Abriendo Neptuno..."
   Set dbsNeptuno = wrkJet.OpenDatabase("Neptuno.mdb", _
      True)

   ' Abre el objeto Database de sólo lectura basado en información
   ' de la cadena de conexión.
   MsgBox "Abriendo Editores..."
   Set dbsEditores = wrkJet.OpenDatabase("Editores", _
      dbDriverNoPrompt, True, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Abre el objeto Database de sólo lectura escribiendo sólo la
   ' información que falta en el cuadro de diálogo Administrador del
   ' controlador ODBC.
   MsgBox "Abriendo la segunda copia de editores..."
   Set dbsEditores2 = wrkJet.OpenDatabase("Editores", _
      dbDriverCompleteRequired, True, _
      "ODBC;DATABASE=eds;DSN=Editores;")

   ' Enumera la colección Databases.
   For Each dbsBucle In wrkJet.Databases
      Debug.Print "Propiedades Database para " & _
         dbsBucle.Name & ":"

      On Error Resume Next
      ' Enumera la colección Properties de cada objeto
      ' Database
      For Each prpBucle In dbsBucle.Properties
         If prpBucle.Name = "Connection" Then
            ' La propiedad devuelve actualmente un objeto Connection.
            Debug.Print "    Connection[.Name] = " & _
               dbsBucle.Connection.Name
         Else
            Debug.Print "    " & prpBucle.Name & " = " & _
               prpBucle
         End If
      Next prpBucle
      On Error GoTo 0
```

```
    Next dbsBucle

    dbsNeptuno.Close
    dbsEditores.Close
    dbsEditores2.Close
    wrkJet.Close

End Sub
```

Ejemplo del método OpenRecordset

Este ejemplo utiliza el método **OpenRecordset** para abrir cinco objetos **Recordset** diferentes y mostrar su contenido. El procedimiento SalidaAbrirRecordset se necesita para ejecutar este procedimiento.

```
Sub OpenRecordsetX()

   Dim wrkJet As Workspace
   Dim wrkODBC As Workspace
   Dim dbsNeptuno As Database
   Dim conEditores As Connection
   Dim rstTemp As Recordset
   Dim rstTemp2 As Recordset

   ' Abre Microsoft Jet y el espacio de trabajo ODBCDirect, la base de
datos
   ' Microsoft Jet y  la conexión ODBCDirect.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
   Set wrkODBC = CreateWorkspace("", "admin", "", dbUseODBC)
   Set dbsNeptuno = wrkJet.OpenDatabase("Neptuno.mdb")
   Set conEditores = wrkODBC.OpenConnection("", , , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Abre cinco objetos Recordset diferentes y muestra
   ' el contenido de cada uno.

   Debug.Print "Abriendo el recordset de tipo Forward-only" & _
      "donde el origen es un objeto QueryDef..."
   Set rstTemp = dbsNeptuno.OpenRecordset( _
      "Los diez productos más caros", dbOpenForwardOnly)
   SalidaAbrirRecordset rstTemp

   Debug.Print " Abriendo el recordset de sólo lectura de tipo Dynaset " &
_
      "donde el origen es una instrucción SQL..."
   Set rstTemp = dbsNeptuno.OpenRecordset( _
      "SELECT * FROM Empleados", dbOpenDynaset, dbReadOnly)
   SalidaAbrirRecordset rstTemp

   ' Utiliza la propiedad Filter para recuperar sólo ciertos
   ' registros con la siguiente llamada OpenRecordset.
   Debug.Print "Abriendo el recordset de un objeto" & _
      "Recordset existente para filtrar..."
   rstTemp.Filter = "Apellidos >= 'M'"
   Set rstTemp2 = rstTemp.OpenRecordset()
   SalidaAbrirRecordset rstTemp2

   Debug.Print " Abriendo el recordset de tipo Dynamic de" & _
      "una conexión ODBC..."
   Set rstTemp = conEditores.OpenRecordset( _
      "SELECT * FROM almacenes", dbOpenDynamic)
   SalidaAbrirRecordset rstTemp

   ' Utiliza la propiedad StillExecuting para determinar cuando está
   ' el Recordset preparado para manipular.
   Debug.Print " Abriendo el recordset de tipo Snapshot basado" & _
```

```
      "en una consulta asíncrona a una conexión ODBC..."
   Set rstTemp = conEditores.OpenRecordset("editores", _
      dbOpenSnapshot, dbRunAsync)
   Do While rstTemp.StillExecuting
      Debug.Print "    [todavía en ejecución...]"
   Loop
   SalidaAbrirRecordset rstTemp

   rstTemp.Close
   dbsNeptuno.Close
   conEditores.Close
   wrkJet.Close
   wrkODBC.Close

End Sub

Sub SalidaAbrirRecordset(rstOutput As Recordset)

   ' Enumera el objeto Recordset especificado.
   With rstOutput
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
   End With

End Sub
```

Ejemplo del método PopulatePartial

El siguiente ejemplo utiliza el método **PopulatePartial** después de cambiar un filtro de réplica.

```
Dim tdfClientes As TableDef, strFiltro As String

   Dim dbs As Database
   Dim strFilter As String
   Dim dbsTemp As Database

' Abre la réplica parcial en modo exclusivo:
   Set dbs = OpenDatabase("F:\VENTAS\FY96CA.MDB", True)

   With dbsTemp
      Set tdfClientes = .TableDefs("Clientes")

      ' Se sincroniza con la réplica completa
      ' antes de configurar los filtros de la réplica
      dbs.Synchronize "C:\VENTAS\FY96.MDB"

      strFiltro = "Región = 'SP'"
      tdfClientes.ReplicaFilter = strFiltro

      ' Rellena registros de la réplica completa
      .PopulatePartial "C:\VENTAS\FY96.MDB"

      dbs.Close
   End With

End Sub
```

Ejemplo del método RefreshLink

Este ejemplo utiliza el método **RefreshLink** para actualizar los datos en una tabla vinculada después de que se haya cambiado la conexión de un origen de datos a otro. El procedimiento SalidaActualizarVínculo se necesita para ejecutar este procedimiento.

```
Sub RefreshLinkX()

   Dim dbsActual As Database
   Dim tdfVinculado As TableDef

   ' Abre una base de datos a la que se va a anexar una tabla
   ' vinculada.
   Set dbsActual = OpenDatabase("BD1.mdb")

   ' Crea una tabla vinculada que señala a una base de datos
   ' SQL Server.
   Set tdfVinculado = _
      dbsActual.CreateTableDef("TablaAutores")
   tdfVinculado.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores"
   tdfVinculado.SourceTableName = "autores"
   dbsActual.TableDefs.Append tdfVinculado

   ' Muestra el contenido de la tabla vinculada.
   Debug.Print _
      "Los datos de la tabla vinculada conectados al primer origen:"
   SalidaActualizarVínculo dbsActual

   ' Cambia la información de conexión para la tabla vinculada y
   ' actualiza la conexión para que estén disponibles los nuevos
   ' datos.
   tdfVinculado.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=NuevosEditores"
   tdfVinculado.RefreshLink

   ' Muestra el contenido de la tabla vinculada.
   Debug.Print _
      "Los datos de la tabla vinculada conectados al segundo origen:"
   SalidaActualizarVínculo dbsActual

   ' elimina la tabla porque esto es un ejemplo.
   dbsActual.TableDefs.Delete tdfVinculado.Name

   dbsActual.Close

End Sub

Sub SalidaActualizarVínculo(dbsTemp As Database)

   Dim rstRemote As Recordset
   Dim intContador As Integer

   ' Abre la tabla vinculada.
   Set rstRemote = _
      dbsTemp.OpenRecordset("TablaAutores")
```

```
   intContador = 0

   ' Enumera el objeto Recordset, pero para a los 50 registros.
   With rstRemote
      Do While Not .EOF And intContador < 50
         Debug.Print , .Fields(0), .Fields(1)
         intContador = intContador + 1
         .MoveNext
      Loop
      If Not .EOF Then Debug.Print , "[más registros]"
      .Close
   End With

End Sub
```

Ejemplo del método RegisterDatabase

Este ejemplo utiliza el método **RegisterDatabase** para registrar un origen de datos Microsoft SQL Server llamado Editores en el Registro de Windows.

Utilizar el icono ODBC del Panel de control de Windows es la mejor forma de crear, modificar o eliminar nombres de orígenes de datos.

```
Sub RegisterDatabaseX()

   Dim dbsRegistro As Database
   Dim strDescripción As String
   Dim strAtributos As String
   Dim errBucle As Error

   ' Genera la cadena de palabras clave.
   strDescripción = InputBox( "Escriba una descripción " & _
      "para registrar la base de datos.")
   strAtributos = "Database=eds" & _
      vbCr & "Description=" & strDescripción & _
      vbCr & "OemToAnsi=No" & _
      vbCr & "Server=Server1"

   ' Actualiza el Registro de Windows.
   On Error GoTo Err_Registro
   DBEngine.RegisterDatabase "Editores", "SQL Server", _
      True, strAtributos
   On Error GoTo 0

   MsgBox "Utilice regedit.exe para ver los cambios: " & _
      "HKEY_CURRENT_USER\" & _
      "Software\ODBC\ODBC.INI"

   Exit Sub

Err_Registro:

   ' Informa al usuario de cualquier error que se produzca debido a
   ' datos no válidos.
   If DBEngine.Errors.Count > 0 Then
      For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & _
            vbCr & errBucle.Description
      Next errBucle
   End If

   Resume Next

End Sub
```

Ejemplo del método Requery

Este ejemplo muestra cómo se puede utilizar el método **Requery** para actualizar una consulta después de que haya cambiado la tabla base.

```
Sub RequeryX()

   Dim dbsNeptuno As Database
   Dim qdfTemp As QueryDef
   Dim rstVer As Recordset
   Dim rstCambiar As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set qdfTemp = dbsNeptuno.CreateQueryDef("", _
      "PARAMETERS VerPaís Text; " & _
      "SELECT Nombre, Apellidos, País FROM Empleados " & _
      "WHERE País = [VerPaís] _
      "ORDER BY Apellidos")

   qdfTemp.Parameters!VerPaís = "México"
   Debug.Print "Datos después de la consulta inicial,_
      [VerPaís] = México"
   Set rstVer = qdfTemp.OpenRecordset
   Do While Not rstVer.EOF
      Debug.Print "    " & rstVer!Nombre & " " & _
         rstVer!Apellidos & ", " & rstVer!País
      rstVer.MoveNext
   Loop

   ' Cambia los datos base.
   Set rstCambiar = dbsNeptuno.OpenRecordset("Empleados")
   rstCambiar.AddNew
   rstCambiar!Nombre = "Guadalupe"
   rstCambiar!Apellidos = "Alonso"
   rstCambiar!País = "México"
   rstCambiar.Update

   rstVer.Requery
   Debug.Print "Requery después de cambiar los datos base"
   Set rstVer = qdfTemp.OpenRecordset
   Do While Not rstVer.EOF
      Debug.Print "    " & rstVer!Nombre & " " & _
         rstVer!Apellidos & ", " & rstVer!País
      rstVer.MoveNext
   Loop

   ' Restaura los datos originales porque esto es sólo
   ' un ejemplo.
   rstCambiar.Bookmark = rstCambiar.LastModified
   rstCambiar.Delete
   rstCambiar.Close

   rstVer.Close
   dbsNeptuno.Close

End Sub
```

Este ejemplo muestra cómo se puede utilizar el método **Requery** para actualizar una consulta después de que se hayan cambiado los parámetros de la consulta.

```
Sub RequeryX2()

   Dim dbsNeptuno As Database
   Dim qdfTemp As QueryDef
   Dim prmPaís As Parameter
   Dim rstVer As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set qdfTemp = dbsNeptuno.CreateQueryDef("", _
      "PARAMETERS VerPaís Text; " & _
      "SELECT Nombre, Apellidos, País FROM Empleados " & _
      "WHERE País = [VerPaís] _
      "ORDER BY Apellidos")
   Set prmPaís = qdfTemp.Parameters!VerPaís

   qdfTemp.Parameters!VerPaís = "México"
   Debug.Print "Datos después dela consulta inicial,_
      [VerPaís] = México"
   Set rstVer = qdfTemp.OpenRecordset
   Do While Not rstVer.EOF
      Debug.Print "    " & rstVer!Nombre & " " & _
         rstVer!Apellidos & ", " & rstVer!País
      rstVer.MoveNext
   Loop

   ' Cambia los parámetros de la consulta.
   qdfTemp.Parameters!VerPaís = "España"
   ' Se debe incluir el argumento QueryDef para que el Recordset
resultante
   ' refleje el cambio en
   ' el parámetro de la consulta.
   rstVer.Requery qdfTemp
   Debug.Print "Requery después de cambiar el parámetro, _
   "[VerPaís] = España"
   Do While Not rstVer.EOF
      Debug.Print "    " & rstVer!Nombre & " " & _
         rstVer!Apellidos & ", " & rstVer!País
      rstVer.MoveNext
   Loop

   rstVer.Close
   dbsNeptuno.Close

End Sub
```

Ejemplo del método SetOption

Este ejemplo utiliza el método **SetOption** para cambiar el valor de dos claves de registro basadas en una entrada del usuario. El método **SetOption** sólo sobrescribe el valor del registro almacenado en la aplicación actual. Los valores almacenados no se cambiarán y estarán sólo los valores visibles al usuario a través de REGEDIT.EXE.

```
Sub SetOptionX()

   Dim intRetrasoExclusivo As Integer
   Dim intRetrasoCompartido As Integer

   ' Obtiene una entrada de usuario para las claves de registro
   ' ExclusiveAsyncDelay y SharedAsyncDelay.
   intRetrasoExclusivo = Val(InputBox("Escriba un nuevo valor " & _
      " para la clave de registro ExclusiveAsyncDelay " & _
      "(en milisegundos):"))
   intRetrasoCompartido = Val(InputBox("Escriba un nuevo valor " & _
      " para la clave de registro SharedAsyncDelay " & _
      "( en milisegundos):"))

   If intRetrasoExclusivo > 0 And intRetrasoCompartido > 0 Then
      ' Cambia los valores de las claves de registro.
      SetOption dbExclusiveAsyncDelay, intRetrasoExclusivo
      SetOption dbSharedAsyncDelay, intRetrasoCompartido
      MsgBox "Claves de registro cambiadas _
      "a los nuevos valores."
   Else
      MsgBox " Claves de registro no cambiadas."
   End If

End Sub
```

Ejemplo del método Synchronize

Estos cuatro ejemplos utilizan el método **Synchronize** para demostrar intercambios de información unidireccionales y bidireccionales entre dos miembros de un conjunto de réplica. Funcionará si ha convertido Neptuno.mdb a un Diseño principal (vea la propiedad **Replicable**) y creó una réplica de él. El nombre de réplica especificado es NuevaRéplica.mdb. Cambie el nombre de la réplica para adaptarla a su situación o utilice el método **MakeReplica** para crear una réplica si la necesita.

Este ejemplo envía los cambios del Diseño principal de Neptuno a NuevaRéplica. Ajuste las rutas de acceso a las ubicaciones de los archivos en su equipo.

```
Sub SendChangeToReplicaX()

   Dim dbsNeptuno As Database

   'Abre la base de datos replicable Neptuno.mdb
   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   'Envía cambios de datos o estructurales a la réplica
   dbsNeptuno.Synchronize "NuevaRéplica.mdb", _
      dbRepExportChanges

   dbsNeptuno.Close

End Sub
```

En este ejemplo, la base de datos replicable Neptuno.mdb recibe cambios de la réplica en la ruta de acceso - NuevaRéplica. Debe ejecutar este procedimiento desde la base de datos que recibe los cambios.

```
Sub ReceiveChangeX()

   Dim dbsNeptuno As Database

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   'Envía cambios desde la réplica al Diseño principal
   dbsNeptuno.Synchronize "NuevaRéplica", _
      dbRepImportChanges

   dbsNeptuno.Close

End Sub
```

En este ejemplo, se intercambian los cambios de la base de datos replicable Neptuno y la réplica. Este es el argumento predeterminado para este método.

```
Sub TwoWayExchangeX()

   Dim dbsNeptuno As Database

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   'Envía cambios realizados en cada réplica a la otra
   dbsNeptuno.Synchronize "NuevaRéplica", _
      dbRepImpExpChanges

   dbsNeptuno.Close
```

```
End Sub
```

El siguiente ejemplo de código sincroniza dos bases de datos en Internet.

```
Sub InternetSynchronizeX()

   Dim dbs As Database

   Set dbs = OpenDatabase("C:\Datos\EntPed.mdb")

   ' Sincroniza la base de datos local con la réplica en el
   ' servidor Internet.

   dbs.Synchronize "http://www.miempresa.miservidor.com" _
      & "/archivos/Pedidos.mdb", dbRepImpExpChanges + _
      dbRepSyncInternet

   dbs.Close

End Sub
```

Ejemplo de la propiedad AbsolutePosition

Este ejemplo utiliza la propiedad **AbsolutePosition** para controlar el progreso de un bucle que enumera todos los registros de un **Recordset**.

```
Sub AbsolutePositionX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strMensaje As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' AbsolutePosition sólo funciona con tipos Dynaset o Snapshot.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenSnapshot)

   With rstEmpleados
      ' Llena el Recordset.
      .MoveLast
      .MoveFirst

      ' Enumera el Recordset.
      Do While Not .EOF
         ' Muestra la información del registro activo.
         ' Agrega 1 al valor de AbsolutePosition porque empieza en cero.
         strMensaje = "Empleado: " & !Apellidos & vbCr & _
            "(registro " & (.AbsolutePosition + 1) & _
            " de " & .RecordCount & ")"
         If MsgBox(strMensaje, vbOKCancel) = vbCancel _
            Then Exit Do
         .MoveNext
      Loop

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad AllowZeroLength

En este ejemplo, la propiedad **AllowZeroLength** permite al usuario establecer el valor de un objeto **Field** a una cadena vacía. En esta situación, el usuario puede distinguir entre un registro del que desconoce sus datos y un registro cuyos datos no puede aplicar.

```
Sub AllowZeroLengthX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim fldTemp As Field
   Dim rstEmpleados As Recordset
   Dim strMensaje As String
   Dim strEntrada As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno.TableDefs("Empleados")
   ' Crea un nuevo objeto Field y lo anexa a la
   ' colección Fields de la tabla Empleados.
   Set fldTemp = tdfEmpleados.CreateField("Fax", _
      dbText, 24)
   fldTemp.AllowZeroLength = True
   tdfEmpleados.Fields.Append fldTemp

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      ' Obtiene la entrada del usuario.
      .Edit
      strMensaje = "Escriba el número de fax de " & _
         !Nombre & " " & !Apellidos & "." & vbCr & _
         "[? - desconocido, X - no tiene fax]"
      strEntrada = UCase(InputBox(strMensaje))
      If strEntrada <> "" Then
         Select Case strEntrada
            Case "?"
               !Fax = Null
            Case "X"
               !Fax = ""
            Case Else
               !Fax = strEntrada
         End Select

         .Update

         ' Imprime un informe.
         Debug.Print "Nombre - Número de fax"
         Debug.Print !Nombre & " " & !Apellidos & " - ";

         If IsNull(!Fax) Then
            Debug.Print "[Desconocido]"
         Else
            If !Fax = "" Then
               Debug.Print "[No tiene fax]"
            Else
               Debug.Print !Fax
```

```
            End If
         End If

      Else
         .CancelUpdate
      End If

      .Close
   End With

   ' Elimina el nuevo campo porque esto es un ejemplo.
   tdfEmpleados.Fields.Delete fldTemp.Name
   dbsNeptuno.Close

End Sub
```

Ejemplo de las propiedades BOF y EOF

Este ejemplo demuestra cómo las propiedades **BOF** y **EOF** permiten que el usuario se mueva hacia adelante y hacia atrás en un objeto **Recordset**.

```
Sub BOFX()

   Dim dbsNeptuno As Database
   Dim rstCategorías As Recordset
   Dim strMensaje As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstCategorías = _
      dbsNeptuno.OpenRecordset("Categorías", _
      dbOpenSnapshot)

   With rstCategorías
      ' Llena el Recordset.
      .MoveLast
      .MoveFirst

      Do While True
         ' Muestra la información del registro activo
         ' y obtiene una entrada del usuario.
         strMensaje = "Categoría: " & !NombreCategoría & _
            vbCr & "(registro " & (.AbsolutePosition + 1) & _
            " de " & .RecordCount & ")" & vbCr & vbCr & _
            "Escriba 1 para avanzar, 2 para retroceder:"

         ' Se mueve hacia adelante o hacia atrás para llegar hasta BOF o
EOF.
         Select Case InputBox(strMensaje)
            Case 1
               .MoveNext
               If .EOF Then
                  MsgBox _
                     "¡Final del archivo!" & vbCr & _
                     "El puntero se ha movido al último registro."
                  .MoveLast
               End If

            Case 2
               .MovePrevious
               If .BOF Then
                  MsgBox _
                     "¡Principio del archivo!" & vbCr & _
                     " El puntero se ha movido al primer registro."
                  .MoveFirst
               End If

            Case Else
               Exit Do
         End Select

      Loop

      .Close
```

```
    End With

    dbsNeptuno.Close

End Sub
```

Ejemplo de las propiedades Bookmark y Bookmarkable

Este ejemplo utiliza las propiedades **Bookmark** y **Bookmarkable** para permitir que el usuario coloque un indicador en un registro de un objeto **Recordset** y vuelva a él más tarde.

```
Sub BookmarkX()

   Dim dbsNeptuno As Database
   Dim rstCategorías As Recordset
   Dim strMensaje As String
   Dim intComando As Integer
   Dim varMarcador As Variant

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstCategorías = _
      dbsNeptuno.OpenRecordset("Categorías", _
      dbOpenSnapshot)

   With rstCategorías

      If .Bookmarkable = False Then
         Debug.Print "¡No se puede poner un marcador en el Recordset!"
      Else
         ' Llena el Recordset.
         .MoveLast
         .MoveFirst

         Do While True
            ' Muestra información acerca del registro
            ' activo y obtiene una entrada del usuario.
            strMensaje = "Categoría: " & !NombreCategoría & _
               " (registro " & (.AbsolutePosition + 1) & _
               " de  " & .RecordCount & ")" & vbCr & _
               "Escriba un comando:" & vbCr & _
               "[1 - siguiente / 2 - anterior /" & vbCr & _
               "3 - establecer marcador / 4 - ir al marcador]"
            intComando = Val(InputBox(strMensaje))

            Select Case intComando
         ' Se mueve hacia adelante o hacia atrás para
         ' llegar hasta BOF o EOF.
               Case 1
                  .MoveNext
                  If .EOF Then .MoveLast
               Case 2
                  .MovePrevious
                  If .BOF Then .MoveFirst

               ' Almacena el marcador del registro activo.
               Case 3
                  varMarcador = .Bookmark

               ' Va al registro indicado en el
               ' marcador almacenado.
               Case 4
                  If IsEmpty(varMarcador) Then
                     MsgBox "¡No hay establecido ningún marcador!"
```

```
                    Else
                       .Bookmark = varMarcador
                    End If

                Case Else
                    Exit Do
            End Select

        Loop

    End If

    .Close
End With

dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Count

Este ejemplo demuestra la propiedad **Count** con tres colecciones diferentes en la base de datos Neptuno. La propiedad obtiene el número de objetos de cada colección y establece el límite superior para los bucles que enumeran estas colecciones. Otra forma de enumerar estas colecciones sin utilizar la propiedad **Count** podría ser utilizando instrucciones **For Each...Next**.

```
Sub CountX()

   Dim dbsNeptuno As Database
   Dim intBucle As Integer

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Imprime información acerca de la colección TableDefs.
      Debug.Print .TableDefs.Count & _
         " TableDefs en Neptuno"
      For intBucle = 0 To .TableDefs.Count - 1
         Debug.Print "    " & .TableDefs(intBucle).Name
      Next intBucle

      ' Imprime información acerca de la colección QueryDefs.
      Debug.Print .QueryDefs.Count & _
         " QueryDefs en Neptuno"
      For intBucle = 0 To .QueryDefs.Count - 1
         Debug.Print "    " & .QueryDefs(intBucle).Name
      Next intBucle

      ' Imprime información acerca de la colección Relations.
      Debug.Print .Relations.Count & _
         " Relations en Neptuno"
      For intBucle = 0 To .Relations.Count - 1
         Debug.Print "    " & .Relations(intBucle).Name
      Next intBucle

      .Close
   End With

End Sub
```

Ejemplo de la propiedad DataUpdatable

Este ejemplo demuestra la propiedad **DataUpdatable** utilizando el primer campo de seis **Recordsets** diferentes. Se necesita la función SalidaDatos para ejecutar este procedimiento.

```
Sub DataUpdatableX()

   Dim dbsNeptuno As Database
   Dim rstNeptuno As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Abre e imprime un informe acerca del Recordset de tipo Table.
      Set rstNeptuno = .OpenRecordset("Empleados")
      SalidaDatos rstNeptuno

      ' Abre e imprime un informe acerca del Recordset de tipo Dynaset.
      Set rstNeptuno = .OpenRecordset("Empleados", _
         dbOpenDynaset)
      SalidaDatos rstNeptuno

      ' Abre e imprime un informe acerca del Recordset de tipo Snapshot.
      Set rstNeptuno = .OpenRecordset("Empleados", _
         dbOpenSnapshot)
      SalidaDatos rstNeptuno

      ' Abre e imprime un informe acerca del Recordset de tipo Forward-
only.
      Set rstNeptuno = .OpenRecordset("Empleados", _
         dbOpenForwardOnly)
      SalidaDatos rstNeptuno

      ' Abre e imprime un informe acerca del
      ' Recordset basado en una consulta de selección.
      Set rstNeptuno = _
         .OpenRecordset("Lista actual de productos ")
      SalidaDatos rstNeptuno

      ' Abre e imprime un informe acerca del
      ' Recordset basado en una consulta de selección que calcula totales.
      Set rstNeptuno = .OpenRecordset("Subtotales por pedido")
      SalidaDatos rstNeptuno

      .Close
   End With

End Sub

Function SalidaDatos(rstTemp As Recordset)

   With rstTemp
      Debug.Print "Recordset: " & .Name & ", ";
      Select Case .Type
         Case dbOpenTable
            Debug.Print "dbOpenTable"
         Case dbOpenDynaset
```

```
            Debug.Print "dbOpenDynaset"
         Case dbOpenSnapshot
            Debug.Print "dbOpenSnapshot"
         Case dbOpenForwardOnly
            Debug.Print "dbOpenForwardOnly"
      End Select
      Debug.Print "    Campo: " & .Fields(0).Name & ", " & _
         "DataUpdatable = " & .Fields(0).DataUpdatable
      Debug.Print
      .Close
   End With

End Function
```

Ejemplo de las propiedades DateCreated y LastUpdated

Este ejemplo demuestra las propiedades **DateCreated** y **LastUpdated** agregando un nuevo objeto **Field** al objeto **TableDef** existente y creando un nuevo **TableDef**. Se necesita la función SalidaDatos para ejecutar este procedimiento.

```
Sub DateCreatedX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim tdfNuevaTabla As TableDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      Set tdfEmpleados = .TableDefs!Empleados

      With tdfEmpleados
         ' Imprime la información actual acerca de
         ' la tabla Empleados.
         SalidaFecha "Propiedades actuales", tdfEmpleados

         ' Crea y anexa un campo a la tabla Empleados.
         .Fields.Append .CreateField("CampoNuevo", dbDate)

         ' Imprime la nueva información acerca de la
         ' tabla Empleados.
         SalidaFecha "Después de crear un campo nuevo", _
            tdfEmpleados

         ' Elimina el nuevo objeto Field porque esto es un ejemplo.
         .Fields.Delete "CampoNuevo"
      End With

      ' Crea y anexa un objeto TableDef nuevo a la
      ' base de datos Neptuno.
      Set tdfNuevaTabla = .CreateTableDef("NuevoTableDef")
      With tdfNuevaTabla
         .Fields.Append .CreateField("CampoNuevo", dbDate)
      End With
      .TableDefs.Append tdfNuevaTabla

      ' Imprime información acerca del objeto TableDef nuevo.
      SalidaFecha "Después de crear una nueva tabla", tdfNuevaTabla

      ' Elimina el objeto TableDef nuevo porque esto
      ' es un ejemplo.
      .TableDefs.Delete tdfNuevaTabla.Name
      .Close
   End With

End Sub

Function SalidaFecha(strTemp As String, _
   tdfTemp As TableDef)

   ' Imprime la información de DateCreated y
```

```
    ' LastUpdated acerca del objeto TableDef especificado.
    Debug.Print strTemp
    Debug.Print "    TableDef: " & tdfTemp.Name
    Debug.Print "        DateCreated = " & _
       tdfTemp.DateCreated
    Debug.Print "        LastUpdated = " & _
       tdfTemp.LastUpdated
    Debug.Print

End Function
```

Ejemplo de la propiedad DefaultValue

Este ejemplo utiliza la propiedad **DefaultValue** para advertir al usuario de un valor normal de un campo mientras se piden datos de entrada. Además, demuestra cómo se llenarán los nuevos registros utilizando **DefaultValue** si no se especifica ninguna otra entrada. Se necesita la función SolicitudPredeterminado para ejecutar este procedimiento.

```
Sub DefaultValueX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim strAntiguoPredeterminado As String
   Dim rstEmpleados As Recordset
   Dim strMensaje As String
   Dim strCódigo As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno.TableDefs!Empleados

   ' Almacena la información original de DefaultValue
   ' y establece un nuevo valor en la propiedad.
   strAntiguoPredeterminado = _
      tdfEmpleados.Fields!CódPostal.DefaultValue
   tdfEmpleados.Fields!CódPostal.DefaultValue = "98052"

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   With rstEmpleados
      ' Agrega un registro nuevo al Recordset.
      .AddNew
      !Nombre = "Carlos"
      !Apellidos = "Peña"

      ' Obtiene una entrada del usuario. Si el usuario
      ' escribe algo, el campo se llenará con esos
      ' datos; de lo contrario, se llenará con la información de
DefaultValue.
      strMensaje = "Escriba el código postal de " & vbCr & _
         !Nombre & " " & !Apellidos & ":"
      strCódigo = SolicitudPredeterminado(strMensaje, !CódPostal)
      If strCódigo <> "" Then !CódPostal = strCódigo
      .Update

      ' Va al registro nuevo e imprime la información.
      .Bookmark = .LastModified
      Debug.Print "    Nombre = " & !Nombre
      Debug.Print "    Apellidos = " & !Apellidos
      Debug.Print "    CódPostal = " & !CódPostal

      ' Elimina el registro nuevo porque esto es un ejemplo.
      .Delete
      .Close
   End With

   ' Restaura el valor original de la propiedad
```

```
   ' DefaultValue porque esto es un ejemplo.
   tdfEmpleados.Fields!CódPostal.DefaultValue = _
      strAntiguoPredeterminado

   dbsNeptuno.Close

End Sub

Function SolicitudPredeterminado(strSolicitud As String, _
   fldTemp As Field) As String

   Dim strSolicitudCompleta As String

   ' Pregunta al usuario el nuevo valor de
   ' DefaultValue para el objeto Field especificado.
   strSolicitudCompleta = strSolicitud & vbCr & _
      "[Predeterminado = " & fldTemp.DefaultValue & _
      ", Cancelar - utilizar predeterminado]"
   SolicitudPredeterminado = InputBox(strSolicitudCompleta)

End Function
```

Ejemplo de la propiedad DistinctCount

Este ejemplo utiliza la propiedad **DistinctCount** para mostrar cómo puede determinar el número de valores únicos en un objeto **Index**. Sin embargo, este valor sólo es exacto inmediatamente después de crear el **Index**. Seguirá siendo exacto si no cambia las claves o si las claves nuevas se agregan y no se eliminan las viejas; de lo contrario, no será fiable. (Si este procedimiento se ejecuta varias veces, puede ver el efecto en los valores de la propiedad **DistinctCount** de los objetos **Index** existentes.)

```
Sub DistinctCountX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim idxPaís As Index
   Dim idxBucle As Index
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno!Empleados

   With tdfEmpleados
      ' Crea y anexa un objeto Index nuevo a la
      ' tabla Empleados.
      Set idxPaís = .CreateIndex("ÍndicePaís")
      idxPaís.Fields.Append _
         idxPaís.CreateField("País")
      .Indexes.Append idxPaís

      ' La colección se debe actualizar para
      ' que los nuevos datos de DistinctCount estén disponibles.
      .Indexes.Refresh

      ' Enumera la colección Indexes para mostrar los
      ' valores de DistinctCount actuales.
      Debug.Print "Indexes antes de agregar un registro nuevo "
      For Each idxBucle In .Indexes
         Debug.Print "    DistinctCount = " & _
            idxBucle.DistinctCount & ", Nombre = " & _
            idxBucle.Name
      Next idxBucle

      Set rstEmpleados = _
         dbsNeptuno.OpenRecordset("Empleados")

      ' Agrega un registro nuevo a la tabla Empleados.
      With rstEmpleados
         .AddNew
         !Nombre = "López García"
         !Apellidos = "Juan"
         !País = "España"
         .Update
      End With

      ' Enumera la colección Indexes para mostrar los
      ' valores de DistinctCount modificados.
      Debug.Print "Indexes después de agregar un registro nuevo y " & _
```

```
         "actualizar Indexes"
      .Indexes.Refresh
      For Each idxBucle In .Indexes
         Debug.Print "    DistinctCount = " & _
            idxBucle.DistinctCount & ", Nombre = " & _
            idxBucle.Name
      Next idxBucle

      ' Elimina el registro nuevo porque esto es un ejemplo.
      With rstEmpleados
         .Bookmark = .LastModified
         .Delete
         .Close
      End With

      ' Elimina la nueva colección Indexes porque esto es un ejemplo.
      .Indexes.Delete idxPaís.Name
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad EditMode

Este ejemplo muestra el valor de la propiedad **EditMode** en varias condiciones. Se necesita la función SalidaModoModificación para ejecutar este procedimiento.

```
Sub EditModeX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   ' Muestra la propiedad EditMode en estados de
   ' modificación diferentes.
   With rstEmpleados
      SalidaModoModificación "Antes de cualquier Edit o
AddNew:", .EditMode
      .Edit
      SalidaModoModificación "Después de Edit:", .EditMode
      .Update
      SalidaModoModificación "Después de Update:", .EditMode
      .AddNew
      SalidaModoModificación " Después de AddNew:", .EditMode
      .CancelUpdate
      SalidaModoModificación " Después de CancelUpdate:", .EditMode
      .Close
   End With

   dbsNeptuno.Close

End Sub

Function SalidaModoModificación(strTemp As String, _
   intEditMode As Integer)

   ' Imprime un informe basado en el valor de la
   ' propiedad EditMode.
   Debug.Print strTemp
   Debug.Print "    EditMode = ";

   Select Case intEditMode
      Case dbEditNone
         Debug.Print "dbEditNone"
      Case dbEditInProgress
         Debug.Print "dbEditInProgress"
      Case dbEditAdd
         Debug.Print "dbEditAdd"
   End Select

End Function
```

Ejemplo de la propiedad FieldSize

Este ejemplo utiliza la propiedad **FieldSize** para enumerar los números de bytes usados por los objetos **Field** de tipo Memo y Long Binary en dos tablas diferentes.

```
Sub FieldSizeX()

   Dim dbsNeptuno As Database
   Dim rstCategorías As Recordset
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstCategorías = _
      dbsNeptuno.OpenRecordset("Categorías", _
      dbOpenDynaset)
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   Debug.Print _
      "Tamaños de campo de los registros en la tabla Categorías"

   With rstCategorías
      Debug.Print "    NombreCategoría - " & _
         "Descripción (bytes) - Ilustración (bytes)"

   ' Enumera el Recordset de Categorías e imprime el
   ' tamaño en bytes del campo Ilustración para cada registro.
      Do While Not .EOF
         Debug.Print "        " & !NombreCategoría & " - " & _
            !Descripción.FieldSize & " - " & _
            !Ilustración.FieldSize
         .MoveNext
      Loop

      .Close
   End With

   Debug.Print "Tamaños de campo de los registros en la tabla Empleados"

   With rstEmpleados
      Debug.Print "    Apellidos - Notas (bytes) - " & _
         "Foto (bytes)"

   ' Enumera el Recordset de Empleados e imprime el
   ' tamaño en bytes del campo Ilustración para cada registro.
      Do While Not .EOF
         Debug.Print "        " & !Apellidos & " - " & _
            !Notas.FieldSize & " - " & !Foto.FieldSize
         .MoveNext
      Loop

      .Close
   End With

   dbsNeptuno.Close
```

```
End Sub
```

Ejemplo de la propiedad Filter

Este ejemplo utiliza la propiedad **Filter** para crear un objeto **Recordset** nuevo a partir de un **Recordset** existente basado en una condición específica. Se necesita la función FiltroCampo para ejecutar este procedimiento.

```
Sub FilterX()

   Dim dbsNeptuno As Database
   Dim rstPedidos As Recordset
   Dim intPedidos As Integer
   Dim strPaís As String
   Dim rstPedidosPaís As Recordset
   Dim strMensaje As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstPedidos = dbsNeptuno.OpenRecordset("Pedidos", _
      dbOpenSnapshot)

   ' Llena el Recordset.
   rstPedidos.MoveLast
   intPedidos = rstPedidos.RecordCount

   ' Obtiene una entrada del usuario.
   strPaís = Trim(InputBox( _
      "Escriba un país para crear el filtro:"))

   If strPaís <> "" Then
      ' Abre un objeto Recordset filtrado.
      Set rstPedidosPaís = _
         FiltroCampo(rstPedidos, "País destinatario", strPaís)

      With rstPedidosPaís
         ' Comprueba RecordCount antes de llenar el Recordset;
         ' de lo contrario, puede ocurrir un error.
         If .RecordCount <> 0 Then .MoveLast
         ' Imprime el número de registros del objeto
         ' Recordset original y el objeto Recordset
         ' filtrado.
         strMensaje = "Pedidos en el Recordset original: " & _
            vbCr & intPedidos & vbCr & _
            "Pedidos en el Recordset filtrado (País = '" & _
            strPaís & "'): " & vbCr & .RecordCount
         MsgBox strMensaje
         .Close
      End With

   End If

   rstPedidos.Close

   dbsNeptuno.Close

End Sub

Function FiltroCampo(rstTemp As Recordset, _
   strField As String, strFilter As String) As Recordset
```

```
    ' Establece un filtro en el objeto Recordset
    ' especificado y abre un objeto Recordset nuevo.
    rstTemp.Filter = strField & " = '" & strFilter & "'"
    Set FiltroCampo = rstTemp.OpenRecordset

End Function
```

Nota Para ver los efectos de filtrar `rstPedidos`, debe establecer su propiedad **Filter** y abrir un segundo objeto **Recordset** basado en `rstPedidos`.

Nota Cuando conoce los datos que desea seleccionar, normalmente es mejor crear un **Recordset** con una instrucción SQL. Este ejemplo muestra cómo puede crear solamente un **Recordset** y obtener los registros de un país concreto.

```
Sub FilterX2()

    Dim dbsNeptuno As Database
    Dim rstPedidos As Recordset

    Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

    ' Abre un objeto Recordset que selecciona registros
    ' de una tabla basada en el país destinatario.
    Set rstPedidos = _
       dbsNeptuno.OpenRecordset("SELECT * " & _
       "FROM Pedidos WHERE PaísDestinatario = 'México'", _
       dbOpenSnapshot)

    rstPedidos.Close
    dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Foreign

Este ejemplo muestra cómo la propiedad **Foreign** puede indicar qué objetos **Index** en un **TableDef** son índices de clave externa. Estos índices los crea el motor de base de datos Microsoft Jet cuando se crea el objeto **Relation**. El nombre predeterminado para los índices de clave externa es el nombre de la tabla principal y el nombre de la tabla externa. Se necesita la función SalidaExterna para ejecutar este procedimiento.

```
Sub ForeignX()

   Dim dbsNeptuno As Database

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Imprime un informe de índices de clave externa
      ' de dos objetos TableDef y un objeto QueryDef.
      SalidaExterna .TableDefs!Productos
      SalidaExterna .TableDefs!Pedidos
      SalidaExterna .TableDefs![Detalles de pedidos]

      .Close
   End With

End Sub

Function SalidaExterna(tdfTemp As TableDef)

   Dim idxBucle As Index

   With tdfTemp
      Debug.Print " Indexes en " & .Name & " TableDef"
      ' Enumera la colección Indexes del objeto
      ' TableDef especificado.
      For Each idxBucle In .Indexes
         Debug.Print "    " & idxBucle.Name
         Debug.Print "        Foreign = " & idxBucle.Foreign
      Next idxBucle
   End With

End Function
```

Ejemplo de las propiedades ForeignName, ForeignTable y Table

Este ejemplo muestra cómo las propiedades **Table**, **ForeignTable** y **ForeignName** definen los términos de un objeto **Relation** entre dos tablas.

```
Sub ForeignNameX()

   Dim dbsNeptuno As Database
   Dim relBucle As Relation

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   Debug.Print "Relación"
   Debug.Print "                    Tabla - Campo"
   Debug.Print "    Principal (Uno)   ";
   Debug.Print ".Table - .Fields(0).Name"
   Debug.Print "    Externa (Varios)  ";
   Debug.Print ".ForeignTable - .Fields(0).ForeignName"

   ' Enumera la colección Relations de la base de
   ' datos Neptuno para crear un informe de los
   ' valores de la propiedad de los objetos Relation y sus objetos Field.
   For Each relBucle In dbsNeptuno.Relations
      With relBucle
         Debug.Print
         Debug.Print .Name & " Relación"
         Debug.Print "                    Tabla - Campo"
         Debug.Print "    Principal (Uno)   ";
         Debug.Print .Table & " - " & .Fields(0).Name
         Debug.Print "    Externa (Varios)  ";
         Debug.Print .ForeignTable & " - " & _
            .Fields(0).ForeignName
      End With
   Next relBucle

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad IgnoreNulls

Este ejemplo establece la propiedad **IgnoreNulls** de un **Index** nuevo a **True** o **False** basándose en una entrada del usuario y demuestra el efecto sobre un objeto **Recordset** con un registro cuyo campo clave contiene un valor **Null**.

```
Sub IgnoreNullsX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim idxNuevo As Index
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno!Empleados

   With tdfEmpleados
      ' Crea un objeto Index nuevo.
      Set idxNuevo = .CreateIndex("ÍndiceNuevo")
      idxNuevo.Fields.Append idxNuevo.CreateField("País")

      ' Establece la propiedad IgnoreNulls del objeto
      ' Index nuevo basándose en la entrada del usuario.
      Select Case MsgBox("¿Desea establecer IgnoreNulls a True?", _
         vbYesNoCancel)
         Case vbYes
            idxNuevo.IgnoreNulls = True
         Case vbNo
            idxNuevo.IgnoreNulls = False
         Case Else
            dbsNeptuno.Close
            End
      End Select

      ' Anexa el objeto Index nuevo a la colección
      ' Indexes de la tabla Empleados.
      .Indexes.Append idxNuevo
      .Indexes.Refresh
   End With

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      ' Agrega un registro nuevo a la tabla Empleados.
      .AddNew
      !Nombre = "Pedro"
      !Apellidos = "Díaz Alonso"
      .Update

      ' Utiliza el índice nuevo para establecer el orden de los registros.
      .Index = idxNuevo.Name
      .MoveFirst

      Debug.Print "Índice = " & .Index & _
         ", IgnoreNulls = " & idxNuevo.IgnoreNulls
      Debug.Print "    País - Nombre"
```

```
      ' Enumera el Recordset. El valor de la propiedad
      ' IgnoreNulls determinará si el nuevo registro
      ' agregado aparece en el resultado.
      Do While Not .EOF
         Debug.Print "        " & _
            IIf(IsNull(!País), "[Null]", !País) & _
            " - " & !Nombre & " " & !Apellidos
         .MoveNext
      Loop

      ' Elimina el registro nuevo porque esto es un ejemplo.
      .Index = ""
      .Bookmark = .LastModified
      .Delete
      .Close
   End With

   ' Elimina el Index nuevo porque esto es un ejemplo.
   tdfEmpleados.Indexes.Delete idxNuevo.Name
   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Index

Este ejemplo utiliza la propiedad **Index** para establecer diferentes formas de ordenar registros de un objeto **Recordset** de tipo Table.

```
Sub IndexPropertyX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim rstEmpleados As Recordset
   Dim idxBucle As Index

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")
   Set tdfEmpleados = dbsNeptuno.TableDefs!Empleados

   With rstEmpleados

      ' Enumera la colección Indexes de la tabla Empleados.
      For Each idxBucle In tdfEmpleados.Indexes
         .Index = idxBucle.Name
         Debug.Print "Índice = " & .Index
         Debug.Print "    IdEmpleado - CódPostal - Nombre"
         .MoveFirst

         ' Enumera el Recordset para mostrar el orden de los registros.
         Do While Not .EOF
            Debug.Print "        " & !IdEmpleado & " - " & _
               !CódPostal & " - " & !Nombre & " " & _
               !Apellidos
            .MoveNext
         Loop

      Next idxBucle

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad LastModified

Este ejemplo utiliza la propiedad **LastModified** para mover el puntero del registro actual a un registro que se modificó y un registro creado recientemente.

```
Sub LastModifiedX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim strNombre As String
   Dim strApellido As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   With rstEmpleados
      ' Almacena los datos actuales.
      strNombre = !Nombre
      strApellido = !Apellidos
      ' Modifica los datos en el registro activo.
      .Edit
      !Nombre = "Julia"
      !Apellidos = "Martín Hernández"
      .Update
      ' Mueve el puntero del registro activo al
      ' registro modificado o agregado más recientemente.
      .Bookmark = .LastModified
      Debug.Print _
         "Datos en el registro LastModified después de Edit: " & _
         !Nombre & " " & !Apellidos

      ' Restaura los datos originales porque esto es un ejemplo.
      .Edit
      !Nombre = strNombre
      !Apellidos = strApellido
      .Update

      ' Agrega un registro nuevo.
      .AddNew
      !Nombre = "Pablo"
      !Apellidos = "Gómez Castaño"
      .Update
      ' Mueve el puntero del registro activo al
      ' registro modificado o agregado más recientemente.
      .Bookmark = .LastModified
      Debug.Print _
         " Datos en el registro LastModified después de AddNew: " & _
         !Nombre & " " & !Apellidos

      ' Elimina el registro nuevo porque esto es un ejemplo.
      .Delete
      .Close
   End With

   dbsNeptuno.Close
```

```
End Sub
```

Ejemplo de la propiedad LockEdits

Este ejemplo demuestra el bloqueo pesimista estableciendo la propiedad **LockEdits** a **True** y demuestra el bloqueo optimista estableciendo la propiedad **LockEdits** a **False**. También demuestra qué tipo de tratamiento de errores se necesita en un entorno de base de datos multiusuario para modificar un campo. Se necesitan las funciones BloqueoPesimista y BloqueoOptimista para ejecutar este procedimiento.

```
Sub LockEditsX()

   Dim dbsNeptuno As Database
   Dim rstClientes As Recordset
   Dim strNombreAntiguo As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstClientes = _
      dbsNeptuno.OpenRecordset("Clientes", _
      dbOpenDynaset)

   With rstClientes
      ' Almacena los datos originales.
      strNombreAntiguo = !NombreCompañía

      If MsgBox("Demostración de bloqueo pesimista...", _
            vbOKCancel) = vbOK Then

         ' Intenta modificar datos con el bloqueo
         ' pesimista activo.
         If BloqueoPesimista(rstClientes, !NombreCompañía, _
               "Antonio Moreno Taquería") Then
            MsgBox "Registro modificado con éxito."

            ' Restaura los datos originales...
            .Edit
            !NombreCompañía = strNombreAntiguo
            .Update
         End If

      End If

      If MsgBox("Demostración de bloqueo optimista...", _
            vbOKCancel) = vbOK Then

         ' Intenta modificar datos con el bloqueo
         ' optimista activo.
         If BloqueoOptimista(rstClientes, !NombreCompañía, _
               "Antonio Moreno Taquería") Then
            MsgBox "Registro modificado con éxito."

            ' Restaura los datos originales...
            .Edit
            !NombreCompañía = strNombreAntiguo
            .Update
         End If

      End If
```

```
      .Close
   End With

   dbsNeptuno.Close

End Sub

Function BloqueoPesimista(rstTemp As Recordset, _
   fldTemp As Field, strNueva As String) As Boolean

   dim ErrBucle as Error

   BloqueoPesimista = True

   With rstTemp
      .LockEdits = True

      ' Cuando establece LockEdits a True, intercepta
      ' los errores al llamar al método Edit.
      On Error GoTo Err_Bloqueo
      .Edit
      On Error GoTo 0

      ' Si Edit todavía está en ejecución, no se
      ' producen errores; puede modificar los datos.
      If .EditMode = dbEditInProgress Then
         fldTemp = strNueva
         .Update
         .Bookmark = .LastModified
      Else
         ' Recupera el registro activo para ver los
         ' cambios realizados por otro usuario.
         .Move 0
      End If

   End With

   Exit Function

Err_Bloqueo:

   If DBEngine.Errors.Count > 0 Then
      ' Enumera la colección Errors.
      For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & _
            vbCr & errBucle.Description
      Next errBucle
      BloqueoPesimista = False
   End If

   Resume Next

End Function

Function BloqueoOptimista(rstTemp As Recordset, _
   fldTemp As Field, strNueva As String) As Boolean
```

```
   dim ErrBucle as Error

   BloqueoOptimista = True

   With rstTemp
      .LockEdits = False
      .Edit
      fldTemp = strNueva

      ' Cuando establece LockEdits a False, intercepta
      ' los errores al llamar al método Update.
      On Error GoTo Err_Bloqueo
      .Update
      On Error GoTo 0

      ' Si Edit todavía está en ejecución, no se
      ' producen errores; puede modificar los datos.
      If .EditMode = dbEditNone Then
         ' Mueve el puntero del registro activo al
         ' registro modificado más recientemente.
         .Bookmark = .LastModified
      Else
         .CancelUpdate
         ' Recupera el registro activo para ver los
         ' cambios realizados por otro usuario.
         .Move 0
      End If

   End With

   Exit Function

Err_Bloqueo:

   If DBEngine.Errors.Count > 0 Then
      ' Enumera la colección Errors.
        For Each errBucle In DBEngine.Errors
         MsgBox "Número de error: " & errBucle.Number & _
            vbCr & errBucle.Description
      Next errBucle
      BloqueoOptimista = False
   End If

   Resume Next

End Function
```

Ejemplo de la propiedad NoMatch

Este ejemplo utiliza la propiedad **NoMatch** para determinar si un método **Seek** y un **FindFirst** tuvieron éxito y si no, ofrecer la información correspondiente. Se necesitan los procedimientos BuscarCoincidencia y EncontrarCoincidencia para ejecutar este procedimiento.

```
Sub NoMatchX()

   Dim dbsNeptuno As Database
   Dim rstProductos As Recordset
   Dim rstClientes As Recordset
   Dim strMensaje As String
   Dim strBuscar As String
   Dim strPaís As String
   Dim varMarcador As Variant

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' De modo predeterminado es dbOpenTable; se
   ' necesita si se va a utilizar la propiedad Index.
   Set rstProductos = dbsNeptuno.OpenRecordset("Productos")

   With rstProductos
      .Index = "ClavePrincipal"

      Do While True
         ' Muestra la información del registro activo;
         ' pide una entrada al usuario.
         strMensaje = "NoMatch con el método Seek" & vbCr & _
            "Id de producto: " & !IdProducto & vbCr & _
            "Nombre del Producto: " & !NombreProducto & vbCr & _
            "No coinciden = " & .NoMatch & vbCr & vbCr & _
            "Escriba un Id de producto."
         strBuscar = InputBox(strMensaje)
         If strBuscar = "" Then Exit Do

         ' Llama al procedimiento que busca un
         ' registro basado en el número de Id suministrado por el usuario.
         BuscarCoincidencia rstProductos, Val(strBuscar)
      Loop

      .Close
   End With

   Set rstClientes = dbsNeptuno.OpenRecordset( _
      "SELECT NombreCompañía, País FROM Clientes " & _
      "ORDER BY NombreCompañía", dbOpenSnapshot)

   With rstClientes

      Do While True
         ' Muestra la información del registro activo;
         ' pide al usuario una entrada.
         strMensaje = "NoMatch con el método FindFirst"& _
            vbCr & "Nombre del cliente: " & !NombreCompañía & _
            vbCr & "País: " & !País & vbCr & _
            "No coinciden = " & .NoMatch & vbCr & vbCr & _
            "Escriba el país que busca."
```

```
         strPaís = Trim(InputBox(strMensaje))
         If strPaís = "" Then Exit Do

         ' Llama al procedimiento que encuentra un
         ' registro basado en el nombre del país suministrado por el
usuario.
         EncontrarCoincidencia rstClientes, _
            "País = '" & strPaís & "'"
      Loop

      .Close
   End With

   dbsNeptuno.Close

End Sub

Sub BuscarCoincidencia(rstTemp As Recordset, _
   intBuscar As Integer)

   Dim varMarcador As Variant
   Dim strMensaje As String

   With rstTemp
      ' Almacena la ubicación del registro activo.
      varMarcador = .Bookmark
      .Seek "=", intBuscar

      ' Si falla el método Seek, se notifica al
      ' usuario y vuelve al último registro activo.
      If .NoMatch Then
         strMensaje = _
            "¡No se ha encontrado! Volviendo al registro activo." & _
            vbCr & vbCr & "No coinciden = " & .NoMatch
         MsgBox strMensaje
         .Bookmark = varMarcador
      End If

   End With

End Sub

Sub EncontrarCoincidencia(rstTemp As Recordset, _
   strEncontrar As String)

   Dim varMarcador As Variant
   Dim strMensaje As String

   With rstTemp
      ' Almacena la ubicación del registro activo.
      varMarcador = .Bookmark
      .FindFirst strEncontrar

      ' Si falla el método Seek, se notifica al
      ' usuario y vuelve al último registro activo.
      If .NoMatch Then
         strMensaje = _
```

```
            "¡No se ha encontrado! Volviendo al registro activo." & _
            vbCr & vbCr & " No coinciden = " & .NoMatch
         MsgBox strMensaje
         .Bookmark = varMarcador
      End If

   End With

End Sub
```

Ejemplo de la propiedad OrdinalPosition

Este ejemplo modifica el valor de la propiedad **OrdinalPosition** en el objeto **TableDef** de Empleados para controlar el orden del objeto **Field** en un objeto **Recordset** resultante. Estableciendo la propiedad **OrdinalPosition** de todos los objetos **Field** a 1, cualquier **Recordset** resultante ordenará la colección **Fields** alfabéticamente. Observe que el valor de **OrdinalPosition** en el **Recordset** no coincide con el valor en el **TableDef**, sino que simplemente refleja el resultado final de las modificaciones de **TableDef**.

```
Sub OrdinalPositionX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim mintPosición() As Integer
   Dim mstrNombreCampo() As String
   Dim intTemp As Integer
   Dim fldTemp As Field
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno.TableDefs("Empleados")

   With tdfEmpleados
      ' Muestra y almacena los datos originales de OrdinalPosition.
      Debug.Print _
         "Datos originales de OrdinalPosition en TableDef."
      ReDim mintPosición(0 To .Fields.Count - 1) As Integer
      ReDim mstrNombreCampo(0 To .Fields.Count - 1) As String
      For intTemp = 0 To .Fields.Count - 1
         mintPosición(intTemp) = _
            .Fields(intTemp).OrdinalPosition
         mstrNombreCampo(intTemp) = .Fields(intTemp).Name
         Debug.Print , mintPosición(intTemp), _
            mstrNombreCampo(intTemp)
      Next intTemp

      ' Modifica los datos de OrdinalPosition.
      For Each fldTemp In .Fields
         fldTemp.OrdinalPosition = 1
      Next fldTemp

      ' Abre un nuevo objeto Recordset para mostrar
      ' cómo los datos de OrdinalPosition han afectado el orden de los
registros.
      Debug.Print _
         "Datos de la propiedad OrdinalPosition del Recordset resultante."
      Set rstEmpleados = dbsNeptuno.OpenRecordset( _
         "SELECT * FROM Empleados")
      For Each fldTemp In rstEmpleados.Fields
         Debug.Print , fldTemp.OrdinalPosition, fldTemp.Name
      Next fldTemp
      rstEmpleados.Close

      ' Restaura los datos originales de
      ' OrdinalPosition porque esto es un ejemplo.
      For intTemp = 0 To .Fields.Count - 1
```

```
            .Fields(mstrNombreCampo(intTemp)).OrdinalPosition = _
               mintPosición(intTemp)
         Next intTemp

      End With

      dbsNeptuno.Close

   End Sub
```

Ejemplo de la propiedad PercentPosition

Este ejemplo utiliza la propiedad **PercentPosition** para mostrar la posición del puntero del registro actual con respecto al principio del objeto **Recordset**.

```
Sub PercentPositionX()

   Dim dbsNeptuno As Database
   Dim rstProductos As Recordset
   Dim strEncontrar As String
   Dim strMensaje As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' PercentPosition sólo funciona con objetos de tipo Dynaset o Snapshot.
   Set rstProductos = dbsNeptuno.OpenRecordset( _
      "SELECT NombreProducto FROM Productos " & _
      "ORDER BY NombreProducto", dbOpenSnapshot)

   With rstProductos
      ' Llena el Recordset.
      .MoveLast
      .MoveFirst

      Do While True
         ' Muestra información del registro activo y
         ' pide una entrada al usuario.
         strMensaje = "Producto: " & !NombreProducto & vbCr & _
            "El puntero del registro está a" & _
            Format(.PercentPosition, "##0.0") & _
            "% del " & vbCr & _
            "principio del Recordset." & vbCr & _
            "Escriba una cadena de caracteres de " & _
            "búsqueda del nombre del producto."
         strEncontrar = Trim(InputBox(strMensaje))
         If strEncontrar = "" Then Exit Do

         ' Intenta encontrar un registro que coincida con la cadena de
búsqueda.
         .FindFirst "NombreProducto >= '" & strEncontrar & "'"
         If .NoMatch Then .MoveLast
      Loop

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Primary

Este ejemplo utiliza la propiedad **Primary** para designar un objeto **Index** nuevo en un **TableDef** nuevo como **Index** principal para la tabla. Observe que establecer la propiedad **Primary** a **True** establece automáticamente las propiedades **Unique** y **Required** también a **True**.

```
Sub PrimaryX()

   Dim dbsNeptuno As Database
   Dim tdfNuevo As TableDef
   Dim idxNuevo As Index
   Dim idxBucle As Index
   Dim fldBucle As Field
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Crea y anexa un objeto TableDef nuevo a la
   ' colección TableDefs de la base de datos Neptuno.
   Set tdfNuevo = dbsNeptuno.CreateTableDef("NuevaTabla")
   tdfNuevo.Fields.Append tdfNuevo.CreateField("CampoNumérico", _
      dbLong, 20)
   tdfNuevo.Fields.Append tdfNuevo.CreateField("CampoTexto", _
      dbText, 20)
   dbsNeptuno.TableDefs.Append tdfNuevo

   With tdfNuevo
      ' Crea y anexa un objeto Index nuevo a la
      ' colección Indexes del objeto TableDef nuevo.
      Set idxNuevo = .CreateIndex("ÍndiceNumérico")
      idxNuevo.Fields.Append idxNuevo.CreateField("CampoNumérico")
      idxNuevo.Primary = True
      .Indexes.Append idxNuevo
      Set idxNuevo = .CreateIndex("ÍndiceTexto")
      idxNuevo.Fields.Append idxNuevo.CreateField("CampoTexto")
      .Indexes.Append idxNuevo

      Debug.Print .Indexes.Count & " Indexes en " & _
         .Name & " TableDef"

      ' Enumera la colección Indexes.
      For Each idxBucle In .Indexes

         With idxBucle
            Debug.Print "    " & .Name

            ' Enumera la colección Fields de cada
            ' objeto Index.
            Debug.Print "        Campos"
            For Each fldBucle In .Fields
               Debug.Print "            " & fldBucle.Name
            Next fldBucle

            ' Enumera la colección Properties de cada
            ' objeto Index.
            Debug.Print "        Propiedades"
            For Each prpBucle In .Properties
```

```
                Debug.Print "            " & prpBucle.Name & _
                   " = " & IIf(prpBucle = "", "[vacía]", _
                   prpBucle)
             Next prpBucle
          End With

       Next idxBucle

   End With

   dbsNeptuno.TableDefs.Delete tdfNuevo.Name
   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad RecordCount

Este ejemplo demuestra la propiedad **RecordCount** con tipos de **Recordsets** diferentes antes y después de llenarlos.

```
Sub RecordCountX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Abre un Recordset de tipo Table y muestra la
      ' propiedad RecordCount.
      Set rstEmpleados = .OpenRecordset("Empleados")
      Debug.Print _
         "Recordset de tipo Table de la tabla Empleados "
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount
      rstEmpleados.Close

      ' Abre un Recordset de tipo Dynaset y muestra
      ' la propiedad RecordCount antes de llenar el Recordset.
      Set rstEmpleados = .OpenRecordset("Empleados", _
         dbOpenDynaset)
      Debug.Print "Recordset de tipo Dynaset" & _
         "de la tabla Empleados antes de MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount

      ' Muestra la propiedad RecordCount después de
      ' llenar el Recordset.
      rstEmpleados.MoveLast
      Debug.Print "Recordset de tipo Dynaset " & _
         "de la tabla Empleados después de MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount
      rstEmpleados.Close

      ' Abre un Recordset de tipo Snapshot y muestra
      ' la propiedad RecordCount antes de llenar el Recordset.
      Set rstEmpleados = .OpenRecordset("Empleados", _
         dbOpenSnapshot)
      Debug.Print "Recordset de tipo Snapshot " & _
         "de la tabla Empleados antes de MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount

      ' Muestra la propiedad RecordCount después de
      ' llenar el Recordset.
      rstEmpleados.MoveLast
      Debug.Print "Recordset de tipo Snapshot " & _
         "de la tabla Empleados después de MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount
      rstEmpleados.Close
```

```
      ' Abre un Recordset de tipo Forward-only y
      ' muestra la propiedad RecordCount antes de
      ' llenar el Recordset.
      Set rstEmpleados = .OpenRecordset("Empleados", _
         dbOpenForwardOnly)
      Debug.Print "Recordset de tipo Forward-only " & _
         "de la tabla Empleados antes de MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount

      ' Muestra la propiedad RecordCount después de
      ' llamar al método MoveNext.
      rstEmpleados.MoveNext
      Debug.Print "Recordset de tipo Forward-only " & _
         "de la tabla Empleados después de MoveNext"
      Debug.Print "    RecordCount = " & _
         rstEmpleados.RecordCount
      rstEmpleados.Close

      .Close
   End With

End Sub
```

Ejemplo de la propiedad RecordsAffected

Este ejemplo utiliza la propiedad **RecordsAffected** con consultas de acciones ejecutadas desde objeto **Database** y un objeto **QueryDef**. Se necesita la función SalidaRegistrosAfectados para ejecutar este procedimiento.

```
Sub RecordsAffectedX()

   Dim dbsNeptuno As Database
   Dim qdfTemp As QueryDef
   Dim strSQLCambiar As String
   Dim strSQLRestaurar As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Imprime un informe del contenido de la tabla
      ' Empleados.
      Debug.Print _
         "Número de registros de la tabla Empleados: " & _
         .TableDefs!Empleados.RecordCount
      SalidaRegistrosAfectados dbsNeptuno

      ' Define y ejecuta una consulta de acciones.
      strSQLCambiar = "UPDATE Empleados " & _
         "SET País = 'Estados Unidos' " & _
         "WHERE País = 'EE.UU.'"
      .Execute strSQLCambiar

      ' Imprime un informe del contenido de la tabla
      ' Empleados.
      Debug.Print _
         "RecordsAffected después de ejecutar la consulta " & _
         "de Database: " & .RecordsAffected
      SalidaRegistrosAfectados dbsNeptuno

      ' Define y ejecuta otra consulta de acciones.
      strSQLRestaurar = "UPDATE Empleados " & _
         "SET País = 'EE.UU.' " & _
         "WHERE País = 'Estados Unidos'"
      Set qdfTemp = .CreateQueryDef("", strSQLRestaurar)
      qdfTemp.Execute

      ' Imprime un informe del contenido de la tabla
      ' Empleados.
      Debug.Print _
         "RecordsAffected después de ejecutar la consulta " & _
         "de QueryDef: " & qdfTemp.RecordsAffected
      SalidaRegistrosAfectados dbsNeptuno

      .Close

   End With

End Sub

Function SalidaRegistrosAfectados(dbsNeptuno As Database)
```

```
    Dim rstEmpleados As Recordset

    ' Abre un objeto Recordset de la tabla Empleados.
    Set rstEmpleados = _
       dbsNeptuno.OpenRecordset("Empleados")

    With rstEmpleados
       ' Enumera el Recordset.
       .MoveFirst
       Do While Not .EOF
          Debug.Print "    " & !Apellidos & ", " & !País
          .MoveNext
       Loop
       .Close
    End With

End Function
```

Ejemplo de la propiedad Restartable

Este ejemplo demuestra la propiedad **Restartable** con objetos **Recordset** diferentes.

```
Sub RestartableX()

   Dim dbsNeptuno As Database
   Dim rstTemp As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Abre un Recordset de tipo Table e imprime su
      ' propiedad Restartable.
      Set rstTemp = .OpenRecordset("Empleados", dbOpenTable)
      Debug.Print _
         "Recordset de tipo Table de la tabla Empleados"
      Debug.Print "    Restartable = " & rstTemp.Restartable
      rstTemp.Close

      ' Abre un Recordset desde una instrucción SQL e
      ' imprime su propiedad Restartable.
      Set rstTemp = _
         .OpenRecordset("SELECT * FROM Empleados")
      Debug.Print "Recordset basado en una instrucción SQL"
      Debug.Print "    Restartable = " & rstTemp.Restartable
      rstTemp.Close

      ' Abre un Recordset a partir de un objeto
      ' QueryDef guardado e imprime su propiedad Restartable.
      Set rstTemp = .OpenRecordset("Lista actual de productos ")
      Debug.Print _
         "Recordset basado en un QueryDef permanente (" & _
         rstTemp.Name & ")"
      Debug.Print "    Restartable = " & rstTemp.Restartable
      rstTemp.Close

      .Close
   End With

End Sub
```

Ejemplo de la propiedad Size

Este ejemplo demuestra la propiedad **Size** enumerando los nombres y los tamaños de los objetos **Field** en la tabla Empleados.

```
Sub SizeX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim fldNuevo As Field
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno.TableDefs!Empleados

   With tdfEmpleados

      ' Crea y anexa un objeto Field nuevo a la tabla
      ' Empleados.
      Set fldNuevo = .CreateField("Fax")
      fldNuevo.Type = dbText
      fldNuevo.Size = 20
      .Fields.Append fldNuevo

      Debug.Print "TableDef: " & .Name
      Debug.Print "    Field.Name - Field.Type - Field.Size"

      ' Enumera la colección Fields; imprime los
      ' nombres de los campos, los tipos y los tamaños.
      For Each fldBucle In .Fields
         Debug.Print "        " & fldBucle.Name & " - " & _
            fldBucle.Type & " - " & fldBucle.Size
      Next fldBucle

      ' Elimina el campo nuevo porque esto es un ejemplo.
      .Fields.Delete fldNuevo.Name

   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Sort

Este ejemplo demuestra la propiedad **Sort** modificando su valor y creando un objeto **Recordset** nuevo. Se necesita la función SalidaOrdenar para ejecutar este procedimiento.

```
Sub SortX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim rstClasificarEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)

   With rstEmpleados
      SalidaOrdenar "Recordset original:", rstEmpleados
      .Sort = "Apellidos, Nombre"
      ' Imprime un informe mostrando la propiedad Sort y el orden de los
registros.
      SalidaOrdenar _
         "Recordset después de modificar la propiedad Sort:", _
         rstEmpleados
      ' Abre un Recordset nuevo a partir del actual.
      Set rstClasificarEmpleados = .OpenRecordset
      ' Imprime un informe que muestra la propiedad Sort y el orden de los
registros.
      SalidaOrdenar "Recordset nuevo:", rstClasificarEmpleados
      rstClasificarEmpleados.Close
      .Close
   End With

   dbsNeptuno.Close

End Sub

Function SalidaOrdenar(strTemp As String, _
   rstTemp As Recordset)

   With rstTemp
      Debug.Print strTemp
      Debug.Print "    Ordenar = " & _
         IIf(.Sort <> "", .Sort, "[Vacío]")
      .MoveFirst

      ' Enumera el Recordset.
      Do While Not .EOF
         Debug.Print "        " & !Apellidos & _
            ", " & !Nombre
         .MoveNext
      Loop

   End With

End Function
```

Nota Cuando conoce los datos que desea seleccionar, normalmente es mejor crear un objeto **Recordset** con una instrucción SQL. Este ejemplo muestra cómo puede crear solamente un **Recordset** y obtener el mismo resultado que en el ejemplo anterior.

```
Sub SortX2()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Abre un Recordset desde una instrucción SQL que
   ' especifica un orden.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("SELECT * " & _
      "FROM Empleados ORDER BY Apellidos, Nombre", _
      dbOpenDynaset)

   dbsNeptuno.Close

End Sub
```

Ejemplo de las propiedades SourceField y SourceTable

Este ejemplo demuestra las propiedades **SourceField** y **SourceTable** abriendo un objeto **Recordset** formado por campos de dos tablas.

```
Sub SourceFieldX()

   Dim dbsNeptuno As Database
   Dim rstCategoríaProducto As Recordset
   Dim fldBucle As Field
   Dim strSQL As String

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Abre un Recordset desde una instrucción SQL
   ' que utiliza campos de dos tablas diferentes.
   strSQL = "SELECT IdProducto AS IdProd, " & _
      "NombreProducto AS NombreProd, " & _
      "Categorías.IdCategoría AS IdCat, " & _
      "NombreCategoría AS NombreCat " & _
      "FROM Categorías INNER JOIN Productos ON " & _
      "Categorías.IdCategoría = Productos.IdCategoría " & _
      "ORDER BY NombreProducto"
   Set rstCategoríaProducto = _
      dbsNeptuno.OpenRecordset(strSQL)

   Debug.Print "Field - SourceTable - SourceField"
   ' Enumera la colección Fields del Recordset,
   ' imprimiendo el nombre, la tabla original y el nombre original.
   For Each fldBucle In rstCategoríaProducto.Fields
      Debug.Print "    " & fldBucle.Name & " - " & _
         fldBucle.SourceTable & " - " & fldBucle.SourceField
   Next fldBucle

   rstCategoríaProducto.Close
   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad SQL

Este ejemplo demuestra la propiedad **SQL** estableciendo y modificando la propiedad **SQL** de un **QueryDef** temporal y comparando los resultados. Se necesita la función SalidaSQL para ejecutar este procedimiento.

```
Sub SQLX()

   Dim dbsNeptuno As Database
   Dim qdfTemp As QueryDef
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set qdfTemp = dbsNeptuno.CreateQueryDef("")

   ' Abre un Recordset utilizando un objeto QueryDef
   ' temporal e imprime un informe.
   SalidaSQL "SELECT * FROM Empleados " & _
      "WHERE País = 'España' " & _
      "ORDER BY Apellidos", qdfTemp

   ' Abre un Recordset utilizando un objeto QueryDef
   ' temporal e imprime un informe.
   SalidaSQL "SELECT * FROM Empleados " & _
      "WHERE País = 'México' " & _
      "ORDER BY Apellidos", qdfTemp

   dbsNeptuno.Close

End Sub

Function SalidaSQL(strSQL As String, qdfTemp As QueryDef)

   Dim rstEmpleados As Recordset

   ' Establece la propiedad SQL del objeto QueryDef
   ' temporal y abre un Recordset.
   qdfTemp.SQL = strSQL
   Set rstEmpleados = qdfTemp.OpenRecordset

   Debug.Print strSQL

   With rstEmpleados
      ' Enumera el Recordset.
      Do While Not .EOF
         Debug.Print "    " & !Nombre & " " & _
            !Apellidos & ", " & !País
         .MoveNext
      Loop
      .Close
   End With

End Function
```

Ejemplo de la propiedad Type

Este ejemplo demuestra la propiedad **Type** devolviendo el nombre de la constante correspondiente al valor de la propiedad **Type** de cuatro **Recordsets** diferentes. Se necesita la función TipoRecordset para ejecutar este procedimiento.

```
Sub TypeX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' De modo predeterminado es dbOpenTable.
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")
   Debug.Print _
      "Recordset de tipo Table (tabla Empleados): " & _
      TipoRecordset(rstEmpleados.Type)
   rstEmpleados.Close

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenDynaset)
   Debug.Print _
      "Recordset de tipo Dynaset (tabla Empleados): " & _
      TipoRecordset(rstEmpleados.Type)
   rstEmpleados.Close

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenSnapshot)
   Debug.Print _
      "Recordset de tipo Snapshot (tabla Empleados): " & _
      TipoRecordset(rstEmpleados.Type)
   rstEmpleados.Close

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados", _
      dbOpenForwardOnly)
   Debug.Print _
      "Recordset de tipo Forward-only (tabla Empleados): " & _
      TipoRecordset(rstEmpleados.Type)
   rstEmpleados.Close

   dbsNeptuno.Close

End Sub

Function TipoRecordset(intTipo As Integer) As String

   Select Case intTipo
      Case dbOpenTable
         TipoRecordset = "dbOpenTable"
      Case dbOpenDynaset
         TipoRecordset = "dbOpenDynaset"
      Case dbOpenSnapshot
```

```
         TipoRecordset = "dbOpenSnapshot"
      Case dbOpenForwardOnly
         TipoRecordset = "dbOpenForwardOnly"
   End Select

End Function
```

Este ejemplo demuestra la propiedad **Type** devolviendo el nombre de la constante correspondiente al valor de la propiedad **Type** de todos los objetos **Field** en la tabla Empleados. Se necesita la función TipoCampo para ejecutar este procedimiento.

```
Sub TypeX2()

   Dim dbsNeptuno As Database
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   Debug.Print "Objetos Field en TableDef Empleados:"
   Debug.Print "    Tipo - Nombre"

   ' Enumera la colección Fields de la tabla Empleados.
   For Each fldBucle In _
      dbsNeptuno.TableDefs!Empleados.Fields
      Debug.Print "        " & TipoCampo(fldBucle.Type) & _
         " - " & fldBucle.Name
   Next fldBucle

   dbsNeptuno.Close

End Sub

Function TipoCampo(intTipo As Integer) As String

   Select Case intTipo
      Case dbBoolean
         TipoCampo = "dbBoolean"
      Case dbByte
         TipoCampo = "dbByte"
      Case dbInteger
         TipoCampo = "dbInteger"
      Case dbLong
         TipoCampo = "dbLong"
      Case dbCurrency
         TipoCampo = "dbCurrency"
      Case dbSingle
         TipoCampo = "dbSingle"
      Case dbDouble
         TipoCampo = "dbDouble"
      Case dbDate
         TipoCampo = "dbDate"
      Case dbText
         TipoCampo = "dbText"
      Case dbLongBinary
         TipoCampo = "dbLongBinary"
      Case dbMemo
         TipoCampo = "dbMemo"
      Case dbGUID
```

```
         TipoCampo = "dbGUID"
   End Select

End Function
```

Este ejemplo demuestra la propiedad **Type** devolviendo el nombre de la constante correspondiente al valor de la propiedad **Type** de todos los objetos **QueryDef** en Neptuno. Se necesita la función TipoDefConsulta para ejecutar este procedimiento.

```
Sub TypeX3()

   Dim dbsNeptuno As Database
   Dim qdfBucle As QueryDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   Debug.Print "QueryDefs en la base de datos Neptuno:"
   Debug.Print "    Tipo - Nombre"

   ' Enumera la colección QueryDefs de la base de datos Neptuno.
   For Each qdfBucle In dbsNeptuno.QueryDefs
      Debug.Print "        " & _
         TipoDefConsulta(qdfBucle.Type) & " - " & qdfBucle.Name
   Next qdfBucle

   dbsNeptuno.Close

End Sub

Function TipoDefConsulta(intTipo As Integer) As String

   Select Case intTipo
      Case dbQSelect
         TipoDefConsulta = "dbQSelect"
      Case dbQAction
         TipoDefConsulta = "dbQAction"
      Case dbQCrosstab
         TipoDefConsulta = "dbQCrosstab"
      Case dbQDelete
         TipoDefConsulta = "dbQDelete"
      Case dbQUpdate
         TipoDefConsulta = "dbQUpdate"
      Case dbQAppend
         TipoDefConsulta = "dbQAppend"
      Case dbQMakeTable
         TipoDefConsulta = "dbQMakeTable"
      Case dbQDDL
         TipoDefConsulta = "dbQDDL"
      Case dbQSQLPassThrough
         TipoDefConsulta = "dbQSQLPassThrough"
      Case dbQSetOperation
         TipoDefConsulta = "dbQSetOperation"
      Case dbQSPTBulk
         TipoDefConsulta = "dbQSPTBulk"
   End Select

End Function
```

Ejemplo de la propiedad Unique

Este ejemplo establece la propiedad **Unique** de un objeto **Index** nuevo a **True** y anexa el **Index** a la colección **Indexes** de la tabla Empleados. Después enumera la colección **Indexes** del objeto **TableDef** y la colección **Properties** de cada **Index**. El **Index** nuevo sólo permitirá un registro con una combinación concreta de País, Apellidos y Nombre en el objeto **TableDef**.

```
Sub UniqueX()

   Dim dbsNeptuno As Database
   Dim tdfEmpleados As TableDef
   Dim idxNuevo As Index
   Dim idxBucle As Index
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfEmpleados = dbsNeptuno!Empleados

   With tdfEmpleados
      ' Crea y anexa un objeto Index nuevo a la
      ' colección Indexes de la tabla Empleados.
      Set idxNuevo = .CreateIndex("NewIndex")

      With idxNuevo
         .Fields.Append .CreateField("País")
         .Fields.Append .CreateField("Apellidos")
         .Fields.Append .CreateField("Nombre")
         .Unique = True
      End With

      .Indexes.Append idxNuevo
      .Indexes.Refresh

      Debug.Print .Indexes.Count & " Indexes en " & _
         .Name & " TableDef"

      ' Enumera la colección Indexes de la tabla Empleados.
      For Each idxBucle In .Indexes
         Debug.Print "    " & idxBucle.Name

         ' Enumera la colección Properties de cada
         ' Index object.
         For Each prpBucle In idxBucle.Properties
            Debug.Print "        " & prpBucle.Name & _
               " = " & IIf(prpBucle = "", "[vacía]", prpBucle)
         Next prpBucle

      Next idxBucle

      ' Elimina el Index nuevo porque esto es un ejemplo.
      .Indexes.Delete idxNuevo.Name
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Updatable

Este ejemplo demuestra la propiedad **Updatable** para un objeto **Database**, cuatro tipos de objetos **Recordset**, un objeto **TableDef** y un objeto **QueryDef**.

```
Sub UpdatableX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      Debug.Print .Name
      Debug.Print "    Actualizable = " & .Updatable

      ' De modo predeterminado es dbOpenTable.
      Set rstEmpleados = .OpenRecordset("Empleados")
      Debug.Print _
         "Recordset de tipo Table de la tabla Empleados"
      Debug.Print "    Actualizable = " & _
         rstEmpleados.Updatable
      rstEmpleados.Close

      Set rstEmpleados = .OpenRecordset("Empleados", _
         dbOpenDynaset)
      Debug.Print _
         "Recordset de tipo Dynaset de la tabla Empleados"
      Debug.Print "    Actualizable = " & _
         rstEmpleados.Updatable
      rstEmpleados.Close

      Set rstEmpleados = .OpenRecordset("Empleados", _
         dbOpenSnapshot)
      Debug.Print _
         "Recordset de tipo Snapshot de la tabla Empleados"
      Debug.Print "    Actualizable = " & _
         rstEmpleados.Updatable
      rstEmpleados.Close

      Set rstEmpleados = .OpenRecordset("Empleados", _
         dbOpenForwardOnly)
      Debug.Print _
         "Recordset de tipo Forward-only de la tabla Empleados"
      Debug.Print "    Actualizable = " & _
         rstEmpleados.Updatable
      rstEmpleados.Close

      Debug.Print "'" & .TableDefs(0).Name & "' TableDef"
      Debug.Print "    Actualizable = " & _
         .TableDefs(0).Updatable

      Debug.Print "'" & .QueryDefs(0).Name & "' QueryDef"
      Debug.Print "    Actualizable = " & _
         .QueryDefs(0).Updatable

      .Close
```

```
    End With

End Sub
```

Ejemplo de la propiedad V1xNullBehavior

Este ejemplo convierte un archivo de base de datos Microsoft Jet versión 1.1 a un archivo de base de datos Microsoft Jet versión 3.0. Durante la conversión, se crea la propiedad **V1xNullBehavior** y se agrega a la colección **Properties** de la base de datos nueva. Las colecciones **Properties** de ambas bases de datos se enumeran para mostrar la modificación. Finalmente, se elimina la propiedad **V1xNullBehavior**. Esto supone que cualquier aplicación se modificará para almacenar valores **Null** en campos de tipo Text y Memo vacíos en vez de cadenas vacías.

Nota A menos que pueda obtener un archivo de Microsoft Jet versión 1.1 llamado "Neptun11.mdb," este procedimiento no se ejecutará.

```
Sub V1xNullBehaviorX()

   Dim dbsNeptuno As Database
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptun11.mdb")

   With dbsNeptuno
      Debug.Print .Name & ", versión " & .Version
      ' Enumera la colección Properties de la base de
      ' datos Neptuno.
      For Each prpBucle In .Properties
         On Error Resume Next
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
         On Error GoTo 0
      Next prpBucle

      .Close
   End With

   DBEngine.CompactDatabase "Neptun11.mdb", _
      "Neptun30.mdb", , dbVersion30

   Set dbsNeptuno = OpenDatabase("Neptun30.mdb")

   With dbsNeptuno
      Debug.Print .Name & ", versión " & .Version

      ' Enumera la colección Properties de la base de
      ' datos compactada. No se puede referir a la
      ' propiedad V1xNullBehavior explícitamente, es
      ' decir, dbsNeptuno.V1xNullBehavior, pero se
      ' puede tener acceso en bucles o por una cadena
      ' de referencia, es decir, dbsNeptuno.Properties("V1xNullBehavior").
      For Each prpBucle In .Properties
         On Error Resume Next
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle
         On Error GoTo 0
      Next prpBucle

      .Properties.Delete "V1xNullBehavior"
      .Close
   End With
```

```
End Sub
```

Ejemplo de la propiedad ValidateOnSet

Este ejemplo utiliza la propiedad **ValidateOnSet** para demostrar cómo se podrían interceptar los errores durante la entrada de datos. Se necesita la función ValidarDatos para ejecutar este procedimiento.

```
Sub ValidateOnSetX()

   Dim dbsNeptuno As Database
   Dim fldDías As Field
   Dim rstEmpleados As Recordset

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Crea y anexa un objeto Field nuevo a la colección
   ' Fields de la tabla Empleados.
   Set fldDías = _
      dbsNeptuno.TableDefs!Empleados.CreateField( _
      "DíasDeVacaciones", dbInteger, 2)
   fldDías.ValidationRule = "BETWEEN 1 AND 20"
   fldDías.ValidationText = _
      "¡El número debe estar entre 1 y 20!"
   dbsNeptuno.TableDefs!Empleados.Fields.Append fldDías

   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados

      Do While True
         ' Agrega un registro nuevo.
         .AddNew

         ' Obtiene una entrada del usuario para tres campos.
         ' Comprueba que los datos no infringen las reglas
         ' de validación para ninguno de los campos.
         If ValidarDatos(!Nombre, _
            "Introduzca el nombre.") = False Then Exit Do
         If ValidarDatos(!Apellidos, _
            "Introduzca los apellidos.") = False Then
         If ValidarDatos(!DíasDeVacaciones, _
            "Introduzca los días de vacaciones.") = False Then Exit Do

         .Update
         .Bookmark = .LastModified
         Debug.Print !Nombre & " " & !Apellidos & _
            " - " & "DíasDeVacaciones = " & !DíasDeVacaciones

         ' Elimina el registro nuevo porque esto es un ejemplo.
         .Delete
         Exit Do
      Loop

      ' Cancela el método AddNew si se infringe alguna
      ' regla de validación.
      If .EditMode <> dbEditNone Then .CancelUpdate
      .Close
```

```
    End With

    ' Elimina el campo nuevo porque esto es un ejemplo.
    dbsNeptuno.TableDefs!Empleados.Fields.Delete _
       fldDías.Name
    dbsNeptuno.Close

End Sub

Function ValidarDatos(fldTemp As Field, _
    strMensaje As String) As Boolean

    Dim strEntrada As String
    Dim errBucle As Error

    ValidarDatos = True
    ' ValidateOnSet es de sólo lectura y escritura para
    ' los objetos Field en los objetos Recordset.
    fldTemp.ValidateOnSet = True

    Do While True
       strEntrada = InputBox(strMensaje)
       If strEntrada = "" Then Exit Do
       ' Intercepta los errores cuando se establece el valor de Field.
       On Error GoTo Err_Datos
       If fldTemp.Type = dbInteger Then
          fldTemp = Val(strEntrada)
       Else
          fldTemp = strEntrada
       End If
       On Error GoTo 0
       If Not IsNull(fldTemp) Then Exit Do
    Loop

    If strEntrada = "" Then ValidarDatos = False

    Exit Function

Err_Datos:

    If DBEngine.Errors.Count > 0 Then
       ' Enumera la colección Errors. La propiedad
       ' Description del último objeto Error se
       ' establecerá a la propiedad ValidationText
       ' del campo relacionado.
       For Each errBucle In DBEngine.Errors
          MsgBox "Número de error: " & errBucle.Number & _
             vbCr & errBucle.Description
       Next errBucle
    End If

    Resume Next

End Function
```

Ejemplo de las propiedades ValidationRule y ValidationText

Este ejemplo crea un objeto **Field** nuevo en el objeto **TableDef** especificado y establece las propiedades **ValidationRule** y **ValidationText** basadas en los datos transferidos. También muestra cómo las propiedades **ValidationRule** y **ValidationText** se utilizan durante la entrada de datos real. Se necesita la propiedad EstablecerValidación para ejecutar este procedimiento.

```
Sub ValidationRuleX()

   Dim dbsNeptuno As Database
   Dim fldDías As Field
   Dim rstEmpleados As Recordset
   Dim strMensaje As String
   Dim strDías As String
   Dim errBucle As Error

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Crea un campo nuevo en el objeto TableDef de la
   ' tabla Empleados utilizando los valores de la propiedad especificada.
   Set fldDías = _
      EstablecerValidación(dbsNeptuno.TableDefs!Empleados, _
      "DíasDeVacaciones", dbInteger, 2, "BETWEEN 1 AND 20", _
      "¡El número debe estar entre 1 y 20!")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados

      ' Enumera el Recordset. Con cada registro, llena
      ' el campo nuevo con datos suministrados por el usuario.
      Do While Not .EOF
         .Edit
         strMensaje = "Introduzca los días de vacaciones para " & _
            !Nombre & " " & !Apellidos & vbCr & _
            "[" & !DíasDeVacaciones.ValidationRule & "]"

         Do While True
            ' Obtiene una entrada del usuario.
            strDías = InputBox(strMensaje)
            If strDías = "" Then
               .CancelUpdate
               Exit Do
            End If
            !DíasDeVacaciones = Val(strDías)

            ' Como ValidateOnSet tiene como valor
            ' predeterminado False, los datos en el
            ' búfer se comprobarán con ValidationRule durante el Update.
            On Error GoTo Err_Regla
            .Update
            On Error GoTo 0

            ' Si el método Update tuvo éxito, imprime
            ' el resultado del cambio de los datos.
            If .EditMode = dbEditNone Then
               Debug.Print !Nombre & " " & !Apellidos & _
                   " - " & "DíasDeVacaciones = " & _
```

```
                    !DíasDeVacaciones
                Exit Do
            End If

            Loop

            If strDías = "" Then Exit Do
            .MoveNext
        Loop

        .Close
    End With

    ' Elimina el campo nuevo porque esto es un ejemplo.
    dbsNeptuno.TableDefs!Empleados.Fields.Delete _
        fldDías.Name
    dbsNeptuno.Close

    Exit Sub

Err_Regla:

    If DBEngine.Errors.Count > 0 Then
        ' Enumera la colección Errors.
        For Each errBucle In DBEngine.Errors
            MsgBox "Número de error: " & _
                errBucle.Number & vbCr & _
                errBucle.Description
        Next errBucle
    End If

    Resume Next

End Sub

Function EstablecerValidación(tdfTemp As TableDef, _
    strNombreCampo As String, intTipo As Integer, _
    intLongitud As Integer, strRegla As String, _
    strTexto As String) As Field

    ' Crea y anexa un objeto Field nuevo a la colección
    ' Fields del objeto TableDef especificado.
    Set EstablecerValidación = tdfTemp.CreateField(strNombreCampo, _
        intTipo, intLongitud)

    EstablecerValidación.ValidationRule = strRegla
    EstablecerValidación.ValidationText = strTexto
    tdfTemp.Fields.Append EstablecerValidación

End Function
```

Ejemplo de la propiedad Value

Este ejemplo demuestra la propiedad **Value** con los objetos **Field** y **Property**.

```
Sub ValueX()

   Dim dbsNeptuno As Database
   Dim rstEmpleados As Recordset
   Dim fldBucle As Field
   Dim prpBucle As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set rstEmpleados = _
      dbsNeptuno.OpenRecordset("Empleados")

   With rstEmpleados
      Debug.Print "Valores de campo en rstEmpleados"
      ' Enumera la colección Fields de la tabla
      ' Empleados.
      For Each fldBucle In .Fields
         Debug.Print "    " & fldBucle.Name & " = ";
         Select Case fldBucle.Type
            Case dbLongBinary
               Debug.Print "[LongBinary]"
            Case dbMemo
               Debug.Print "[Memo]"
            Case Else
               ' Como Value es la propiedad
               ' predeterminada de un objeto Field,
               ' el uso de la clave real aquí es opcional.
               Debug.Print fldBucle.Value
         End Select
      Next fldBucle

      Debug.Print "Valores de Property en rstEmpleados"
      ' Enumera la colección Properties del objeto
      ' Recordset.
      For Each prpBucle In .Properties
         On Error Resume Next
         ' Como Value es la propiedad predeterminada
         ' de un objeto Property, es opcional utilizar aquí
         ' la clave activa.
         If prpBucle <> "" Then Debug.Print "    " & _
            prpBucle.Name & " = " & prpBucle.Value
         On Error GoTo 0
      Next prpBucle

      .Close
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplos de las propiedades AllPermissions, Permissions y SystemDB

Este ejemplo utiliza las propiedades **SystemDB**, **AllPermissions** y **Permissions** para mostrar que los usuarios puede tener niveles de permiso diferentes dependiendo de los permisos del grupo al que pertenecen.

```
Sub AllPermissionsX()

   ' Comprueba que el archivo de grupo de trabajo Microsoft Jet
   ' está disponible.
   DBEngine.SystemDB = "system.mdw"

   Dim dbsNeptuno As Database
   Dim ctrBucle As Container

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Enumera la colección Containers y muestra el usuario
   ' activo y el conjunto de permisos del mismo.
   For Each ctrBucle In dbsNeptuno.Containers
      With ctrBucle
         Debug.Print "Container: " & .Name
         Debug.Print "User: " & .UserName
         Debug.Print "    Permissions: " & .Permissions
         Debug.Print "    AllPermissions: " & _
            .AllPermissions
      End With
   Next ctrBucle

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Attributes

Este ejemplo muestra la propiedad **Attributes** de los objetos **Field**, **Relation** y **TableDef** en la base de datos Neptuno.

```
Sub AttributesX()

   Dim dbsNeptuno As Database
   Dim fldBucle As Field
   Dim relBucle As Relation
   Dim tdfBucle As TableDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno

      ' Muestra los atributos de los campos del objeto
      ' TableDef.
      Debug.Print "Atributos de los campos en la tabla " & _
         .TableDefs(0).Name
      For Each fldBucle In .TableDefs(0).Fields
         Debug.Print "    " & fldBucle.Name & " = " & _
            fldBucle.Attributes
      Next fldBucle

      ' Muestra los atributos de las relaciones de la
      ' base de datos Neptuno.
      Debug.Print "Atributos de las relaciones en " & _
         .Name & ":"
      For Each relBucle In .Relations
         Debug.Print "    " & relBucle.Name & " = " & _
            relBucle.Attributes
      Next relBucle

      ' Muestra los atributos de las tablas de la base
      ' de datos Neptuno.
      Debug.Print " Atributos de las tablas en " & .Name & ":"
      For Each tdfBucle In .TableDefs
         Debug.Print "    " & tdfBucle.Name & " = " & _
            tdfBucle.Attributes
      Next tdfBucle

      .Close
   End With

End Sub
```

Ejemplo de la propiedad BatchCollisionCount y el método Update

Este ejemplo utiliza la propiedad **BatchCollisionCount** y el método **Update** para demostrar la actualización por lotes donde las colisiones se resolverán forzando la actualización por lotes.

```
Sub BatchX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim rstTemp As Recordset
   Dim intBucle As Integer
   Dim strSolicitud As String

   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' Se necesita este valor de DefaultCursorDriver
   ' para la actualización por lotes.
   wrkPrincipal.DefaultCursorDriver = dbUseClientBatchCursor

   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")
   ' Se necesita el siguiente argumento de bloqueo para
   ' la actualización por lotes.
   Set rstTemp = conPrincipal.OpenRecordset( _
      "SELECT * FROM tipoimpuesto", dbOpenDynaset, 0, _
      dbOptimisticBatch)

   With rstTemp
      ' Modifica los datos en el conjunto de registros local.
      Do While Not .EOF
         .Edit
         If !impuesto <= 20 Then
            !impuesto = !impuesto - 4
         Else
            !impuesto = !impuesto + 2
         End If
         .Update
         .MoveNext
      Loop

      ' Intenta una actualización por lotes.
      .Update dbUpdateBatch

      ' Si hay colisiones, permite al usuario la opción
      ' de obligar los cambios o solucionarlos
      ' individualmente.
      If .BatchCollisionCount > 0 Then
         strSolicitud = "Hay colisiones. " & vbCr & _
            "¿Desea que el programa fuerce una actualización " & _
            vbCr & " utilizando los datos locales?"
         If MsgBox(strSolicitud, vbYesNo) = vbYes Then
            .Update dbUpdateBatch, True
      End If

      .Close
   End With
```

```
    conPrincipal.Close
    wrkPrincipal.Close

End Sub
```

Ejemplo de las propiedades BatchSize y UpdateOptions

Este ejemplo utiliza las propiedades **BatchSize** y **UpdateOptions** para controlar las apariencias de cualquier actualización por lotes para el objeto **Recordset** especificado.

```
Sub BatchSizeX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim rstTemp As Recordset

   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' Se necesita este valor de DefaultCursorDriver para la
   ' actualización por lotes.
   wrkPrincipal.DefaultCursorDriver = dbUseClientBatchCursor

   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")
   ' Se necesita el siguiente argumento de bloqueo para la
   ' actualización por lotes.
   Set rstTemp = conPrincipal.OpenRecordset( _
      "SELECT * FROM tipoimpuesto", dbOpenDynaset, 0, _
      dbOptimisticBatch)

   With rstTemp
      ' Aumenta el número de instrucciones enviadas al servidor
      ' durante una actualización por lotes sencilla, así disminuye el
      ' número de veces que una actualización tendría que tener acceso
      ' al servidor.
      .BatchSize = 25

      ' Cambia la propiedad UpdateOptions para que la cláusula WHERE
      ' de las instrucciones por lotes que se dirijan al servidor
      ' incluyan las columnas actualizadas además de la(s)
      ' columna(s) de clave. También, cualquier modificación a registros
      ' se hará eliminando el registro original y agregando una versión
      ' modificada mientras que sólo se modifica el
      ' registro original.
      .UpdateOptions = dbCriteriaModValues + _
         dbCriteriaDeleteInsert

      ' Llama a la actualización por lotes utilizando los valores
      ' nuevos descritos anteriormente.
      ' ...

      .Close
   End With

   conPrincipal.Close
   wrkPrincipal.Close

End Sub
```

Ejemplo de la propiedad CollatingOrder

Este ejemplo muestra la propiedad **CollatingOrder** de la base de datos Neptuno y de los campos individuales de una tabla.

```
Sub CollatingOrderX()

   Dim dbsNeptuno As Database
   Dim fldBucle As Field

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Muestra la secuencia de ordenación de la base de datos Neptuno.
      Debug.Print "Secuencia de ordenación de " & .Name & " = " & _
         .CollatingOrder

      ' Muestra la secuencia de ordenación de los campos del objeto
TableDef.
      Debug.Print " Secuencia de ordenación de los campos en " & _
         .TableDefs(0).Name & " table:"
      For Each fldBucle In .TableDefs(0).Fields
         Debug.Print "    " & fldBucle.Name & " = " & _
            fldBucle.CollatingOrder
      Next fldBucle

      .Close
   End With

End Sub
```

Ejemplo de la propiedad ConflictTable

Este ejemplo imprime los nombres de las tablas que tienen conflictos.

```
Sub ConflictTableX()

   Dim dbsNeptuno As Database
   Dim tdfPrueba As TableDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Enumera la colección TableDefs y comprueba la propiedad ConflictTable
   ' de cada elemento.
   For Each tdfPrueba In dbsNeptuno.TableDefs
      If tdfPrueba.ConflictTable <> "" Then _
         Debug.Print tdfPrueba.Name & " tiene un conflicto."
   Next tdfPrueba

   dbsNeptuno.Close

End Sub
```

Este ejemplo abre un objeto **Recordset** de la tabla en conflicto y otro uno de la tabla que originó el conflicto. Después procesa los registros de estas tablas utilizando el campo FechaPedido para copiar información de una tabla a la otra dependiendo de qué registro se actualizó más recientemente.

```
Sub ConflictTableX2(dbsResolver As Database)

   Dim tdfPrueba As TableDef
   Dim rstOrigen As Recordset
   Dim rstConflicto As Recordset
   Dim fldBucle As Field

   Set tdfPrueba = dbsResolver.TableDefs("Pedidos")

   If tdfPrueba.ConflictTable <> "" Then

      Set rstOrigen = dbsResolver.OpenRecordset(_
         tdfPrueba.Name, dbOpenTable)
      Set rstConflicto = dbsResolver.OpenRecordset(_
         tdfPrueba.ConflictTable, dbOpenTable)
      rstOrigen.Index = "[Id_Guid]"
      rstConflicto.MoveFirst

      Do Until rstConflicto.EOF
         rstOrigen.Seek "=", rstConflicto![s_Guid]
         If Not rstOrigen.NoMatch Then
            If rstOrigen!FechaPedido <
                  rstConflicto!FechaPedido Then
               On Error Resume Next
                  For Each fldBucle in rstConflicto.Fields
                     fldBucle = rstOrigen(fldBucle.Name)
                  Next fldBucle
               On Error Goto 0
            End If
         End If
         rstConflicto.Delete
```

```
         rstConflicto.MoveNext
      Loop

      rstConflicto.Close
      rstOrigen.Close
   End If

End Sub
```

Ejemplo de las propiedades Connect y SourceTableName (Ejemplos de las propiedades)

Este ejemplo utiliza las propiedades **Connect** y **SourceTableName** para vincular varias tablas externas a una base de datos Microsoft Jet. Se necesita el procedimiento SalidaConexión para ejecutar este procedimiento.

```
Sub ConnectX()

   Dim dbsTemp As Database
   Dim strMenú As String
   Dim strEntrada As String

   ' Abre una base de datos Microsoft Jet a la que vinculará
   ' una tabla.
   Set dbsTemp = OpenDatabase("BD1.mdb")

   ' Crea el texto del menú.
   strMenú = "Escriba el número del origen de datos :" & vbCr
   strMenú = strMenú & _
      "   1. Base de datos Microsoft Jet" & vbCr
   strMenú = strMenú & _
      "   2. Tabla Microsoft FoxPro 3.0" & vbCr
   strMenú = strMenú & _
      "   3. Tabla dBASE" & vbCr
   strMenú = strMenú & _
      "   4. Tabla Paradox" & vbCr
   strMenú = strMenú & _
      "   M. (ver opciones 5-9)"

   ' Obtiene la elección del usuario.
   strEntrada = InputBox(strMenú)

   If UCase(strEntrada) = "M" Then

      ' Crea el texto del menú.
      strMenú = "Escriba el número del origen de datos :" & vbCr
      strMenú = strMenú & _
         "   5. Hoja de cálculo Microsoft Excel" & vbCr
      strMenú = strMenú & _
         "   6. Hoja de cálculo Lotus" & vbCr
      strMenú = strMenú & _
         "   7. Texto delimitado por comas (CSV)" & vbCr
      strMenú = strMenú & _
         "   8. Tabla HTML" & vbCr
      strMenú = strMenú & _
         "   9. Carpeta de Microsoft Exchange"

      ' Obtiene la elección del usuario.
      strEntrada = InputBox(strMenú)

   End If

   ' Llama al procedimiento SalidaConexión. El tercer argumento
   ' se utilizará como la cadena Connect y el cuarto argumento
   ' se utilizará como el SourceTableName.
   Select Case Val(strEntrada)
      Case 1
```

```
         SalidaConexión dbsTemp, _
            "JetTable", _
            ";DATABASE=C:\Mis Documentos\Neptuno.mdb", _
            "Empleados"
      Case 2
         SalidaConexión dbsTemp, _
            "FoxProTable", _
            "FoxPro 3.0;DATABASE=C:\FoxPro30\Ejemplos", _
            "VentasT1"
      Case 3
         SalidaConexión dbsTemp, _
            "dBASETable", _
            "dBase IV;DATABASE=C:\dBASE\Ejemplos", _
            "Cuentas"
      Case 4
         SalidaConexión dbsTemp, _
            "ParadoxTable", _
            "Paradox 3.X;DATABASE=C:\Paradox\Ejemplos", _
            "Cuentas"
      Case 5
         SalidaConexión dbsTemp, _
            "ExcelTable", _
            "Excel 5.0;" & _
               "DATABASE=C:\Excel\Ejemplos\VentasT1.xls", _
            "Ventas Enero"
      Case 6
         SalidaConexión dbsTemp, _
            "LotusTable", _
            "Lotus WK3;" & _
               "DATABASE=C:\Lotus\Ejemplos\Ventas.xls", _
            "TERCERT"
      Case 7
         SalidaConexión dbsTemp, _
            "CSVTable", _
            "Text;DATABASE=C:\Ejemplos", _
            "Ejemplo.txt"
      Case 8
         SalidaConexión dbsTemp, _
            "HTMLTable", _
            "HTML Import;DATABASE=http://" & _
               "www.servidor1.com/ejemplos/página1.html", _
            "DatosVentasT1"
      Case 9
         SalidaConexión dbsTemp, _
            "ExchangeTable", _
            "Exchange 4.0;MAPILEVEL=" & _
               "Mailbox - Miguel López (Exchange)" & _
               "|Gente\Importante;", _
            "Pablo Alonso"
   End Select

   dbsTemp.Close

End Sub

Sub SalidaConexión(dbsTemp As Database, _
   strTabla As String, strConectar As String, _
```

```
    strTablaOrigen As String)

    Dim tdfVinculado As TableDef
    Dim As Recordset
    Dim intTemp As Integer

    ' Crea un nuevo TableDef, establece las propiedades Connect
    ' y SourceTableName basadas en los argumentos transferidos
    ' y lo anexa a la colección TableDefs.
    Set tdfVinculado = dbsTemp.CreateTableDef(strTabla)

    tdfVinculado.Connect = strConectar
    tdfVinculado.SourceTableName = strTablaOrigen
    dbsTemp.TableDefs.Append tdfVinculado

    Set rstVinculado = dbsTemp.OpenRecordset(strTabla)

    Debug.Print "Datos de la tabla vinculada :"

    ' Muestra los primeros tres registros de la tabla vinculada.
    intTemp = 1
    With rstVinculado
       Do While Not .EOF And intTemp <= 3
          Debug.Print , .Fields(0), .Fields(1)
          intTemp = intTemp + 1
          .MoveNext
       Loop
       If Not .EOF Then Debug.Print , "[registros adicionales]"
       .Close
    End With

    ' Elimina la tabla vinculada porque esto es un ejemplo.
    dbsTemp.TableDefs.Delete strTabla

End Sub
```

Ejemplo de la propiedad Container

Este ejemplo muestra la propiedad **Container** para diversos objetos **Document**.

```
Sub ContainerPropertyX()

   Dim dbsNeptuno As Database
   Dim ctrBucle As Container

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   ' Muestra el nombre del contenedor para el primer objeto Document
   ' en cada objeto Container de la colección Documents.
   For Each ctrBucle In dbsNeptuno.Containers
      Debug.Print "Document: " & ctrBucle.Documents(0).Name
      Debug.Print "    Container = " & _
         ctrBucle.Documents(0).Container
   Next ctrBucle

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Database

Este ejemplo utiliza **Database** para mostrar cómo el código utilizado para tener acceso datos ODBC mediante el motor de base de datos Microsoft Jet se puede convertir para utilizar objetos **Connection** de ODBCDirect.

El procedimiento AntiguoCódigoBasedeDatos utiliza un origen de datos conectado a Microsoft Jet para tener acceso a bases de datos ODBC.

```
Sub AntiguoCódigoBasedeDatos()

   Dim wrkPrincipal As Workspace
   Dim dbsEditores As Database
   Dim prpBucle As Property

   ' Crea un objeto Workspace Microsoft Jet.
   Set wrkPrincipal = CreateWorkspace("", "admin", "", dbUseJet)

   ' Abre un objeto Database basado en información de la
   ' cadena de conexión.
   Set dbsEditores = wrkPrincipal.OpenDatabase("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Enumera la colección Properties del objeto
   ' Database.
   With dbsEditores
      Debug.Print "Propiedades Database de " & _
         .Name & ":"

      On Error Resume Next
      For Each prpBucle In .Properties
         If prpBucle.Name = "Connection" Then
            ' La propiedad realmente devuelve un objeto Connection.
            Debug.Print "    Connection[.Name] = " & _
               .Connection.Name
         Else
            Debug.Print "    " & prpBucle.Name & " = " & _
               prpBucle
         End If
      Next prpBucle
      On Error GoTo 0

   End With

   dbsEditores.Close
   wrkPrincipal.Close

End Sub
```

El ejemplo NuevoCódigoBasedeDatos abre un objeto **Connection** en un espacio de trabajo ODBCDirect. Después asigna la propiedad **Database** del objeto **Connection** a una variable de objeto con el mismo nombre que le origen de datos en el antiguo procedimiento. No se ha cambiado nada del código subsiguiente mientras que no se utilicen unas características específicas para espacios de trabajo Microsoft Jet.

```
Sub NuevoCódigoBasedeDatos()
```

```
    Dim wrkPrincipal As Workspace
    Dim conEditores As Connection
    Dim dbsEditores As Database
    Dim prpBucle As Property

    ' Crea un objeto Workspace ODBCDirect en vez de un objeto
    ' Workspace Microsoft Jet.
    Set wrkPrincipal = CreateWorkspace("", "admin", "", dbUseODBC)

    ' Abre el objeto Connection basado en información de
    ' la cadena de conexión.
    Set conEditores = wrkPrincipal.OpenConnection("Editores", _
       dbDriverNoPrompt, False, _
       "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")
    ' Asigna la propiedad Database a la misma variable de
    ' objeto que en el código antiguo.
    Set dbsEditores = conEditores.Database

    ' Enumera la colección Properties del objeto Database.
    ' Desde este punto, el código es el mismo que el ejemplo
    ' antiguo.
    With dbsEditores
       Debug.Print "Propiedades Database de " & _
          .Name & ":"

       On Error Resume Next
       For Each prpBucle In .Properties
          If prpBucle.Name = "Connection" Then
             ' La propiedad realmente devuelve un objeto Connection.
             Debug.Print "    Connection[.Name] = " & _
                .Connection.Name
          Else
             Debug.Print "    " & prpBucle.Name & " = " & _
                prpBucle
          End If
       Next prpBucle
       On Error GoTo 0

    End With

    dbsEditores.Close
    wrkPrincipal.Close

End Sub
```

Ejemplo de la propiedad DefaultType

Este ejemplo utiliza la propiedad **DefaultType** para predeterminar qué tipo de objeto **Workspace** se creará cuando llame al método **CreateWorkspace**. Se necesita la función TipoSalida para ejecutar este procedimiento.

```
Sub DefaultTypeX()

   Dim wrkODBC As Workspace
   Dim wrkJet As Workspace
   Dim prpBucle As Property

   ' Establece la propiedad DefaultType y crea el objeto Workspace
   ' sin especificar un tipo.
   DBEngine.DefaultType = dbUseODBC
   Set wrkODBC = CreateWorkspace("WorkspaceODBC", _
      "admin", "")

   Debug.Print "DBEngine.DefaultType = " & _
      TipoSalida(DBEngine.DefaultType)
   With wrkODBC
      ' Enumera la colección Properties del objeto Workspace.
      Debug.Print "Propiedades de " & .Name
      On Error Resume Next
      For Each prpBucle In .Properties
         Debug.Print "    " & prpBucle.Name & " = " & prpBucle
         If prpBucle.Name = "Tipo" Then Debug.Print _
            "        (" & TipoSalida(prpBucle.Value) & ")"
      Next prpBucle
      On Error GoTo 0
   End With

   ' Establece la propiedad DefaultType y crea el objeto Workspace
   ' sin especificar un tipo.
   DBEngine.DefaultType = dbUseJet
   Set wrkJet = CreateWorkspace("WorkspaceJet", "admin", "")

   Debug.Print "DBEngine.DefaultType = " & _
      TipoSalida(DBEngine.DefaultType)
   With wrkJet
      ' Enumera la colección Properties del objeto Workspace.
      Debug.Print "Propiedades de " & .Name
      On Error Resume Next
      For Each prpBucle In .Properties
         Debug.Print "    " & prpBucle.Name & " = " & prpBucle
         If prpBucle.Name = "Tipo" Then Debug.Print _
            "        (" & TipoSalida(prpBucle.Value) & ")"
      Next prpBucle
      On Error GoTo 0
   End With

   wrkODBC.Close
   wrkJet.Close

End Sub

Function TipoSalida(intTemp As Integer) As String
```

```
   If intTemp = dbUseJet Then
      TipoSalida = "dbUseJet"
   Else
      TipoSalida = "dbUseODBC"
   End If

End Function
```

Ejemplo de las propiedades DefaultUser y DefaultPassword

Este ejemplo establece las propiedades **DefaultUser** y **DefaultPassword** que determinarán los valores de los objetos **Workspace** predeterminados.

```
Sub DefaultUserX()

   ' Define las propiedades DefaultUser y DefaultPassword para el
   ' objeto DBEngine.
   DBEngine.DefaultUser = "NuevoUsuario"
   DBEngine.DefaultPassword = ""

   Debug.Print _
      "Valor DBEngine.DefaultUser para 'NuevoUsuario'..."
   Debug.Print _
      "Valor DBEngine.DefaultPassword para & _
         "[cadena de longitud cero]..."

   Dim wrkJet As Workspace
   Dim wrkBucle As Workspace
   Dim prpBucle As Property

   Set wrkJet = CreateWorkspace("nuevo", "", _
      "", dbUseJet)

   ' Enumera la colección Workspaces.
   On Error Resume Next
   For Each wrkBucle In Workspaces
      Debug.Print "Workspace: " & wrkBucle.Name
      ' Enumera la colección Properties para cada
      ' objeto Workspace.
      For Each prpBucle In wrkBucle.Properties
         Debug.Print "    " & prpBucle.Name & " = " & prpBucle
      Next prpBucle
   Next wrkBucle
   On Error GoTo 0

   wrkJet.Close

End Sub
```

Ejemplo de la propiedad DesignMasterID

Este ejemplo establece la propiedad **DesignMasterID** para el valor de la propiedad **ReplicaID** de otra base de datos, haciendo esa base de datos el Diseño principal en el conjunto de réplica. Los Diseños principales nuevo y antiguo se sincronizan para actualizar el cambio de diseño. Para que funcione este código, debe crear un Diseño principal y una réplica, incluir sus nombres y rutas de acceso adecuadas y ejecutar este código desde una base de datos distinta del Diseño principal nuevo o antiguo.

```
Sub EstablecerNuevoModeloDiseño(strAntiguoDM as String, _
   strNuevoDM as String)

   Dim dbsAntigua As Database
   Dim dbsNueva As Database

   ' Abre el Diseño principal activo en modo exclusivo.
   Set dbsAntigua = OpenDatabase(strAntiguoDM, True)

   ' Abre la base de datos que será el nuevo
   ' Diseño principal.
   Set dbsNueva = OpenDatabase(strNuevoDM)

   ' Hace la nueva base de datos el Diseño principal.
   dbsAntigua.DesignMasterID = dbsNueva.ReplicaID

   ' Sincroniza el Diseño principal antiguo con el Diseño
   ' principal nuevo y permite intercambio bidireccional.
   dbsAntigua.Synchronize strNuevoDM, dbRepImpExpChanges
   dbsAntigua.Close
   dbsNueva.Close

End Sub
```

Ejemplo de la propiedad Direction

Este ejemplo utiliza la propiedad **Direction** para configurar los parámetros de una consulta a un origen de datos ODBC.

```
Sub DirectionX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset
   Dim strSQL As String
   Dim intBucle As Integer

   ' Crea un espacio de trabajo ODBC y abre una conexión a una
   ' base de datos Microsoft SQL Server.
   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Establece la cadena SQL para llamar al procedimiento almacenado
   ' Obtempportrabajo.
   strSQL = "{ call Obtempportrabajo (?, ?) }"

   Set qdfTemp = conPrincipal.CreateQueryDef("", strSQL)

   With qdfTemp
      ' Indica que dos consulta de parámetros sólo
      ' transferirán información a los procedimiento almacenados.
      .Parameters(0).Direction = dbParamInput
      .Parameters(1).Direction = dbParamInput

      ' Asigna los valores del parámetro inicial.
      .Parameters(0) = "0877"
      .Parameters(1) = 0

      Set rstTemp = .OpenRecordset()

      With rstTemp
         ' Hace un bucle a través de todos los valores válidos para el
segundo
         ' parámetro. Para cada valor, vuelve a consultar el conjunto de
registros
         ' para obtener los resultados correctos y después imprime el
contenido
         ' del conjunto de registros.
         For intBucle = 1 To 14
            qdfTemp.Parameters(1) = intBucle
            .Requery
            Debug.Print "Editor = " & _
               qdfTemp.Parameters(0) & _
               ", trabajo = " & intBucle
            Do While Not .EOF
               Debug.Print , .Fields(0), .Fields(1)
               .MoveNext
```

```
            Loop
         Next intBucle
         .Close
      End With

   End With

   conPrincipal.Close
   wrkPrincipal.Close

End Sub
```

Ejemplo de las propiedades Description, Number, Source, HelpFile y HelpContext del objeto Error, colección Errors

Este ejemplo genera un error, lo intercepta y muestra las propiedades **Description**, **Number**, **Source**, **HelpContext** y **HelpFile** del objeto **Error** resultante.

```
Sub DescriptionX()

   Dim dbsPrueba As Database

   On Error GoTo ControlError

   ' Desencadena un error intencionadamente.
   Set dbsPrueba = OpenDatabase("NingunaBasededatos")

   Exit Sub

ControlError:
   Dim strError As String
   Dim errBucle As Error

   ' Enumera la colección Errors y muestra las propiedades de
   ' cada objeto Error.
   For Each errBucle In Errors
      With errBucle
         strError = _
            "Número de Error " & .Number & vbCr
         strError = strError & _
            "    " & .Description & vbCr
         strError = strError & _
            "    (Origen: " & .Source & ")" & vbCr
         strError = strError & _
            "Presione F1 para ver el tema " & .HelpContext & vbCr
         strError = strError & _
            "    en el archivo " & .HelpFile & "."
      End With
      MsgBox strError
   Next

   Resume Next

End Sub
```

Ejemplo de la propiedad Inherit

Este ejemplo establece la propiedad **Inherit** del contenedor **Tables** a True para que cualquier objeto **Document** creado posteriormente en el contenedor **Tables** tenga las mismas definiciones de seguridad que el contenedor **Tables**.

```
Sub InheritX()

   Dim dbsNeptuno As Database
   Dim conTables As Container

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set conTables = dbsNeptuno.Containers("Tables")

   ' Definiendo la propiedad Inherit del contenedor Tables
   ' a verdadero y estableciendo sus permisos, cualquier objeto
   ' Document nuevo de este contenedor heredará el mismo valor
   ' de permisos.
   conTables.Inherit = True
   conTables.Permissions = dbSecWriteSec

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Inherited

Este ejemplo utiliza la propiedad **Inherited** para determinar si un objeto **Property** definido por el usuario se creó para un objeto **Recordset** o para algún objeto base.

```
Sub InheritedX()

   Dim dbsNeptuno As Database
   Dim tdfPrueba As TableDef
   Dim rstPrueba As Recordset
   Dim prpNueva As Property
   Dim prpBucle As Property

   ' Crea una nueva propiedad para un objeto TableDef guardado, después
   ' abre un Recordset de ese objeto TableDef.
   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set tdfPrueba = dbsNeptuno.TableDefs(0)
   Set prpNueva = tdfPrueba.CreateProperty("NuevaPropiedad", _
      dbBoolean, True)
   tdfPrueba.Properties.Append prpNueva
   Set rstPrueba = tdfPrueba.OpenRecordset(dbOpenForwardOnly)

   ' Muestra la propiedad Name e Inherited del nuevo objeto
   ' Property en el objeto TableDef.
   Debug.Print "NuevaPropiedad de " & tdfPrueba.Name & _
      " TableDef:"
   Debug.Print "    Inherited = " & _
      tdfPrueba.Properties("NuevaPropiedad").Inherited

   ' Muestra la propiedad Name e Inherited del nuevo objeto
   ' Property del objeto Recordset.
   Debug.Print "NuevaPropiedad de " & rstPrueba.Name & _
      " Recordset:"
   Debug.Print "    Inherited = " & _
      rstPrueba.Properties("NuevaPropiedad").Inherited

   ' Elimina el nuevo objeto TableDef porque esto es un ejemplo.
   tdfPrueba.Properties.Delete prpNueva.Name
   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad IniPath

Este ejemplo establece la ruta de acceso en la propiedad **IniPath** para una clave de aplicación en el Registro de Windows.

```
Sub IniPathX()

   ' Cambia la propiedad IniPath para señalar una sección
   ' diferente del Registro de Windows para información de
   ' valores.
   Debug.Print "Valor de IniPath original = " & _
      IIf(DBEngine.IniPath = "", "[Vacío]", _
      DBEngine.IniPath)
   DBEngine.IniPath = _
      "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\" & _
      "Jet\3.5\ISAM Formats\FoxPro 3.0"
   Debug.Print " Nuevo valor de IniPath = " & _
      IIf(DBEngine.IniPath = "", "[Vacío]", _
      DBEngine.IniPath)

End Sub
```

Ejemplo de la propiedad IsolateODBCTrans

Este ejemplo abre tres espacios de trabajo ODBCDirect y establece sus propiedades **IsolateODBCTrans** a True para separar del mismo origen de datos transacciones múltiples.

```
Sub IsolateODBCTransX()

   DBEngine.DefaultType = dbUseJet

   Dim wrkJet1 As Workspace
   Dim wrkJet2 As Workspace
   Dim wrkJet3 As Workspace

   ' Abre tres espacios de trabajo ODBCDirect para separar
   ' transacciones involucradas en el mismo origen de datos ODBC.
   Set wrkJet1 = CreateWorkspace("", "admin", "")
   wrkJet1.IsolateODBCTrans = True

   Set wrkJet2 = CreateWorkspace("", "admin", "")
   wrkJet2.IsolateODBCTrans = True

   Set wrkJet3 = CreateWorkspace("", "admin", "")
   wrkJet3.IsolateODBCTrans = True

   wrkJet1.Close
   wrkJet2.Close
   wrkJet3.Close

End Sub
```

Ejemplo de la propiedad KeepLocal

El siguiente ejemplo anexa la propiedad **KeepLocal** a la colección de propiedades de un objeto documento del módulo Funciones de utilidad en la base de datos Neptuno. Establezca este propiedad en un objeto (como una tabla) antes de hacer replicable una base de datos. Cuando la base de datos se convierte en un Diseño principal, el objeto que especificó para mantenerse local no se copiará a otros miembros del conjunto de réplica. Ajuste la ruta de acceso de Neptuno.mdb a la adecuada en su equipo.

```
Sub KeepLocalNWObjectX()

   Dim dbsNeptuno As Database
   Dim docTemp As Document
   Dim prpTemp As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   Set docTemp = dbsNeptuno.Containers("Modules"). _
      Documents("Funciones de utilidad")
   Set prpTemp = docTemp.CreateProperty("KeepLocal", _
      dbText, "T")
   docTemp.properties.append prpTemp
   dbsNeptuno.Close

End Sub
```

El siguiente código establece la propiedad **KeepLocal** del objeto **TableDef** especificado a `"T"`. Si no existe la propiedad **KeepLocal**, se crea y se anexa la colección **Properties** de la tabla y se le proporciona el valor `"T"`.

```
Sub EstablecerMantenerLocal(tdfTemp As TableDef)

   On Error GoTo ControlError

   tdfTemp.Properties("KeepLocal") = "T"

   On Error GoTo 0

   Exit Sub

ControlError:

   Dim prpNuevo As Property

   If Err.Number = 3270 Then
      Set prpNuevo = tdfTemp.CreateProperty("KeepLocal", _
         tdfTemp, dbText, "T")
      tdfTemp.Properties.Append prpNuevo
   Else
      MsgBox "Error " & Err & ": " & Error
   End If

End Sub
```

Ejemplo de la propiedad LoginTimeout

Este ejemplo establece la propiedad **LoginTimeout** del objeto **DBEngine** a 120 segundos. Después abre tres espacios de trabajo ODBCDirect y modifica de sus propiedades **LoginTimeout** el valor predeterminado heredado del objeto **DBEngine**.

```
Sub LoginTimeoutX()

   ' Cambia el valor de la propiedad LoginTimeout predeterminado.
   DBEngine.LoginTimeout = 120

   Dim wrkODBC1 As Workspace
   Dim wrkODBC2 As Workspace
   Dim wrkODBC3 As Workspace

   Set wrkODBC1 = CreateWorkspace("", "admin", "", _
      dbUseODBC)
   Set wrkODBC2 = CreateWorkspace("", "admin", "", _
      dbUseODBC)
   Set wrkODBC3 = CreateWorkspace("", "admin", "", _
      dbUseODBC)

   ' Cambia la propiedad LoginTimeout de los espacios de trabajo
   ' ODBCDirect individuales a 60 segundos, el tiempo predeterminado
   ' (120 segundos) y sin tiempo de espera.
   wrkODBC1.LoginTimeout = 60
   wrkODBC2.LoginTimeout = -1
   wrkODBC2.LoginTimeout = 0

   wrkODBC1.Close
   wrkODBC2.Close
   wrkODBC3.Close

End Sub
```

Ejemplo de las propiedades LogMessages y ReturnsRecords

Este ejemplo utiliza las propiedades **LogMessages** y **ReturnsRecords** para crear una consulta de paso a través que devolverá datos y cualquier mensaje generado por el servidor remoto.

```
Sub LogMessagesX()

   Dim wrkJet As Workspace
   Dim dbsActual As Database
   Dim qdfTemp As QueryDef
   Dim prpNueva As Property
   Dim rstTemp As Recordset

   ' Crea el objeto Workspace Microsoft Jet.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)

   Set dbsActual = wrkJet.OpenDatabase("DB1.mdb")

   ' Crea un objeto QueryDef que registrará cualquier mensaje del
   ' servidor en tablas temporales.
   Set qdfTemp = dbsActual.CreateQueryDef("NuevaDefConsulta")
   qdfTemp.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores"
   qdfTemp.SQL = "SELECT * FROM almacenes"
   qdfTemp.ReturnsRecords = True
   Set prpNueva = qdfTemp.CreateProperty("LogMessages", _
      dbBoolean, True)
   qdfTemp.Properties.Append prpNueva

   ' Ejecuta la consulta y muestra los resultados.
   Set rstTemp = qdfTemp.OpenRecordset()

   Debug.Print "Contenido del Recordset:"
   With rstTemp
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
      .Close
   End With

   ' Elimina el nuevo objeto QueryDef porque esto es un ejemplo.
   dbsActual.QueryDefs.Delete qdfTemp.Name
   dbsActual.Close
   wrkJet.Close

End Sub
```

Ejemplo de la propiedad MaxRecords

Este ejemplo utiliza la propiedad **MaxRecords** para establecer el límite de cuántos registros se devuelven en una consulta en un origen de datos ODBC.

```
Sub MaxRecordsX()

   Dim dbsActual As Database
   Dim qdfPasoaTravés As QueryDef
   Dim qdfLocal As QueryDef
   Dim rstTemp As Recordset

   ' Abre una base de datos de la que se pueden crear los objetos
   ' QueryDef.
   Set dbsActual = OpenDatabase("DB1.mdb")

   ' Crea una consulta de paso a través para recuperar datos de
   ' una base de datos SQL Server.
   Set qdfPasoaTravés = _
      dbsActual.CreateQueryDef("")

   ' Establece las propiedades de la nueva consulta, limitando
   ' a 20 el número de registros devueltos.
   qdfPasoaTravés.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores"
   qdfPasoaTravés.SQL = "SELECT * FROM títulos"
   qdfPasoaTravés.ReturnsRecords = True
   qdfPasoaTravés.MaxRecords = 20

   Set rstTemp = qdfPasoaTravés.OpenRecordset()

   ' Muestra los resultados de la consulta.
   Debug.Print "Resultados de la consulta:"
   With rstTemp
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
      .Close
   End With

   dbsActual.Close

End Sub
```

Ejemplo de la propiedad Name

Este ejemplo utiliza la propiedad **Name** para dar un nombre a un objeto creado nuevo, para mostrar qué objetos están en una colección determinada y para eliminar un objeto de una colección.

```
Sub NameX()

   Dim dbsNeptuno As Database
   Dim qdfNuevo As QueryDef
   Dim qdfBucle As QueryDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Crea un nuevo objeto QueryDef permanente y lo anexa
      ' a la colección QueryDefs.
      Set qdfNuevo = .CreateQueryDef()
      qdfNuevo.Name = "NuevaDefConsulta"
      qdfNuevo.SQL = "SELECT * FROM Empleados"
      .QueryDefs.Append qdfNuevo

      ' Enumera la colección QueryDefs para mostrar los
      ' nombres de los objetos QueryDef.
      Debug.Print "Nombres de las consultas de " & .Name

      For Each qdfBucle In .QueryDefs
         Debug.Print "    " & qdfBucle.Name
      Next qdfBucle

      ' Elimina el nuevo objeto QueryDef porque esto es un
      ' ejemplo.
      .QueryDefs.Delete qdfNuevo.Name

      .Close
   End With

End Sub
```

Ejemplo de las propiedades ODBCTimeout y QueryTimeout

Este ejemplo utiliza las propiedades **ODBCTimeout** y **QueryTimeout** para mostrar cómo el valor **QueryTimeout** de un objeto **Database** establece el valor predeterminado **ODBCTimeout** de cualquier objeto **QueryDef** creado de un objeto **Database**.

```
Sub ODBCTimeoutX()

   Dim dbsActual As Database
   Dim qdfAlmacenes As QueryDef
   Dim rstAlmacenes As Recordset

   Set dbsActual = OpenDatabase("Neptuno.mdb")

   ' Cambia la propiedad QueryTimeout predeterminada de la base de datos
   ' Neptuno.
   Debug.Print "QueryTimeout predeterminado de Database: " & _
      dbsActual.QueryTimeout
   dbsActual.QueryTimeout = 30
   Debug.Print "QueryTimeout nuevo de Database: " & _
      dbsActual.QueryTimeout

   ' Crea un nuevo objeto QueryDef.
   Set qdfAlmacenes = dbsActual.CreateQueryDef("Almacenes", _
      "SELECT * FROM almacenes")
   qdfAlmacenes.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores"

   ' Cambia el valor de ODBCTimeout del nuevo objeto QueryDef
   ' de su valor predeterminado.
   Debug.Print "ODBCTimeout predeterminado de QueryDef: " & _
      qdfAlmacenes.ODBCTimeout
   qdfAlmacenes.ODBCTimeout = 0
   Debug.Print "ODBCTimeout nuevo de QueryDef: " & _
      qdfAlmacenes.ODBCTimeout

   ' Ejecuta la consulta y muestra los resultados.
   Set rstAlmacenes = qdfAlmacenes.OpenRecordset()

   Debug.Print "Contenido del Recordset:"
   With rstAlmacenes
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
      .Close
   End With

   ' Elimina el nuevo QueryDef porque esto es un ejemplo.
   dbsActual.QueryDefs.Delete qdfAlmacenes.Name
   dbsActual.Close

End Sub
```

Ejemplo de las propiedades Owner y SystemDB

Este ejemplo utiliza las propiedades **Owner** y **SystemDB** para mostrar los propietarios de diversos objetos **Document**.

```
Sub OwnerX()

   ' Comprueba que está disponible el archivo de grupo
   ' de trabajo Microsoft Jet.
   DBEngine.SystemDB = "system.mdw"

   Dim dbsNeptuno As Database
   Dim ctrBucle As Container

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      Debug.Print "Propietarios de Document:"
      ' Enumera la colección Containers y muestra el propietario
      ' del primer Document en cada Documents del
      ' contenedor.
      For Each ctrBucle In .Containers
         With ctrBucle
            Debug.Print "    [" & .Documents(0).Name & _
               "] en el contenedor [" & .Name & _
               "] propiedad de [" & _
               .Documents(0).Owner & "]"
         End With
      Next ctrBucle

      .Close
   End With

End Sub
```

Ejemplo de la propiedad PartialReplica

El siguiente ejemplo utiliza el propiedad **PartialReplica** para replicar todos los registros que representan a los pedidos de los clientes cuya región es Lara:

```
Sub PartialReplicaX()

' Supuestos: La base de datos abierta actual, dbs, es la réplica
' parcial y ya existen las relaciones adecuadas entre
' las tablas.

   Dim tdfPedidos As TableDef
   Dim relPedClie As Relation
   Dim dbsTemp As Database
   Dim relLoop As Relation

   Set dbsTemp = OpenDatabase("Neptuno.mdb")
   Set tdfPedidos = dbsTemp.TableDefs("Pedidos")

   ' Encuentra el objeto Relation "Pedidos de Clientes".
   For Each relLoop In dbsTemp.Relations
      If relLoop.Table = "Clientes" And _
            relLoop.ForeignTable = "Pedidos" Then
         ' Establece la propiedad PartialReplica del objeto Relation
         ' a True.
         relLoop.PartialReplica = True
         Exit For
      End If
   Next relLoop

End Sub
```

Nota Si establece un filtro de réplica y una relación de réplica en la misma tabla, ambos actúan en combinación como una operación lógica OR, no como una operación lógica AND. Por ejemplo, en el ejemplo anterior, los registros intercambiados durante la sincronización son todos pedidos mayores que 1000 dólares OR todos los pedidos cuya región es Lara, no todos los pedidos cuya región es Lara que son mayores de 1000 dólares.

Ejemplo de la propiedad Prepare

Este ejemplo utiliza la propiedad **Prepare** para especificar que se debe ejecutar directamente una consulta en vez de crear antes un procedimiento almacenado temporal en el servidor.

```
Sub PrepareX()

   Dim wrkODBC As Workspace
   Dim conEdits As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset

   ' Crea un objeto Workspace ODBCDirect y abre un objeto Connection.
   Set wrkODBC = CreateWorkspace("", _
      "admin", "", dbUseODBC)
   Set conEdits = wrkODBC.OpenConnection("Editores", , , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   Set qdfTemp = conEdits.CreateQueryDef("")

   With qdfTemp
      ' Ya que sólo ejecutará esta consulta una vez, especifica
      ' la función API de ODBC SQLExecDirect. Si no establece
      ' esta propiedad antes de establecer la propiedad SQL,
      ' la función API de ODBC SQLPrepare será llamada de cualquier
      ' modo lo que anulará cualquier ventaja de
      ' rendimiento.
      .Prepare = dbQUnprepare
      .SQL = "UPDATE tipoimpuesto " & _
         "SET impuesto = impuesto * 2 " & _
         "WHERE id_título LIKE 'BU____' OR " & _
         "id_título LIKE 'PC____'"
      .Execute
   End With

   Debug.Print "Resultados de la consulta:"

   ' Abre el conjunto de registros que contiene los registros modificados.
   Set rstTemp = conEdits.OpenRecordset( _
      "SELECT * FROM tipoimpuesto " & _
      "WHERE id_título LIKE 'BU____' OR " & _
      "id_título LIKE 'PC____'")

   ' Enumera el conjunto de registros.
   With rstTemp
      Do While Not .EOF
         Debug.Print , !id_título, !intmenor, _
            !intmayor, !impuesto
         .MoveNext
      Loop
      .Close
   End With

   conEdits.Close
   wrkODBC.Close

End Sub
```

Ejemplo de las propiedades RecordStatus y DefaultCursorDriver

Este ejemplo utiliza las propiedades **RecordStatus** y **DefaultCursorDriver** para mostrar cómo los cambios en un **Recordset** local se siguen durante una actualización por lotes. Se necesita la función SalidaRecordStatus para ejecutar este procedimiento.

```
Sub RecordStatusX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim rstTemp As Recordset

   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' Se necesita este valor de DefaultCursorDriver para
   ' actualización por lotes.
   wrkPrincipal.DefaultCursorDriver = dbUseClientBatchCursor

   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")
   ' se necesita el siguiente argumento de bloqueo para
   ' la actualización por lotes.
   Set rstTemp = conPrincipal.OpenRecordset( _
      "SELECT * FROM autores", dbOpenDynaset, 0, _
      dbOptimisticBatch)

   With rstTemp
      .MoveFirst
      Debug.Print "Registro original: " & !noml_autor
      Debug.Print , SalidaRecordStatus(.RecordStatus)

      .Edit
      !noml_autor = "García"
      .Update
      Debug.Print "Registro modificado: " & !noml_autor
      Debug.Print , SalidaRecordStatus(.RecordStatus)

      .AddNew
      !noml_autor = "NuevoNombre"
      .Update
      Debug.Print "Registro nuevo: " & !noml_autor
      Debug.Print , SalidaRecordStatus(.RecordStatus)

      .Delete
      Debug.Print "Registro eliminado: " & !noml_autor
      Debug.Print , SalidaRecordStatus(.RecordStatus)

      ' Cierra el conjunto de registros local si actualizar los
      ' datos en el servidor.
      .Close
   End With

   conPrincipal.Close
   wrkPrincipal.Close

End Sub
```

```
Function SalidaRecordStatus(lngTemp As Long) As String

   Dim strTemp As String

   strTemp = ""

   ' Genera una cadena de salida basada en el valor de
   ' RecordStatus.
If lngTemp = dbRecordUnmodified Then _
      strTemp = "[dbRecordUnmodified]"
   If lngTemp = dbRecordModified Then _
      strTemp = "[dbRecordModified]"
   If lngTemp = dbRecordNew Then _
      strTemp = "[dbRecordNew]"
   If lngTemp = dbRecordDeleted Then _
      strTemp = "[dbRecordDeleted]"
   If lngTemp = dbRecordDBDeleted Then _
      strTemp = "[dbRecordDBDeleted]"

   SalidaRecordStatus = strTemp

End Function
```

Ejemplo de la propiedad Replicable

Estos ejemplos presentan dos situaciones para utilizar la propiedad **Replicable** - en una base de datos y en un objeto de la base de datos.

Este ejemplo Neptuno.mdb una base de datos replicable. Se recomienda que haga una copia de seguridad de Neptuno antes de ejecutar este código y que ajuste la ruta de acceso de Neptuno a la adecuada en su equipo.

```
Sub CreateDesignMasterX()

   Dim dbsNeptuno As Database
   Dim prpNueva As Property

   ' Abre la base de datos parámetro acceso en modo exclusivo.
   Set dbsNeptuno = OpenDatabase("Neptuno.mdb", _
      True)

   With dbsNeptuno

      ' Si no existe la propiedad Replicable, la crea.
      ' Desactiva el control de errores en el caso de que exista.
      On Error Resume Next
      Set prpNueva =.CreateProperty("Replicable", _
         dbText, "T")
      .Properties.Append prpNueva

      ' Establece la propiedad Replicable de la base de datos a True.
      .Properties("Replicable ") = "T"

      .Close

   End With

End Sub
```

Este ejemplo crea un nuevo objeto **TableDef** y después lo hace replicable. Para que este procedimiento funcione la base de datos deber ser replicable.

```
Sub CreateLocalTabelReplicableX()

   Dim dbsNeptuno As Database
   Dim tdfNuevo As TableDef
   Dim fldNuevo As Field
   Dim prpNueva As Property

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   ' Crea un nuevo objeto TableDef llamado "Impuestos".
   Set tdfNuevo = dbsNeptuno.CreateTableDef("Impuestos")
   ' Define un campo de texto llamado "Grado".
   Set fldNuevo = tdfNuevo.CreateField("Grado", dbText, 3)
   ' Anexa el nuevo campo al objeto TableDef.
   tdfNuevo.Fields.Append fldNuevo

   ' Agrega el nuevo objeto TableDef a la base de datos.
   dbsNeptuno.TableDefs.Append tdfNuevo

   ' Crea una nueva propiedad Replicable para el nuevo objeto TableDef.
```

```
   Set prpNueva = tdfNuevo.CreateProperty("Replicable", _
      dbText, "T")
   ' Anexa la propiedad Replicable al nuevo objeto
   ' TableDef.
   tdfNuevo.Properties.Append prpNueva
   dbsNeptuno.Close

End Sub
```

El siguiente código establece la propiedad **Replicable** en el objeto **TableDef** especificado a `"T"`. Si la propiedad no existe, se crea y se anexa a la colección **Properties** de la tabla y se le proporciona el valor `"T"`.

```
Sub EstablecerReplicableBool(tdfTemp As TableDef)

   On Error GoTo ControlError

   tdfTemp.Properties("Replicable") = "T"

   On Error GoTo 0

   Exit Sub

ControlError:

   Dim prpNuevo As Property

   If Err.Number = 3270 Then
      Set prpNuevo = tdfTemp.CreateProperty("Replicable", tdfTemp, dbText,
_
         "T")
      tdfTemp.Properties.Append prpNuevo
   Else
      MsgBox "Error " & Err & ": " & Error
   End If

End Sub
```

Ejemplo de la propiedad ReplicaFilter

El siguiente ejemplo utiliza la propiedad **ReplicaFilter** para replicar sólo los registros de clientes cuya región es Lara.

```
Sub ReplicaFilterX()

   ' Este ejemplo asume que la base de datos abierta activo
   ' es la réplica.
   Dim tdfClientes As TableDef
   Dim strFiltro As String
   Dim dbsTemp As Database

   Set dbsTemp = OpenDatabase("Neptuno.mdb")
   Set tdfClientes = dbsTemp.TableDefs("Clientes")

   ' Sincroniza con la réplica completa
   ' antes de establecer el filtro de réplica.
   dbsTemp.Synchronize "C:\VENTAS\AÑO96.MDB"

   strFiltro = "Region = 'Lara'"
   tdfClientes.ReplicaFilter = strFiltro
   dbsTemp.PopulatePartial "C:\VENTAS\AÑO96.MDB"

   ' Ahora quita el filtro de réplica (sólo para
   ' el ejemplo).
   tdfClientes.ReplicaFilter = False
   ' Rellena la base de datos.
   dbsTemp.PopulatePartial "C:\VENTAS\DATOS96.MDB"

End Sub
```

Ejemplo de la propiedad ReplicaID

Este ejemplo hace una réplica del Diseño principal de Neptuno.mdb y después devuelve la propiedad **ReplicaID** de la réplica, que crea automáticamente el motor de base de datos Microsoft Jet. (Si no tiene todavía creada un Diseño principal de Neptuno, haga referencia a la propiedad **Replicable** o cambie el nombre de la base de datos en el código para un Diseño principal existente.)

```
Sub CreateReplicaReplicaIDX()

   Dim dbsNeptuno As Database
   Dim prpIdRéplica As Property
   Dim dbsRéplica As Database

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   'Crea una nueva réplica
   dbsNeptuno.MakeReplica("NuevaRéplica2.mdb"), _
      "segunda réplica"
   dbsNeptuno.Close

   'Abre la nueva réplica para leer su propiedad ReplicaID.
   Set dbsRéplica = OpenDatabase("NuevaRéplica2.mdb")

   Debug.Print dbsRéplica. ReplicaID
   dbsRéplica.Close

End Sub
```

Ejemplo de la propiedad Required

Este ejemplo utiliza la propiedad **Required** para informar qué campos en tres tablas diferentes deben contener los datos en orden para agregar un registro nuevo. Se necesita el procedimiento RequiredOutput para ejecutar este procedimiento.

```
Sub RequiredX()

   Dim dbsNeptuno As Database
   Dim tdfBucle As TableDef

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   With dbsNeptuno
      ' Muestra qué campos se necesitan en la colección Fields
      ' de los tres objetos TableDef diferentes.
      RequiredOutput .TableDefs("Categorías")
      RequiredOutput .TableDefs("Clientes")
      RequiredOutput .TableDefs("Empleados")
      .Close
   End With

End Sub

Sub RequiredOutput(tdfTemp As TableDef)

   Dim fldBucle As Field

   ' Enumera la colección Fields del objeto TableDef especificado
   ' y muestra la propiedad Required.
   Debug.Print "Fields de " & tdfTemp.Name & ":"
   For Each fldBucle In tdfTemp.Fields
      Debug.Print , fldBucle.Name & ", Required = " & _
         fldBucle.Required
   Next fldBucle

End Sub
```

Ejemplo de la propiedad StillExecuting y el método Cancel

Este ejemplo utiliza la propiedad **StillExecuting** y el método **Cancel** para abrir un objeto **Connection** asíncronamente.

```
Sub CancelConnectionX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim sngHora As Single

   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' Abre la conexión asíncronamente.
   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt + dbRunAsync, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   sngHora = Timer

   ' Espera cinco segundos.
   Do While Timer - sngHora < 5
   Loop

   ' Si no se realiza la conexión, pregunta al usuario
   ' si desea seguir esperando. Si no, cancela la conexión
   ' y sale del procedimiento.
   Do While conPrincipal.StillExecuting

      If MsgBox("Todavía no hay conexión, ¿sigue esperando?", _
            vbYesNo) = vbNo Then
         conPrincipal.Cancel
         MsgBox "¡Conexión cancelada!"
         wrkPrincipal.Close
         Exit Sub
      End If

   Loop

   With conPrincipal
      ' Utiliza el objeto Connection conPrincipal.
      .Close
   End With

   wrkPrincipal.Close

End Sub
```

Este ejemplo utiliza la propiedad **StillExecuting** y el método **Cancel** para ejecutar asíncronamente un objeto **QueryDef**.

```
Sub CancelQueryDefX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim qdfTemp As QueryDef
   Dim sngHora As Single
```

```
   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   Set qdfTemp = conPrincipal.CreateQueryDef("")

   With qdfTemp
      .SQL = "UPDATE tipoimpuesto " & _
         "SET impuesto = impuesto * 2 " & _
         "WHERE id_título LIKE 'BU____' OR " & _
         "id_título LIKE 'PC____'"

      ' Ejecuta la consulta asíncronamente.
      .Execute dbRunAsync

      sngHora = Timer

      ' Espera cinco segundos.
      Do While Timer - sngHora < 5
      Loop

   ' Si no se completa la conexión, pregunta al usuario
   ' si desea seguir esperando. Si no, cancela la conexión
   ' y sale del procedimiento.
      Do While .StillExecuting

         If MsgBox( _
               "Todavía se está ejecutando la consulta, ¿sigue
esperando?", _
               vbYesNo) = vbNo Then
            .Cancel
            MsgBox "¡Consulta cancelada!"
            Exit Do
         End If

      Loop

   End With

   conPrincipal.Close
   wrkPrincipal.Close

End Sub
```

Este ejemplo utiliza la propiedad **StillExecuting** y el método **Cancel** para moverse asíncronamente al último registro de un objeto **Recordset**.

```
Sub CancelRecordsetX()

   Dim wrkPrincipal As Workspace
   Dim conPrincipal As Connection
   Dim rstTemp As Recordset
   Dim sngHora As Single

   Set wrkPrincipal = CreateWorkspace("ODBCWorkspace", _
```

```
      "admin", "", dbUseODBC)
   Set conPrincipal = wrkPrincipal.OpenConnection("Editores", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")
   Set rstTemp = conPrincipal.OpenRecordset( _
      "SELECT * FROM tipoimpuesto", dbOpenDynaset)

   With rstTemp

      ' Llama al método MoveLast asíncronamente.
      .MoveLast dbRunAsync

      sngHora = Timer

      ' Espera cinco segundos.
      Do While Timer - sngHora < 5
      Loop

   ' Si no se completa el método MoveLast, pregunta al usuario
   ' si desea seguir esperando. Si no, cancela la el método MoveLast
   ' y sale del procedimiento.
      Do While .StillExecuting

         If MsgBox( _
               "Todavía no es el último registro, ¿sigue esperando?", _
               vbYesNo) = vbNo Then
            .Cancel
            MsgBox "¡MoveLast cancelado!"
            conPrincipal.Close
            wrkPrincipal.Close
            Exit Sub
         End If

      Loop

      ' Utiliza el conjunto de registros.

      .Close

   End With

   conPrincipal.Close
   wrkPrincipal.Close

End Sub
```

Ejemplo de la propiedad Transactions

Este ejemplo demuestra la propiedad **Transactions** en espacios de trabajo Microsoft Jet y espacio de trabajo ODBCDirect.

```
Sub TransactionsX()

   Dim wrkJet As Workspace
   Dim wrkODBC As Workspace
   Dim dbsNeptuno As Database
   Dim conEditores As Connection
   Dim rstTemp As Recordset

   ' Abre Microsoft Jet y espacios de trabajo ODBCDirect, una base de
datos
   ' Microsoft Jet y una conexión ODBCDirect.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
   Set wrkODBC = CreateWorkspace("", "admin", "", dbUseODBC)
   Set dbsNeptuno = wrkJet.OpenDatabase("Neptuno.mdb")
   Set conEditores = wrkODBC.OpenConnection("", , , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Abre dos objetos Recordset diferentes y muestra la
   ' propiedad Transactions de cada uno.

   Debug.Print "Abriendo un Recordset Microsoft Jet de tipo Table "
   Set rstTemp = dbsNeptuno.OpenRecordset( _
      "Empleados", dbOpenTable)
   Debug.Print "    Transacciones = " & rstTemp.Transactions

   Debug.Print " Abriendo un Recordset de tipo Forward-only " & _
      "donde el origen es una instrucción SQL..."
   Set rstTemp = dbsNeptuno.OpenRecordset( _
      "SELECT * FROM Empleados", dbOpenForwardOnly)
   Debug.Print "    Transacciones = " & rstTemp.Transactions

   ' Muestra la propiedad Transactions de un objeto Connection en
   ' un espacio de trabajo ODBCDirect.
   Debug.Print "Probando la propiedad Transaction de " & _
      "una conexión ODBC..."
   Debug.Print "    Transacciones = " & conEditores.Transactions

   rstTemp.Close
   dbsNeptuno.Close
   conEditores.Close
   wrkJet.Close
   wrkODBC.Close

End Sub
```

Ejemplo de la propiedad UserName

Este ejemplo utiliza la propiedad **UserName** para cambiar los permisos de un usuario concreto en un objeto y para comprobar la capacidad del usuario para anexar nuevos datos al mismo objeto.

```
Sub UserNameX()

   ' Comprueba que el archivo de grupo de trabajo Microsoft Jet está
   ' disponible.
   DBEngine.SystemDB = "system.mdw"

   Dim dbsNeptuno As Database
   Dim docTemp As Document

   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")

   Set docTemp = _
      dbsNeptuno.Containers("Tables").Documents(0)

   ' Cambia los permisos de NuevoUsuario en el primer objeto
   ' Document del contenedor Tables.
   With docTemp
      .UserName = "NuevoUsuario"
      .Permissions = dbSecRetrieveData
      If (.Permissions And dbSecInsertData) = _
            dbSecInsertData Then
         Debug.Print .UserName & " puede insertar datos."
      Else
         Debug.Print .UserName & " no puede insertar datos."
      End If
   End With

   dbsNeptuno.Close

End Sub
```

Ejemplo de la propiedad Version

Este ejemplo utiliza la propiedad **Version** para informar del motor de base de datos Microsoft Jet, una base de datos Microsoft Jet y una conexión ODBC que están en memoria.

```
Sub VersionX()

   Dim wrkJet As Workspace
   Dim dbsNeptuno As Database
   Dim wrkODBC As Workspace
   Dim conEditores As Connection

   ' Abre un objeto Database Microsoft Jet.
   Set wrkJet = CreateWorkspace("WorkspaceJetNuevo", _
      "admin", "", dbUseJet)
   Set dbsNeptuno = wrkJet.OpenDatabase("Neptuno.mdb")

   ' Crea un objeto Workspace ODBCDirect y abre los objetos
   ' Connection.
   Set wrkODBC = CreateWorkspace("WorkspaceODBCNuevo", _
      "admin", "", dbUseODBC)
   Set conEditores = wrkODBC.OpenConnection("Conexión1", , , _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Editores")

   ' Muestra tres usos diferentes de la propiedad Version.
   Debug.Print "Versión de DBEngine (Microsoft Jet " & _
      "en memoria) = " & DBEngine.Version
   Debug.Print " Versión del motor de Microsoft Jet " & _
      "con el que se creó " & dbsNeptuno.Name & _
      " = " & dbsNeptuno.Version
   Debug.Print " Versión de la conexión ODBCDirect " & _
      "(utilizando la propiedad Database) = " & _
      conEditores.Database.Version

   dbsNeptuno.Close
   conEditores.Close
   wrkJet.Close
   wrkODBC.Close

End Sub
```

Ejemplo de las propiedades Connect y ReturnsRecords (Cliente/Servidor)

Este ejemplo utiliza las propiedades **Connect** y **ReturnsRecords** para seleccionar los cinco primeros títulos de una base de datos de Microsoft SQL Server basándose en las ventas del presente año. En caso de igualdad de ventas, el ejemplo aumenta el tamaño de la lista mostrando los resultados de la consulta e imprime un mensaje explicando lo ocurrido.

```
Sub ClienteServidorX1()

   Dim dbsActual As Database
   Dim qdfPasoaTravés As QueryDef
   Dim qdfLocal As QueryDef
   Dim rstCincoMásVendidos As Recordset
   Dim strMensaje As String

   ' Abre una base de datos de la que se pueden crear los
   ' objetos QueryDef.
   Set dbsActual = OpenDatabase("DB1.mdb")

   ' Crea una consulta de paso a través para recuperar datos de
   ' una base de datos Microsoft SQL Server.
   Set qdfPasoaTravés = _
      dbsActual.CreateQueryDef("TodoslosTítulos")
   qdfPasoaTravés.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Publicaciones"
   qdfPasoaTravés.SQL = "SELECT * FROM títulos " & _
      "ORDER BY Ventas_año DESC"
   qdfPasoaTravés.ReturnsRecords = True

   ' Crea un objeto QueryDef temporal para recuperar
   ' los datos de la consulta de paso a través.
   Set qdfLocal = dbsActual.CreateQueryDef("")
   qdfLocal.SQL = "SELECT TOP 5 Título FROM TodoslosTítulos"

   Set rstCincoMásVendidos = qdfLocal.OpenRecordset()

   ' Muestra los resultados de las consultas.
   With rstCincoMásVendidos
      strMensaje = _
         "Nuestros 5 libros mejor vendidos son:" & vbCr

      Do While Not .EOF
         strMensaje = strMensaje & "    " & !Rítulo & _
            vbCr
         .MoveNext
      Loop

      If .RecordCount > 5 Then
         strMensaje = strMensaje & _
            "(Había un vínculo, resultan " & _
            vbCr & .RecordCount & _
            " libros en la lista.)"
      End If

      MsgBox strMensaje
      .Close
   End With
```

```
    ' Elimina la nueva consulta de paso a través porque esto
    ' es un ejemplo.
    dbsActual.QueryDefs.Delete "TodoslosTítulos"
    dbsActual.Close

End Sub
```

Ejemplo de los métodos CreateQueryDef, OpenRecordset y la propiedad SQL (Cliente/Servidor)

Este ejemplo utiliza los métodos **CreateQueryDef** y **OpenRecordset** y la propiedad **SQL** para consultar la tabla de títulos de la base de datos de muestra Ediciones de Microsoft SQL Server y devuelve el Título y el identificador del Título del libro más vendido. El ejemplo consulta después la tabla de autores y le pide al usuario que le envíe un cheque de bonificación a cada autor según el porcentaje de sus derechos (la bonificación total es de $1000 y cada autor debe recibir un porcentaje de esa cantidad).

```
Sub ClienteServidorX2()

   Dim dbsActual As Database
   Dim qdfMásVendidos As QueryDef
   Dim qdfIngresos As QueryDef
   Dim rstMásVendido As Recordset
   Dim rstDestinatarioIngreso As Recordset
   Dim strListaAutores As String

   ' Abre una base de datos de la que se pueden crear los
   ' objetos QueryDef.
   Set dbsActual = OpenDatabase("DB1.mdb")

   ' Crea un objeto QueryDef temporal para recuperar datos de
   ' una base de datos Microsoft SQL Server.
   Set qdfMásVendidos = dbsActual.CreateQueryDef("")
   With qdfMásVendidos
      .Connect = "ODBC;DATABASE=eds;UID=sa;PWD=;" & _
          "DSN=Publicaciones"
      .SQL = "SELECT Título, Título_id FROM títulos " & _
         "ORDER BY ventas_año DESC"
      Set rstMásVendido = .OpenRecordset()
      rstMásVendido.MoveFirst
   End With

   ' Crea un objeto QueryDef temporal para recuperar datos de
   ' una base de datos Microsoft SQL Server basada en los resultados
   ' de la primera consulta.
   Set qdfIngresos = dbsActual.CreateQueryDef("")
   With qdfIngresos
      .Connect = "ODBC;DATABASE=eds;UID=sa;PWD=;" & _
         "DSN=Publicaciones"
      .SQL = "SELECT * FROM tituloautor " & _
         "WHERE Título_id = '" & _
         rstMásVendido!Título_id & "'"
      Set rstDestinatarioIngreso = .OpenRecordset()
   End With

   ' Genera la cadena de salida.
   With rstDestinatarioIngreso
      Do While Not .EOF
         strListaAutores = strListaAutores & "    " & _
            !au_id & ":  $" & (10 * !tipoimpuesto) & vbCr
         .MoveNext
      Loop
   End With
```

```
    ' Muestra los resultados.
    MsgBox "Enviar un cheque a los siguientes " & _
       "autores del importe indicado:" & vbCr & _
       strListaAutores & "por ventas pendientes de " & _
       rstMásVendido!Rítulo & "."

    rstMásVendido.Close
    dbsActual.Close

End Sub
```

Ejemplo de los métodos CreateTableDef, FillCache, y de las propiedades CacheSize, CacheStart y SourceTableName (Cliente/Servidor)

Este ejemplo utiliza los métodos **CreateTableDef** y **FillCache** y las propiedades **CacheSize**, **CacheStart** y **SourceTableName** para enumerar dos veces los registros en una tabla vinculada. Después enumera los registros dos veces con una memoria caché de 50 registros. El ejemplo muestra a continuación las estadísticas de rendimiento para la ejecución con almacenamiento en memoria caché y sin él con tablas vinculadas.

```
Sub ClienteServidorX3()

   Dim dbsActual As Database
   Dim tdfDerechos As TableDef
   Dim rstRemoto As Recordset
   Dim sngInicio As Single
   Dim sngFin As Single
   Dim sngSinCaché As Single
   Dim sngCaché As Single
   Dim intBucle As Integer
   Dim strTemp As String
   Dim intRegistros As Integer

   ' Abre una base de datos a la que se agregó
   ' una tabla vinculada.
   Set dbsActual = OpenDatabase("DB1.mdb")

   ' Crea una tabla vinculada que se conecta a una base de datos
   ' Microsoft SQL Server.
   Set tdfDerechos = _
      dbsActual.CreateTableDef("Derechos")
   tdfDerechos.Connect = _
      "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Publicaciones"
   tdfDerechos.SourceTableName = "EsqDer"
   dbsActual.TableDefs.Append tdfDerechos
   Set rstRemoto = _
      dbsActual.OpenRecordset("Derechos")

   With rstRemoto
      ' Enumera el objeto Recordset dos veces y graba
      ' el tiempo transcurrido.
      sngInicio = Timer

      For intBucle = 1 To 2
         .MoveFirst
         Do While Not .EOF
            ' Ejecuta una operación sencilla para la
            ' prueba de rendimiento.
            strTemp = !Título_id
            .MoveNext
         Loop
      Next intBucle

      sngFin = Timer
      sngSinCaché = sngFin - sngInicio

      ' Almacena en memoria caché los primero 50 registros.
```

```
      .MoveFirst
      .CacheSize = 50
      .FillCache
      sngInicio = Timer

      ' Enumera el objeto Recordset dos veces y graba
      ' el tiempo transcurrido.
      For intBucle = 1 To 2
         intRegistros = 0
         .MoveFirst
         Do While Not .EOF
            ' Ejecuta una operación sencilla para la
            ' prueba de rendimiento.
            strTemp = !Título_id
            ' Cuenta los registros. Si se encuentra el fin de la memoria
            ' caché, restablece la memoria caché a los 50 registros
            ' siguientes.
            intRegistros = intRegistros + 1
            .MoveNext
            If intRegistros Mod 50 = 0 Then
               .CacheStart = .Bookmark
               .FillCache
            End If
         Loop
      Next intBucle

      sngFin = Timer
      sngCaché = sngFin - sngInicio

      ' Muestra los resultados de rendimiento.
      MsgBox "Resultados del rendimiento de la memoria caché:" & vbCr & _
         "    Sin caché: " & Format(sngSinCaché, _
         "##0.000") & " segundos" & vbCr & _
         "    Caché de 50 registros: " & Format(sngCaché, _
         "##0.000") & " segundos"
      .Close
   End With

   ' Elimina la tabla vinculada porque esto es un ejemplo.
   dbsActual.TableDefs.Delete tdfDerechos.Name
   dbsActual.Close

End Sub
```