



Guía de Base

Capítulo 9

Macros

Derechos de autor

Este documento tiene derechos de autor © 2021 por el equipo de documentación. Los colaboradores se listan más abajo. Se puede distribuir y modificar bajo los términos de la [GNU General Public License](#) versión 3 o posterior o la [Creative Commons Attribution License](#), versión 4.0 o posterior.

Todas las marcas registradas mencionadas en esta guía pertenecen a sus propietarios legítimos.

Colaboradores

Este libro está adaptado de versiones anteriores del mismo.

De esta edición

Pulkit Krishna	Alain Romedenne	Jean-Pierre Ledure
Jean Hollis Weber		
Juan Peramos	Juan Carlos Sanz Cabrero	B. Antonio Fernández
Ignacio Alcalá		

De ediciones previas

Jochen Schiffers	Robert Großkopf	Jost Lange
Hazel Russman	Jean Hollis Weber	

Comentarios y sugerencias

Puede dirigir cualquier clase de comentario o sugerencia acerca de este documento a: documentation@es.libreoffice.org.



Nota

Todo lo que envíe a la lista de correo, incluyendo su dirección de correo y cualquier otra información personal que escriba en el mensaje se archiva públicamente y no puede ser borrado.

Fecha de publicación y versión del programa

Versión en español publicada el 30 de julio de 2021. Basada en la versión 6.2 de LibreOffice.

Uso de LibreOffice en macOS

Algunas pulsaciones de teclado y opciones de menú son diferentes en macOS de las usadas en Windows y Linux. La siguiente tabla muestra algunas sustituciones comunes para las instrucciones dadas en este capítulo. Para una lista detallada vea la ayuda de la aplicación.

<i>Windows o Linux</i>	<i>Equivalente en Mac</i>	<i>Efecto</i>
Herramientas > Opciones opción de menú	LibreOffice > Preferencias	Acceso a las opciones de configuración
<i>Clic con el botón derecho</i>	<i>Control+clic</i> o <i>clic derecho</i> depende de la configuración del equipo	Abre menú contextual
<i>Ctrl (Control)</i>	⌘ (<i>Comando</i>)	Utilizado con otras teclas
<i>F5</i>	<i>Mayúscula+⌘+F5</i>	Abre el navegador
<i>F11</i>	⌘+T	Abre la ventana de estilos y formato

Contenido

Derechos de autor	2
Colaboradores.....	2
De esta edición.....	2
De ediciones previas.....	2
Comentarios y sugerencias.....	2
Fecha de publicación y versión del programa.....	2
Uso de LibreOffice en macOS	2
Observaciones generales sobre macros	6
Macros en Base	7
Usar macros.....	7
Asignar macros.....	8
Sucesos que se desencadenan en un formulario cuando la ventana se abre o se cierra.....	8
Sucesos dentro de una ventana de un formulario.....	8
Sucesos dentro de un formulario.....	9
Componentes de macros.....	10
La estructura de una macro.....	10
Uso de variables.....	10
Uso de matrices.....	11
Acceso a formularios.....	12
Acceso a elementos de formulario.....	13
Acceso a la base de datos.....	14
Leer y usar registros.....	16
Editar registros: agregar, modificar, eliminar.....	19
Prueba y cambio de controles.....	21
Nombres de las propiedades de los controles en macros.....	22
Propiedades de formularios y controles.....	22
Métodos para formularios y controles.....	29
Mejorar la facilidad de uso	33
Actualización automática de formularios.....	33
Filtrar registros.....	34
Preparar datos para campos de texto que se ajusten a convenciones SQL.....	37
Calcular valores anticipadamente.....	38
Proporcionar la versión actual de LibreOffice.....	39
Obtener el valor devuelto por los listados.....	40
Limitar el contenido de los listados ingresando letras iniciales.....	41
Convertir fechas de un formulario en una variable de tipo fecha.....	43
Buscar registros de datos.....	43
Resaltar términos de búsqueda en formularios y resultados.....	46
Revisar la ortografía durante la entrada de datos.....	50
Cuadros combinados, como listados con opción de entrada.....	52
Visualizar texto en cuadros combinados.....	53
Transferir un valor de clave foránea de un cuadro combinado a un campo numérico.....	55
Función para medir la longitud de la entrada del cuadro combinado.....	63
Generar acciones de base de datos.....	63
Navegar de un formulario a otro.....	63
Listados jerárquicos.....	65

Ingresar horas con milisegundos.....	69
Un suceso: varias implementaciones.....	70
Guardar con confirmación.....	71
Clave primaria con año en curso y número.....	71
Ampliación de tareas de bases de datos mediante macros.....	73
Crear una conexión a una base de datos.....	73
Copiar datos de una base de datos a otra.....	74
Acceso a consultas.....	75
Asegurar la base de datos.....	76
Compactar la base de datos.....	79
Disminuir el índice de la tabla para los campos automáticos.....	80
Imprimir desde Base.....	81
Imprimir un informe desde un formulario interno.....	81
Lanzar, formatear, imprimir directamente y cerrar un informe.....	81
Imprimir informes desde un formulario externo.....	84
Combinación de correspondencia desde Base.....	84
Imprimir mediante campos de texto.....	85
Llamar a aplicaciones externas a LibreOffice.....	87
Llamada a aplicaciones para abrir archivos.....	87
Llamada a un programa en función del texto introducido por el usuario.....	87
Llamada a un programa de correo con contenido predefinido.....	88
Cambiar el puntero del ratón al situarse sobre un enlace.....	90
Mostrar formularios sin una barra de herramientas.....	90
Formularios sin barra de herramientas en la ventana.....	90
Formularios en modo de pantalla completa.....	92
Lanzar formularios directamente desde la apertura de la base de datos.....	93
Acceder a una base de datos MySQL con macros.....	93
Código MySQL en macros.....	94
Tablas temporales como almacenamiento intermedio individual.....	94
Diálogos.....	94
Iniciar y finalizar diálogos.....	95
Diálogo sencillo para ingresar nuevos registros.....	96
Diálogo para editar registros en una tabla.....	97
Usar un diálogo para limpiar entradas incorrectas en tablas.....	104
Escribir macros con Access2Base.....	113
El modelo Objeto.....	114
Algunos ejemplos.....	114
Imprimir una lista de nombres de tablas y campos.....	114
Almacenar los datos producidos por una consulta en una matriz básica.....	115
Establecer valores predeterminados en entradas de formulario.....	115
Funciones de base de datos.....	115
Comandos especiales.....	115

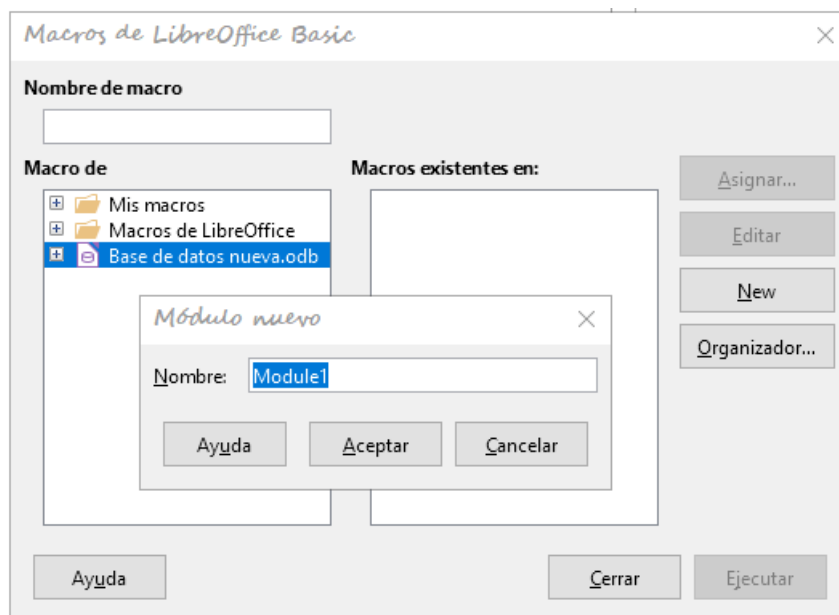
Observaciones generales sobre macros

En principio, una base de datos en Base se puede administrar sin macros. Aunque a veces, pueden ser necesarios para:

- Prevención más efectiva de errores de entrada.
- Simplificar ciertas tareas de procesamiento (cambiar de un formulario a otro, actualizar datos después de ingresarlos en un formulario, etc.).
- Permitir que ciertas órdenes SQL se ejecuten más fácilmente que con el editor SQL.

Debe decidir por sí mismo con qué intensidad desea usar macros en Base. Las macros pueden facilitar el uso, pero siempre están asociadas con pequeñas reducciones en la velocidad de respuesta del programa y, a veces, ralentizan (cuando están mal codificadas). Siempre es mejor comenzar utilizando todas las posibilidades de la base de datos y sus capacidades para configurar los formularios antes de intentar proporcionar una funcionalidad adicional con macros. Las macros siempre deben probarse en bases de datos grandes para determinar su efecto en el rendimiento.

Para la creación y el acceso a las macros se utiliza el diálogo *Macros de LibreOffice Basic* > al que se accede mediante **Herramientas > Macros > Organizar macros > LibreOffice Basic** del menú principal.



Para que las macros estén disponibles en todo momento para la base de datos, en la que está trabajando, es necesario que se guarden en el archivo de la base de datos seleccionándolo en el área de la izquierda (*Macro de*).

Pulse en el botón *Nuevo* en el diálogo y se abrirá el diálogo *Módulo nuevo*, que solicita el nombre del módulo (el contenedor dónde se escribirán los procedimientos).

En cuanto se hace esto, aparece la *interfaz de programación Basic*. En el *área de edición* aparecerá el inicio y el fin para que escriba un procedimiento:

```
REM ***** BASIC *****  
Sub Main  
...  
End Sub
```

Para poder utilizar las macros, en LibreOffice, es necesario seguir los siguientes pasos:

- En **Herramientas > Opciones > Seguridad > Seguridad de macros:** >El *Nivel de seguridad* debe reducirse a *Medio*. También es posible usar la pestaña *Orígenes de*

confianza para elegir certificados de confianza o establecer la ruta a sus archivos de macro y así evitar las ventanas emergentes relativas a la activación de macros al abrir los archivos.

- Después de la creación del primer módulo, debe guardar el archivo, cerrarlo y luego volver a abrirlo.

Algunos principios básicos para el uso del código basic de LibreOffice:

- Las Instrucciones deben terminar con un retorno de línea. Aunque las líneas no están numeradas de forma predeterminada (existe una opción para activar la numeración).
- Las funciones, expresiones reservadas y elementos similares no distinguen entre mayúsculas y minúsculas. Así pues, *String* es lo mismo que *STRING*, *string* o cualquier otra combinación de mayúsculas y minúsculas (el uso de mayúsculas se debe emplear para mejorar la legibilidad). Sin embargo, los nombres de constantes y enumeraciones si distinguen entre mayúsculas y minúsculas la primera vez que los ve el compilador de macros, por lo que es mejor acostumbrarse a escribir siempre de una manera adecuada.
- Se puede distinguir entre *Procedimientos* (que comienzan con **Sub**) y *Funciones* (que comienzan con **Function**). Los procedimientos fueron originalmente trozos de programa sin un valor de retorno, mientras que las funciones devuelven valores que se pueden procesar posteriormente. Pero esta distinción se está volviendo cada vez más irrelevante. Hoy en día, las personas usan términos como *método* o *rutina* para designar a los procedimientos, tanto si hay un valor de retorno o no. Un procedimiento también puede tener un valor de retorno (distinto del tipo **Variant**).

```
Sub miProcedimiento As Integer
```

```
...
```

```
End Sub
```

Para obtener información más detallada, consulte el «Capítulo 13, Introducción a las macros», en la *Guía de primeros pasos con Base*.



Nota

Las macros en las versiones PDF y ODT de este capítulo están coloreadas de acuerdo con las reglas del editor de macros LibreOffice:

Comentario

Identificador

Expresión reservada

Cadena de texto

Número

Operador

Macros en Base

Usar macros

Aunque es posible utilizar la forma directa para ejecutar las macros, usando **Herramientas > Macros > Ejecutar macro**, esto no es habitual para las macros que se usan en Base. Normalmente la macro se asigna a un suceso y esta se inicia cuando se produce el desencadenante.

Las macros se usan para:

- Gestionar los sucesos en formularios
- Editar datos dentro de un formulario

- Cambiar entre controles de formulario
- Reaccionar ante una acción del usuario dentro de un control.

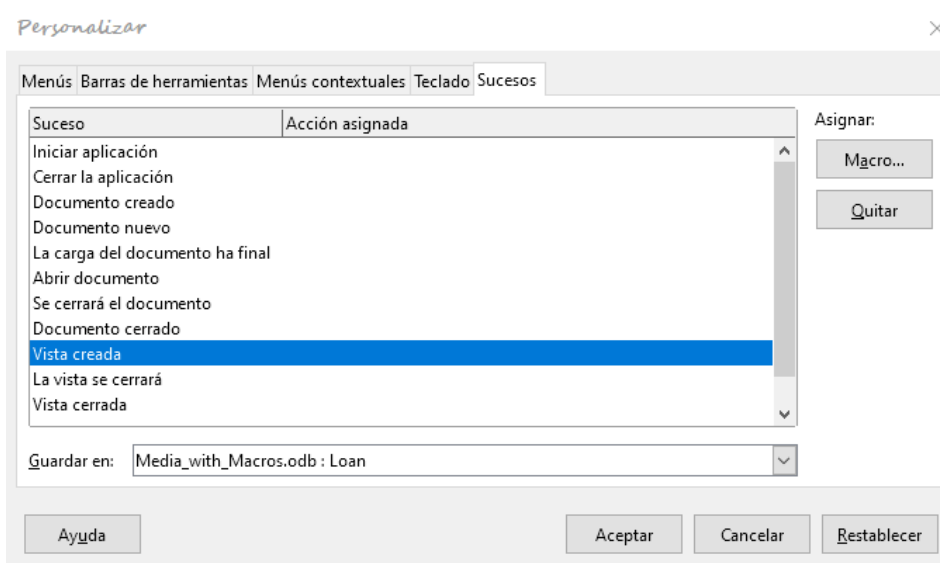
Cuando se utiliza alguno de los objetos de **ThisComponent** u **oEvent** no es posible utilizar la forma directa, ni siquiera para pruebas, (consulte «Acceso a formularios» en la página 12 y también «Acceso a elementos de formulario» en la página 13).

Asignar macros

Si se quiere iniciar una macro, respondiendo a un suceso, primero se tiene que definir esa macro, de manera que se pueda asignar a ese suceso. Se puede acceder a los sucesos a través de dos ubicaciones, en función de su desencadenante:

Sucesos que se desencadenan en un formulario cuando la ventana se abre o se cierra

Las acciones que tienen lugar cuando se abre o cierra un formulario se registran de la siguiente manera:



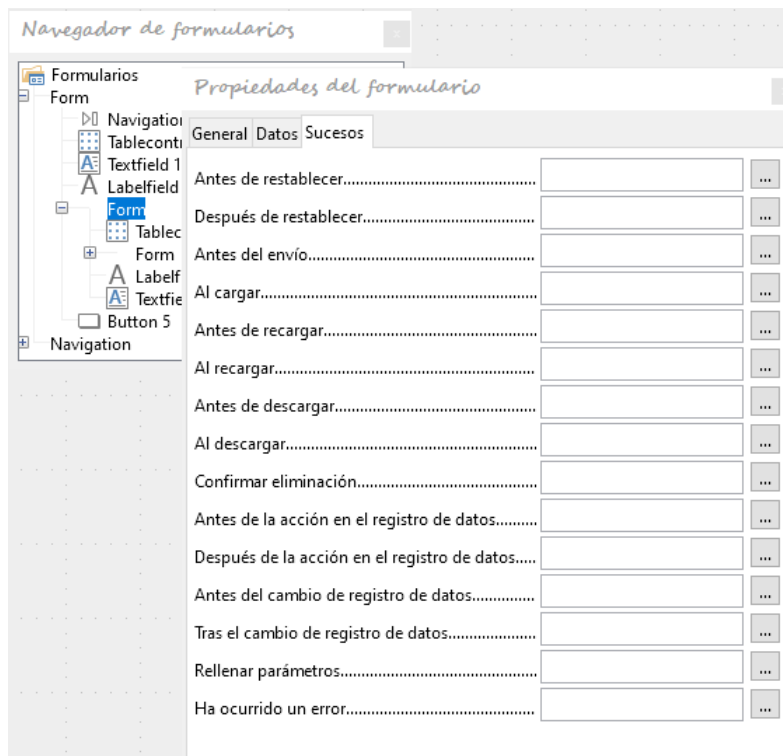
- 1) Durante la creación de un formulario, utilice el menú **Herramientas > Personalizar** y elija la pestaña **Sucesos**.
- 2) Elija el suceso apropiado. Algunas macros solo se pueden iniciar cuando se elige el suceso *Vista creada*. Otras macros, por ejemplo para crear un formulario en pantalla completa se deben iniciar con el suceso *Abrir documento*.
- 3) Use el botón *Macro* para buscar la macro que desea aplicar y confirme su elección.
- 4) En la parte inferior *Guardar en*, indique el nombre del formulario.
- 5) Finalmente, confirme con *Aceptar*.

Sucesos dentro de una ventana de un formulario

Una vez abierta la ventana para editar el contenido general del formulario, se puede acceder a los elementos individuales del mismo. Incluidos los elementos que haya asignado al formulario.

Se puede acceder a los elementos del formulario utilizando el *Navegador de formularios*, como se muestra en la siguiente ilustración, o también mediante el menú contextual de los controles dentro de la interfaz del formulario.

En la pestaña de *Sucesos* de *Propiedades del formulario*, los Sucesos enumerados tienen lugar mientras la ventana del formulario está abierta. Se puede configurar por separado para cada formulario o subformulario durante la edición del formulario.



Nota

Desafortunadamente, Base usa la palabra formulario tanto para una ventana que se abre para la entrada de datos, como para los algunos elementos dentro de esta ventana que están vinculados a un origen de datos específico (tabla o consulta).

Una sola ventana de formulario puede contener varios subformularios con diferentes fuentes de datos. En el *Navegador de formularios*, siempre se visualiza primero el término *Formularios* con diversas entradas de formularios y controles. En el caso de un formulario simple contiene solo una entrada subordinada.

Sucesos dentro de un formulario

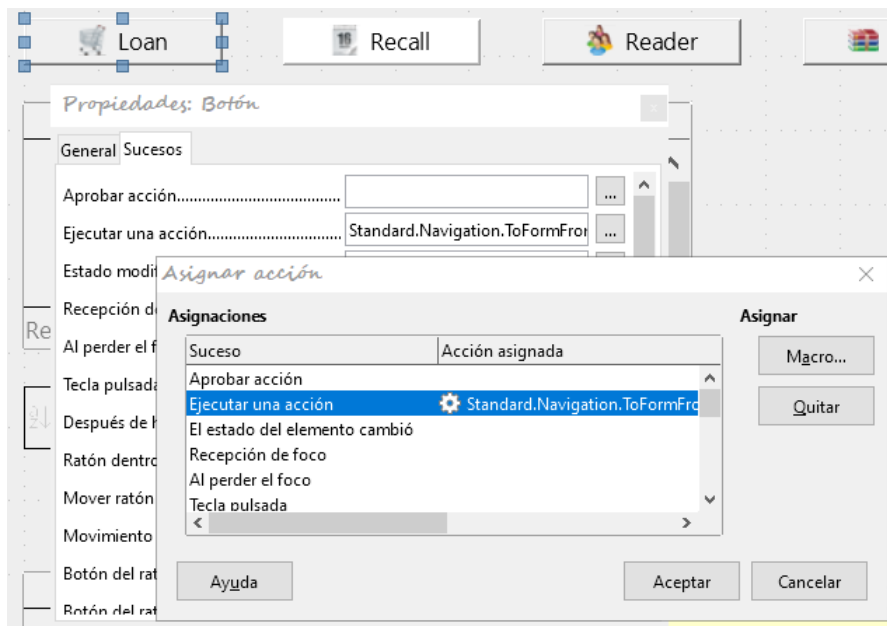
Todas las demás macros se registran utilizando las propiedades de subformularios y controles a través de la pestaña *Sucesos*.

- 1) Abra la ventana para las propiedades del control.
- 2) Elija un suceso adecuado en la pestaña *Sucesos*.
- 3) Para los encargados de editar la fuente de datos, use sucesos que se refieran a *Grabar*, *Actualizar*, o *Restablecer*.

Para los botones, o las opciones dentro de los campos de lista u opción, utilice *Ejecutar una acción*.

Todos los demás sucesos dependen del tipo de control y la acción deseada.

- 4) Haga clic en el botón (...) a la derecha del suceso para abrir el diálogo *Asignar acción*.
- 5) Haga clic en el botón *Macro* para elegir la macro específica para esa acción.
- 6) Haga clic en *Aceptar* para confirmar la asignación.



Componentes de macros

Esta sección explica parte del lenguaje Basic para macros que se usa comúnmente en Base, especialmente dentro de los formularios. En las siguientes secciones se muestran varios ejemplos en la medida de lo posible (y razonable).

La estructura de una macro

La definición de un procedimiento comienza con su tipo `Sub` o `Function` y termina con `End Sub` o `End Function` respectivamente. Una macro asignada a un suceso puede recibir argumentos (valores); el único útil es el argumento `oEvent`. Todos los procedimientos a los que puede llamar una macro pueden definirse con o sin un valor de retorno, dependiendo de su propósito, y estar provista de argumentos si es necesario.

```
Sub update_loan
End Sub
```

```
Sub from_Form_to_Form(oEvent As Object)
End Sub
```

```
Function confirm_delete(oEvent As Object) As Boolean
    confirm_delete = False
End Function
```

Es útil empezar con este marco (inicio y final) e incluir el contenido después. No olvide agregar comentarios para explicar la macro, recordando la regla: «*Tantos como sea necesario, tan pocos como sea posible*».

Basic no distingue entre mayúsculas y minúsculas salvo en casos especiales. Por lo general, los términos fijos como `SUB` se escriben preferiblemente en mayúsculas o mayúscula inicial, otros conceptos con mayúsculas y minúsculas.

Uso de variables

En el siguiente paso, al comienzo de la rutina, la instrucción `Dim` se usa para definir las variables que se utilizarán en el procedimiento, cada una con su tipo de datos apropiado. Aunque la programación en Basic no necesita las definiciones (acepta cualquier nueva variable que se cree).

El código de la macro es más seguro si se declaran las variables, y especialmente si se indica el tipo de datos.

Muchos programadores utilizan `option Explicit` al principio de un módulo. Con esta opción se aseguran que todas las variables tengan que estar declaradas de antemano, si una variable no se ha declarado el programa mostrará un error y detendrá su ejecución.

```
Dim oDoc As Object
Dim oDrawpage As Object
Dim oForm As Object
Dim sName As String
Dim bOKEnabled As Boolean
Dim iCounter As Integer
Dim dBirthday As Date
```

Los nombres de las variables deben empezar siempre por un carácter alfabético, solo se pueden usar caracteres alfabéticos (A-Z o a-z), números y el carácter de subrayado (_). No se permiten caracteres especiales y aunque se permiten espacios bajo determinadas condiciones, es mejor evitarlos.

Una práctica común y muy recomendable, es utilizar el primer carácter para especificar el tipo de datos¹. De esta manera se distingue claramente el tipo de datos que contiene la variable en cualquier lugar que aparezca, evitando errores al procesarla. También se recomienda usar nombres explicativos, para que el contenido de la variable sea obvio.

Puede encontrar una lista de tipos de datos admisibles por *Star Basic* en el «Apéndice A» de esta guía. Difieren de algún modo de los tipos propios de la base de datos y de la API de LibreOffice. Estos cambios se verán más claros en los ejemplos

Uso de matrices

Para bases de datos en particular, el agrupamiento de varias variables en un registro es importante. Una matriz es un conjunto de varias variables que se almacenan juntas en una única ubicación común. Se debe definir una matriz antes de que se puedan escribir datos en ella.

```
Dim arData()
```

Crea una matriz vacía.

```
arData = Array("Lisa", "Schmidt")
```

Crea una matriz de un tamaño específico (2 elementos) y le proporciona valores.

```
Print arData(0), arData(1)
```

Muestra los dos elementos asignados en pantalla. El recuento de elementos comienza con 0.

```
Dim arData(2)
arData(0) = "Lisa"
arData(1) = "Schmidt"
arData(2) = "Cologne"
```

Crea una matriz en la que se pueden almacenar tres elementos de cualquier tipo, por ejemplo, un registro que contiene "Lisa" "Schmidt" "Cologne". No puede poner más de tres elementos en esta matriz.

1 Algunas veces, una sólo letra no le permite distinguir entre los tipos de datos (Double y Data o "Single" y String). Utilice las necesarias para evitar confusiones.

Si desea almacenar más elementos, debe agrandar la matriz con `ReDim`. Sin embargo, si el tamaño de una matriz se redefine mientras se ejecuta una macro, la matriz queda vacía como si fuera una nueva matriz.

```
ReDim Preserve arData(3)
arData(3) = "18.07.2003"
```

Al agregar `Preserve` se conservan los datos anteriores y la matriz se extenderá agregando un cuarto elemento: la fecha (en este caso en forma de texto).

La matriz que se muestra arriba solo puede almacenar un registro. Si se desea almacenar varios registros, como lo hace una tabla, hay que definir una matriz bidimensional.

```
Dim arData(2,1) '3 elementos, en cada fila(2 filas) >
arData(0,0) = "Lisa" 'primer elemento de la primera fila
arData(1,0) = "Schmidt" 'segundo elemento de la primera fila
arData(2,0) = "Cologne"
arData(0,1) = "Egon" 'primer elemento de la segunda fila
arData(1,1) = "Müller"
arData(2,1) = "Hamburg"
```

Aquí también es posible extender la matriz previamente definida y preservar los contenidos existentes utilizando `Preserve`.

Acceso a formularios

El formulario se encuentra en el documento actualmente activo. La región que se representa se llama *drawpage*. El contenedor en el que se guardan todos los formularios se llama *forms*; en el *Navegador de formularios*, se muestra como el encabezado principal con todos los formularios individuales adjuntos. Las variables mencionadas anteriormente reciben sus valores de esta manera:

```
oDoc = thisComponent 'acceso al documento
oDrawpage = oDoc.drawpage 'acceso a la región de formularios
oForms = oDrawpage.forms 'acceso a los formularios
oForm = oForms.getByName("Filter") 'acceso al formulario filter
```

Todo esto se puede resumir en una sola variable:

```
oForm = ThisComponent.drawpage.forms.getByName("Filter")
```

El formulario al que se accede en esta ocasión se llama *Filter*. Este es el nombre que está visible en el nivel superior del Navegador de formularios (de forma predeterminada, el primer formulario se llama *MainForm*).

Los subformularios se encuentran en orden jerárquico dentro del formulario principal y se puede llegar paso a paso:

```
Dim oSubForm As Object
Dim oSubSubForm As Object
oSubForm = oForm.getByName("Readerselect")
oSubSubForm = oSubForm.getByName("Readerdisplay")
```

En lugar de usar valores intermedios, puede ir directamente a un formulario concreto. Un objeto intermedio, que se puede usar más de una vez, debe declararse y asignarle un valor separado. En el siguiente ejemplo, *oSubForm* ya no se usa.

```
oForm = thisComponent.drawpage.forms.getByName("Filter")
oSubSubForm =
```

```
oForm.getByName("readerselect").getByName("readerdisplay")
```



Nota

Si un nombre de formulario contiene únicamente letras ASCII y números sin espacios ni caracteres especiales, se puede usar directamente en asignación a una variable.

```
oForm = thisComponent.drawpage.forms.Filter  
oSubSubForm = oForm.readerselect.readerdisplay
```

Contrariamente al uso normal en Basic, los nombres de formularios, y de los controles son sensibles al uso de mayúsculas.

El suceso que desencadena la macro proporciona un modo diferente de acceso al formulario. Si se inicia una macro desde un suceso de formulario, como por ejemplo *Antes de la acción de registro*, se puede acceder al formulario utilizando el método `oEvent` de la siguiente manera:

```
Sub MacroexampleCalc(oEvent As Object)  
    oForm = oEvent.Source  
    ...  
End Sub
```

Si la macro se inicia desde un suceso en un control de formulario, como por ejemplo en un *Cuadro de texto*, *Al perder el foco*, tanto el formulario como el campo serán accesibles:

```
Sub MacroexampleCalc(oEvent As Object)  
    oField = oEvent.Source.Model  
    oForm = oField.Parent  
    ...  
End Sub
```

El acceso a los sucesos tiene la ventaja de que no necesita preocuparse si se trata de un formulario principal o un subformulario. Además, el nombre del formulario no es importante para el funcionamiento de la macro.

Acceso a elementos de formulario

Se accede a los elementos dentro de los formularios de manera similar: declare una variable como objeto y asigne el control apropiado dentro del formulario:

```
Dim btnOK As Object 'se declara la variable para el botón OK  
btnOK = oSubSubForm.getByName("button 1") ' pertenece al  
formulario readerdisplay
```

Este método siempre funciona cuando se sabe con qué elemento debe iniciarse la macro. Sin embargo, cuando se debe determinar qué suceso inició la macro, el método `oEvent` mencionado anteriormente es más útil.

La variable se declara en la definición de la macro y se le asigna un valor cuando se inicia la macro. La propiedad `Source` siempre devuelve el elemento que lanzó la macro, mientras que la propiedad `Model` describe el control en detalle:

```
Sub confirm_choice(oEvent As Object)  
    Dim btnOK As Object  
    btnOK = oEvent.Source.Model  
End Sub
```

Si se desea, se pueden realizar más acciones con el objeto obtenido por este método. Tenga en cuenta que los subformularios cuentan como componentes de un formulario.

Acceso a la base de datos

Normalmente, el acceso a la base de datos está controlado por *formularios*, *consultas*, *informes* o la *combinación de correspondencia*, como se describe en los capítulos anteriores. Si estas posibilidades resultan insuficientes, una macro puede acceder específicamente a la base de datos de varias maneras.

Conectar con la base de datos

El método más sencillo usa la misma conexión que el formulario. *oForm* se define como se indicó antes.

```
Dim oConnection As Object
oConnection = oForm.activeConnection()
```

O se puede buscar la fuente de datos (es decir, la base de datos) a través del documento y usar su conexión existente en la macro.

```
Dim oDatasource As Object
Dim oConnection As Object
oDatasource = thisComponent.Parent.dataSource
oConnection = oDatasource.getConnection("", "")
```

Una forma adicional que permite que la conexión se cree sobre la marcha sería:

```
Dim oDatasource As Object
Dim oConnection As Object
oDatasource = thisComponent.Parent.CurrentController
If Not (oDatasource.isConnected()) Then oDatasource.connect()
oConnection = oDatasource.ActiveConnection()
```

Aquí la condición `If` utiliza solo una línea, por lo que no se necesita acabarla con `End If`.

Si la macro se va a iniciar a través de la interfaz de usuario y no desde un suceso en un formulario, se puede usar la siguiente variante:

```
Dim oDatasource As Object
Dim oConnection As Object
oDatasource = thisDatabaseDocument.CurrentController
If Not (oDatasource.isConnected()) Then oDatasource.connect()
oConnection = oDatasource.ActiveConnection()
```

Se puede acceder a otras bases de datos que estén registradas en LibreOffice de la siguiente manera:

```
Dim oDatabaseContext As Object
Dim oDatasource As Object
Dim oConnection As Object
oDatabaseContext =
createUnoService("com.sun.star.sdb.DatabaseContext")
oDatasource = oDatabaseContext.getByname("nombre registrado de la
Base de datos")
oConnection = oDatasource.GetConnection("", "")
```

También son posibles las conexiones a bases de datos no registradas con LibreOffice. En estos casos, en lugar del nombre registrado, se debe proporcionar la ruta y nombre de la base de datos como archivo: `:///...../database.odt`.

Pueden verse más explicaciones sobre las conexiones de la base de datos en «Crear una conexión a una base de datos» en la página 73.

Órdenes SQL

Para trabajar con la base de datos se utilizan órdenes SQL. Estas órdenes deben ser creadas y enviadas a la base de datos; (el resultado se determina según el tipo de orden) los resultados obtenidos se pueden seguir procesando. La directiva `createStatement` crea un objeto adecuado para este propósito.

```
Dim oSQL_Statement As Object ' el objeto que llevará a cabo la orden SQL
Dim stSql As String ' Texto de la actual orden SQL
Dim oResult As Object ' resultado de executeQuery
Dim iResult As Integer ' resultado de executeUpdate
oSQL_Statement = oConnection.createStatement()
```

Para consultar datos, utilice el método `executeQuery`. Los nombres de tablas y campos suelen estar entre comillas dobles. La macro debe enmascararlos con comillas dobles adicionales para garantizar que aparezcan en la orden.

```
stSql = "SELECT * FROM ""Table1"""
oResult = oSQL_Statement.executeQuery(stSql)
```

Para modificar datos, es decir, `INSERT`, `UPDATE` o `DELETE`, o para influir en la estructura de la base de datos, llame al método `executeUpdate`. Dependiendo de la orden y la base de datos, esto no produce nada útil, o un cero, o el número de registros modificados.

```
stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
iResult = oSQL_Statement.executeUpdate(stSql)
```

En aras de la exhaustividad, hay un caso especial más que se debe mencionar: el método `execute` que es útil si la directiva `createStatement` se va a utilizar de diferentes maneras para `SELECT` o para otros fines, pero no lo usaremos aquí. Para obtener más información, consulte la Referencia de la API.

Órdenes SQL previamente preparadas con parámetros

En todos los casos en que la entrada manual de un usuario debe transferirse a una instrucción SQL, es más fácil y seguro no crear la orden como una cadena de caracteres porque puede ser muy larga, sino prepararla de antemano y usarla con parámetros. Esto facilita el formateo de números, fechas y cadenas, se evitan las repeticiones de las comillas dobles y evita que una frase mal formada provoque la pérdida de datos.

Para usar este método, se crea y se prepara un objeto para una orden SQL determinada :

```
Dim oSQL_Statement As Object ' el objeto, para ejecutar la orden SQL
Dim stSql As String ' Texto de la orden SQL
stSql = "UPDATE author " _
& "SET lastname = ?, firstname = ?" _
& "WHERE ID = ?"
oSQL_Statement = oConnection.prepareStatement(stSql)
```

El objeto se crea con `prepareStatement` para que la orden SQL se conozca de antemano. Cada signo de interrogación indica una posición que, antes de ejecutar la orden, recibirá un valor

real. Debido a que la orden se prepara por adelantado, la base de datos sabe cual es el tipo de entrada, en este caso, se esperan dos cadenas de texto y un número. Las distintas entradas se distinguen por el número de su posición (contando desde 1).

Luego, los valores se transfieren con declaraciones adecuadas y se ejecuta la orden SQL. Los valores se toman de los controles de formulario, pero también pueden originarse en otras macros o darse como texto sin formato:

```
oSQL_Statement.setString(1, oTextfeld1.Text) ' Texto para lastname
oSQL_Statement.setString(2, oTextfeld2.Text) ' Texto para first name
oSQL_Statement.setLong(3, oNumericfield1.Value) ' valor para el ID
iResult = oSQL_Statement.executeUpdate
```

La lista completa de métodos está en "Parámetros para órdenes SQL preparadas" (página 33).

Para obtener más información sobre las ventajas de este método, consulte los siguientes enlaces externos:

- [SQL-Injection \(Wikipedia\)](#)
- [Inyección SQL](#)
- [Why use PreparedStatement \(Java JDBC\)](#)
- [SQL-commands \(Introduction to SQL\)](#)
- [MySQL Gestión Lenguajes de consulta y extracción de datos](#)

Leer y usar registros

Existen varias formas de transferir información de una base de datos a una macro para que esta información se pueda procesar.

Tenga en cuenta que las referencias a un formulario incluyen subformularios. se entiende que ese formulario o parte del formulario está vinculado a una fuente de datos en particular.

Usar formularios

El registro actual y sus datos siempre están disponibles a través del formulario que muestra los datos relevantes (*tabla, consulta, SELECT*). Existen varios métodos para obtener los datos (*getdata_type*), en este ejemplo:

```
Dim ID As Long
Dim sName As String
Dim dValue AS Currency
Dim dEntry As New com.sun.star.util.Date
ID = oForm.getLong(1)
sName = oForm.getString(2)
dValue = oForm.getDouble(4)
dEntry = oForm.getDate(7)
```

Todos estos métodos requieren el número de columna de un origen de datos; su cuenta comienza en 1.



Nota

Para todos los métodos que funcionan con bases de datos, se empiezan a contar desde 1. Esto se aplica tanto a las columnas como a las filas.

Si se prefiere usar nombres de columna en lugar de números para trabajar con la fuente de datos (*tabla, consulta, vista*), el número de columna se puede sustituir por su nombre, usando `findColumn`. Un ejemplo para utilizar la columna llamada *Name*:


```
Dim sName As String
nName = oForm.findColumn("Name") 'accede a la columna Name
sName = oForm.getString(nName) ' obtiene el contenido (texto) de la columna
```

El tipo de valor devuelto siempre coincide con el tipo de método, pero deben tenerse en cuenta los siguientes casos especiales:

- No existen métodos para los datos de los tipos `Decimal`, `Currency`, etc. que se utilizan para cálculos comerciales exactos. Como Basic realiza automáticamente la conversión adecuada, puede usar `getDouble`.
- Al usar `getBoolean`, debe tener en cuenta que `TRUE` y `FALSE` se definen en la base de datos. Las definiciones habituales (valores lógicos: 1 = TRUE) se procesan correctamente.
- Los valores de fecha se pueden definir no solo con el tipo de datos `Date`, sino también (como se indicó anteriormente) como `util.Date`. Esto facilita la lectura y modificación del año, mes y día.
- Con números enteros, hay que tener cuidado con los diferentes tipos de datos. El ejemplo anterior usa `getLong`; La clave primaria >ID en la variable Basic también debe tener el tipo de datos `Long`, que coincide con el tipo `Integer` en la base de datos.

La lista completa de estos métodos se encuentra en "Edición de filas de datos (registros)" (página 30).



Consejo

Si los valores de un formulario se van a utilizar directamente para su posterior procesamiento en SQL (por ejemplo, para ingresar en otra tabla), es mucho más simple no tener que consultar el tipo de campo.

La siguiente macro, que está vinculada a las propiedades del Botón **Sucesos > Ejecutar una acción**, lee el primer campo en el formulario independientemente del tipo necesario para el procesamiento futuro en Basic.

```
SUB ReadValues(oEvent As Object)
  Dim oForm As Object
  Dim stFeld1 As String
  oForm = oEvent.Source.Model.Parent
  stFeld1 = oForm.getString(1)
End Sub
```

Si los campos se leen usando `getString()`, se conserva todo el formato necesario para el procesamiento posterior de SQL. Una fecha que se muestra como 08.03.19 se lee en el formato 2019-03-08 y se puede usar directamente en SQL.

Leer en un formato correspondiente al tipo de datos solo es obligatorio si el valor se va a procesar posteriormente dentro de la macro, por ejemplo, en un cálculo.

Resultado de una consulta

El conjunto de resultados de una consulta se puede usar de la misma manera. En la Sección de «Órdenes SQL» en la página 15, encontrará la variable `oResult` para este conjunto de resultados, que en general se lee de la siguiente manera:

```
While oResult.next ' bucle para obtener un registro tras otro
  REM transferir el resultado a variables
  stVar = oResult.getString(1)
  inVar = oResult.getLong(2)
```

```

    boVar = oResult.getBoolean(3)
    REM hacer algo con estos valores
Wend ' fin del bucle

```

Según el tipo de orden SQL, el resultado esperado y su propósito, el ciclo `WHILE` se puede acortar o eliminar por completo. Pero básicamente un conjunto de resultados siempre se puede evaluar de esta manera:

Si solo se va a evaluar el primer registro, con

```
oResult.next
```

se accede a la fila para este registro, y con .

```
stVar = oResult.getString(1)
```

se lee el contenido del primer campo. El ciclo termina aquí.

La consulta para el ejemplo anterior tiene texto en la primera columna, un número entero en la segunda (`Integer` en la base de datos corresponde a `Long` en Basic) y un campo booleano (Sí/No) en la tercera. Se accede a los campos a través de un índice de campo que, a diferencia de un índice de matriz, comienza en 1.

Navegar a partir de tal resultado no es posible. Solo se permiten pasos individuales para el siguiente registro. Para navegar dentro del registro, se debe conocer `ResultSetType` cuando se crea la consulta. A esto se accede usando:

```
oSQL_Result.ResultSetType = 1004 o bien SQL_Result.ResultSetType = 1005
```

El Tipo `1004` - `SCROLL_INTENSIVE` permite navegar libremente, pero no detecta cambios en los datos originales. El Tipo `1005` - `SCROLL_SENSITIVE` reconoce los cambios en los datos originales que pueden afectar el resultado de la consulta.

El número total de filas en el conjunto de resultados se puede determinar solo después de que se haya especificado un tipo numérico para el resultado. Se lleva a cabo de la siguiente manera:

```

Dim iResult As Long
If oResult.last ' va al último registro si es posible
    iResult = oResult.getRow ' el número resultante es el total de
    filas
Else
    iResult = 0
End If

```

Uso de un control

Si un control está vinculado a una fuente de datos, el valor se puede leer directamente, como se describe en la siguiente sección. Sin embargo, esto puede conducir a problemas. Es más seguro usar el procedimiento descrito en «Usar formularios» (página 16) o bien el siguiente método, que se muestra para varios tipos diferentes de control:

```

sValue = oTextField.BoundField.Text ' ejemplo para un cuadro de
texto
nValue = oNumericField.BoundField.Value ' ejemplo para un campo
numérico
dValue = oDateField.BoundField.Date ' ejemplo para un campo Fecha

```

`BoundField` representa el enlace entre el control visible y el contenido del conjunto de datos.

Navegar en un conjunto de datos

En el último ejemplo, el método `Next` se utilizó para pasar de una registro del conjunto de resultados al siguiente. Existen otros métodos y pruebas similares que se pueden usar tanto para los datos en un formulario, representado por la variable `oForm`, como para un conjunto de resultados.

Por ejemplo, usando el método descrito en “Actualización automática de formularios” (página 33), el registro anterior se puede volver a seleccionar:

```
Dim loRow As Long
loRow = oForm.getRow() ' guarda el número del registro
oForm.reload() ' recarga el conjunto de registros
oForm.absolute(loRow) ' vuelve al mismo registro
```

La sección “Actualización automática de formularios” muestra los métodos adecuados para ello.



Nota

Desafortunadamente, desde el comienzo de LibreOffice, ha habido un error (transferido desde OpenOffice) que afecta los formularios: establece a "0" el número de registro en uso cuando los datos se alteran dentro de un formulario. Vea: https://bugs.documentfoundation.org/show_bug.cgi?id=82591. Para obtener el número de fila actual correcto, vincule la siguiente macro al las propiedades del Formulario: **Sucesos > Después del cambio de registro.**

```
Global loRow As Long
Sub RowCounter(oEvent As Object)
    loRow = oEvent.Source.Row
End Sub
```

El nuevo número de fila se lee y se asigna a la variable global `loRow`. Esta variable se debe colocar en todos los módulos, al comienzo de todos los procedimientos y retendrá su contenido hasta que salga de Base o cambie el valor llamando a `RowCounter` de nuevo.

Editar registros: agregar, modificar, eliminar

Para editar registros, tienen que cumplirse conjuntamente varias condiciones:

- El usuario tiene que ingresar la información en un control, utilizando el teclado.
- El conjunto de datos contenidos en el formulario tiene que ser informado sobre el cambio. Sucede cuando el foco se mueve de un campo a otro.
- La base de datos en sí tiene que ser modificada. Sucede cuando se mueve un registro a otro.

Cuando se hace a través de una macro, todos estos pasos parciales deben ser rigurosos. Si falta alguno de ellos o se lleva a cabo incorrectamente, los cambios se perderán y no se reflejarán en la base de datos. En primer lugar, el cambio no tiene por qué estar en el valor mostrado del control sino en el conjunto de datos en sí. Esto hace que no tenga sentido cambiar la propiedad `Text` de un control.

Tenga en cuenta que las tablas son los únicos conjuntos de datos que se pueden modificar sin causar problemas. Para otros conjuntos de datos (consultas, por ejemplo), la edición solo es posible bajo circunstancias especiales.

Cambiar el contenido de un control

Si desea cambiar un solo valor, la propiedad `BoundField` del control se puede usar con un método apropiado. Posteriormente, el cambio tiene que transmitirse a la base de datos.

Aquí hay un ejemplo para un campo de fecha en el que se debe ingresar la fecha actual:

```
Dim unoDate As New com.sun.star.util.Date
unoDate.Year = Year(Date)
unoDate.Month = Month(Date)
unoDate.Day = Day(Date)
oDateField.BoundField.updateDate( unoDate )
oForm.updateRow() ' el cambio se transmite a la base de datos
```

Para `BoundField`, se debe utilizar el método `updateXxx` que coincida con el tipo de datos del campo. En este ejemplo, el campo es un campo `Date`. El nuevo valor se pasa como argumento, en este caso la fecha actual, convertida al formato que requiere la macro.

Alterar registros en un archivo de datos

El método anterior no es adecuado cuando es necesario cambiar varios valores en un registro. Por un lado, tendría que existir un control en el formulario para cada campo, lo que a menudo no es útil o no se desea. Además, se tiene que asignar un objeto para cada campo. La forma simple y directa usa un formulario de esta manera:

```
Dim unoDate As New com.sun.star.util.Date
unoDate.Year = Year(Date)
unoDate.Month = Month(Date)
unoDate.Day = Day(Date)
oForm.updateDate(3, unoDate )
oForm.updateString(4, "un Texto")
oForm.updateDouble(6, 3.14)
oForm.updateInt(7, 16)
oForm.updateRow()
```

Para cada columna del conjunto de datos, se llama al método `updateXxx` apropiado para su tipo. Los argumentos son el número de columna (contando desde 1) y el valor deseado. Luego, las modificaciones se tienen que pasar a la base de datos.

Crear, modificar y eliminar registros

Los cambios aquí nombrados se refieren al registro en uso del conjunto de datos subyacente al formulario. En algunos casos, es necesario llamar a un método desde «Navegación en los conjuntos de datos» (página 29). Son necesarios los siguientes pasos:

- 1) Elegir el registro deseado.
- 2) Cambiar los valores como se describe en la sección anterior.
- 3) Confirmar el cambio del registro con la siguiente orden:

```
oForm.updateRow()
```

- 4) En casos especiales, es posible cancelar y volver al estado anterior:

```
oForm.cancelRowUpdates()
```

Para un nuevo registro hay un método especial, comparable con el cambio a una nueva fila en un control de tabla. Se hace de la siguiente manera:

- 1) Preparar para crear un nuevo registro:

```
oForm.moveToInsertRow()
```

- 2) Ingresar todos los valores deseados/necesarios usando los métodos `updateXXX` como se indica en la sección anterior.
- 3) Confirmar la creación de un nuevo registro con los nuevos datos con la siguiente orden:


```
oForm.insertRow()
```
- 4) La nueva entrada no se puede revertir fácilmente. En lugar de ello, tendrá que eliminarse el nuevo registro.

Hay una orden simple para eliminar un registro:

- 1) Elegir el registro deseado y actualizarlo, como se hace para una modificación.
- 2) Usar la siguiente orden para eliminarlo:


```
oForm.deleteRow()
```



Consejo

Para garantizar que los cambios se transfieran a la base de datos, tienen que confirmarse explícitamente con `updateRow` o `insertRow` según corresponda. Mientras se presiona el botón *Guardar*, se usará automáticamente la función apropiada, con una macro tiene que determinar antes de guardar si el registro es nuevo (Insert) o es una modificación de uno existente (Update).

```
If oForm.isNew Then
  oForm.insertRow()
Else
  oForm.updateRow()
End If
```

Prueba y cambio de controles

Además del contenido de un conjunto de datos, se puede leer editar y modificar mucha más información en un control,. Esto es particularmente cierto para las propiedades, como se describe en el «Capítulo 4, Formularios».

En «Mejorar la facilidad de uso» (página 33) encontrará varios ejemplos que usan la información adicional del control:

```
Dim stTag As String
stTag = oEvent.Source.Model.Tag
```

Como se mencionó en la sección anterior, la propiedad `Text` solo se puede modificar de manera útil si el control no está vinculado a un conjunto de datos. Sin embargo, hay otras propiedades que se establecen como parte de la definición del formulario, pero que se pueden adaptar en tiempo de ejecución.

Por ejemplo, una etiqueta podría recibir un color de texto diferente si tiene que representar una advertencia en lugar de una información:

```
Sub showWarning(oField As Object, iType As Integer)
  Select Case iType
    Case 1
      oField.TextColor = RGB(0,0,255) ' 1 = azul
    Case 2
      oField.TextColor = RGB(255,0,0) ' 2 = rojo
    Case Else
      oField.TextColor = RGB(0,255,0) ' 0 = verde (ni 1 ni 2)
  End Select
```

End Sub

Nombres de las propiedades de los controles en macros

Mientras que el diseñador de un formulario puede usar las designaciones en su idioma nativo para las propiedades y acceso a datos, en Basic solo se pueden usar términos en inglés. Se exponen en la siguiente sinopsis.

Las propiedades que normalmente solo se establecen en la definición del formulario no se incluyen aquí. Tampoco los métodos (funciones y/o procedimientos) que rara vez se usan o solo se requieren para declaraciones más complejas.

La sinopsis incluye lo siguiente:

Nombre	Nombre que se utilizará para la propiedad en el código de la macro
Tipo	Tipo de datos de Basic. Para funciones, el tipo de retorno.No incluido para procedimientos
R/W	Indica cómo puede usarse el valor <ul style="list-style-type: none">• R solo lectura• W > solo escritura (modificar)• (R) Lectura posible, inadecuada para procesamiento posterior• (W) Escritura posible pero no útil• R+W Adecuado para lectura y escritura

Se puede encontrar más información en la «Referencia de la interfaz de programación (API)» buscando el nombre del control en inglés. También puede utilizar una herramienta bastante útil llamada *XrayTool* que muestra las propiedades y métodos disponibles para un objeto. Puede obtenerla en el enlace externo: <https://berma.pagesperso-orange.fr/index2.html>

```
Sub Main(oEvent)
  Xray(oEvent)
End Sub
```

Con este procedimiento se inicia la macro *XrayTool* para el argumento (objeto que se quiera inspeccionar, en este caso *oEvent*) .

Propiedades de formularios y controles.

El modelo de un control describe sus propiedades. Según la situación, se puede acceder al valor de una propiedad de solo lectura o de solo escritura. El orden sigue al de las listas de «Propiedades de los campos de control» en el «Capítulo 4, Formularios».

Además de las propiedades genéricas que se indican al principio, se enumeran las propiedades específicas para cada tipo de control.

Fuente (Font)

En cada control que muestra texto, las propiedades de fuente se pueden personalizar.

Nombre	Tipo	R/W	Propiedad
FontName	string	R+W	Nombre de la fuente
FontHeight	single	R+W	Tamaño de la fuente
FontWeight	single	R+W	Negrita o normal
FontSlant	integer	R+W	Itálica o cursiva
FontUnderline	integer	R+W	Subrayada
FontStrikeout	integer	R+W	Tachada

Formulario (Form)

Nombre	Tipo	R/W	Propiedad
ApplyFilter	boolean	R+W	Filtro aplicado
Filter	string	R+W	Filtro para el registro
FetchSize	long	R+W	Número de registros cargados a la vez
Row	long	R	Número de fila (registro) en uso
RowCount	long	R	Número de filas (registros)

Propiedades que se aplican a todos los controles (Control)

Consulte también *FormComponent*

Nombre	Tipo	R/W	Propiedad
Name	string	R+(W)	Nombre del control
Enabled	boolean	R+W	Activo (el control se puede seleccionar)
EnableVisible	boolean	R+W	Mostrar el control
ReadOnly	boolean	R+W	El contenido del control no se puede cambiar
TabStop	boolean	R+W	Se puede acceder al control con la tecla Tab
Align	integer	R+W	Alineación horizontal: 0 = izquierda, 1 = centrado, 2 = derecha
BackgroundColor	long	R+W	Color de fondo
Tag	string	R+W	Información Adicional
HelpText	string	R+W	Texto de la descripción emergente

Propiedades para muchos tipos de controles

Nombre	Tipo	R/W	Propiedad
Text	string	(R+W)	Contenido visualizado en el control. En los controles de texto, se puede leer y procesar, pero no suele funcionar para otros controles.
Spin	boolean	R+W	Botón de incremento o para desplegar el contenido en un campo formateado.
TextColor	long	R+W	Color del texto (primer plano).
DataField	string	R	Nombre del campo en el conjunto de datos.
BoundField	object	R	Conexión con el conjunto de datos (campo enlazado), proporciona acceso al contenido del campo.

Cuadro de texto (TextField)

Nombre	Tipo	R/W	Propiedad
String	string	R+W	Contenido visual del control.
MaxTextLen	integer	R+W	Longitud máxima del texto.

Nombre	Tipo	R/W	Propiedad
DefaultText	string	R+W	Texto predeterminado.
MultiLine	boolean	R+W	Indica si hay más de una línea.
EchoChar	(integer)	R+W	Carácter mostrado durante la entrada de contraseña.

Campo numérico (NumericField)

Nombre	Tipo	R/W	Propiedad
ValueMin	double	R+W	Valor mínimo aceptable
ValueMax	double	R+W	Valor máximo aceptable
Value	double	R+(W)	Valor actual (No lo use para valores del conjunto de datos).
ValueStep	double	R+W	Intervalo de un clic para la rueda del ratón o botón de incremento /decremento del control
DefaultValue	double	R+W	Valor predeterminado.
DecimalAccuracy	integer	R+W	Número de lugares decimales.
ShowThousandsSeparator	boolean	R+W	Permite mostrar el separador de miles (se usa el establecido en la configuración regional).

Campo de moneda (CurrencyField)

Un Campo de moneda es un Campo numérico con las siguientes posibilidades adicionales.

Nombre	Tipo	R/W	Propiedad
CurrencySymbol	string	R+W	Símbolo de moneda solo para visualización.
PrependCurrencySymbol	boolean	R+W	Muestra el símbolo de moneda antes de la cantidad.

Campo de fecha (DateField)

Los valores de fecha están definidos por el tipo de datos long y se muestran en formato ISO: AAAAMMDD, por ejemplo 20190304 para el 04 de marzo de 2019. Para usar este tipo con getDate y updateDate, y con el tipo com.sun.star.util.Date, consulte los ejemplos.

Nombre	Tipo	TipoTipo de datos desde LO 4.1.1	R/W	Propiedad
DateMin	long	com.sun.star.util.Date	R+W	Valor de entrada mínimo aceptable
DateMax	long	com.sun.star.util.Date	R+W	Valor de entrada máximo aceptable
Date	long	com.sun.star.util.Date	R+(W)	Fecha

Nombre	Tipo	TipoTipo de datos desde LO 4.1.1	R/W	Propiedad
DateFormat	integer		R+W	Formato de fecha específico del sistema operativo: 0 = fecha corta (sencilla) 1 = fecha corta dd.mm.aa (año de 2 dígitos) 2 = fecha corta d.mm.aaaa (año de 4 dígitos) 3 = Fecha larga (con día de la semana y el nombre del mes) Se pueden encontrar más posibilidades en la definición del formulario o en la referencia de la API.
DefaultDate	long	com.sun.star.util.Date	R+W	Valor predeterminado.
DropDown	boolean		R+W	Muestra un calendario mensual desplegable.

Campo de hora (TimeField)

Los valores de hora también son de tipo Long.

Nombre	Tipo	Tipo desde LO 4.1.1	R/W	Propiedad
TimeMin	long	com.sun.star.util.Time	R+W	Valor de entrada mínimo aceptable
TimeMax	long	com.sun.star.util.Time	R+W	Valor de entrada máximo aceptable
Time	long	com.sun.star.util.Time	R+(W)	Valor actual (No lo use para valores del conjunto de datos).
TimeFormat	integer		R+W	Formato de hora: 0 = corto hh: mm (horas, minutos, reloj de 24 horas) 1 = largo hh: mm: ss (reloj de 24 horas) 2 = corto >hh: mm (reloj de 12 horas con AM / PM) 3 = largo hh: mm: ss (reloj de 12 horas con AM / PM) 4 = entrada corta durante un tiempo 5 = entrada larga durante un tiempo
DefaultTime	long	com.sun.star.util.Time	R+W	Valor predeterminado.

Campo formateado (FormattedControl)

Un control de *campo formateado* se puede usar como se desee para números, moneda o fecha / hora. Muchas de las propiedades ya descritas se aplican aquí pero con nombres diferentes.

Nombre	Tipo	R/W	Propiedad
CurrentValue	variant	R	Valor actual del contenido. El tipo de datos real depende del contenido y el formato.
EffectiveValue		R+(W)	
EffectiveMin	double	R+W	Valor de entrada mínimo aceptable
EffectiveMax	double	R+W	Valor de entrada máximo aceptable
EffectiveDefault	variant	R+W	Valor predeterminado.
FormatKey	long	R+(W)	Formato para visualización y entrada. No hay una manera fácil de alterarlo usando una macro.
EnforceFormat	boolean	R+W	El formato se comprueba durante la entrada. Solo se permiten ciertos caracteres y combinaciones.

Listado (ListBox)

El acceso de lectura y escritura al valor relativo a la línea seleccionada, es algo complicado pero posible.

Nombre	Tipo	R/W	Propiedad
ListSource	array of string	R+W	Fuente de datos: fuente de los contenidos del listado o nombre del conjunto de datos.
ListSourceType	integer	R+W	Tipo de fuente de datos: 0 = Lista de valores. 1 = tabla. 2 = consulta. 3 = Conjunto de resultados de una orden SQL. 4 = Resultado de un comando de base de datos. 5 = Nombres de los campos de una tabla.
StringItemList	array of string	R	Lista de opciones disponibles para la selección.
ItemCount	integer	R	Número de opciones disponibles en la lista
ValueItemList	array of string	R	Lista de valores que se pasarán del formulario a la tabla.
DropDown	boolean	R+W	Lista desplegable.
LineCount	integer	R+W	Total de líneas mostradas cuando está totalmente desplegado.
MultiSelection	boolean	R+W	Selección múltiple.
SelectedItems	array of integer	R+W	Lista de opciones seleccionadas como una lista de posiciones en la lista general de entradas.

El primer elemento seleccionado del control de listado se obtiene así:

```
oControl = oForm.getByName("Nombre_del_listado")
sEintrag = oControl.ValueItemList( oControl.SelectedItems(0) )
```



Nota

Desde LibreOffice 4.1, el valor pasado a la base de datos se puede determinar directamente.

```
oControl = oForm.getByname("Nombre_del_Listado")
iD = oControl.getCurrentValue() getCurrentValue()
```

devuelve el valor que se almacenará en la tabla de la base de datos.

En los listados, depende del campo al que están vinculados (**BoundField**). Hasta LibreOffice 4.0 inclusive, esta función devolvía el contenido mostrado, no el valor subyacente en la tabla.

Tenga en cuenta que la entrada es una "matriz de cadenas de texto", la consulta de un control de listado debería cambiarse para restringir una opción de selección:

```
Sub Listenfeldfilter
  Dim stSql(0) As String
  Dim oDoc As Object
  Dim oDrawpage As Object
  Dim oForm As Object
  Dim oFeld As Object
  oDoc = thisComponent
  oDrawpage = oDoc.drawpage
  oForm = oDrawpage.forms.getByname("MainForm")
  oFeld = oForm.getByname("Nombre_del_Listado")
  stSql(0) = "SELECT ""Name"", ""ID"" FROM ""Filter_Name"" ORDER BY
""Name"""
  oFeld.ListSource = stSql
  oFeld.refresh
End Sub
```

Cuadro combinado (ComboBox)

A pesar de tener una funcionalidad similar a los listados, las propiedades de los cuadros combinados son algo diferentes. Vea el ejemplo «Cuadros combinados, como listados con opción de entrada» en la página 52.

Nombre	Tipo	R/W	Propiedad
Autocomplete	boolean	R+W	Rellenar automáticamente.
StringItemList	array of string	R+W	Lista de opciones disponibles para su uso.
ItemCount	integer	R	Número de opciones de lista disponibles.
DropDown	boolean	R+W	Lista desplegable.
LineCount	integer	R+W	Número de filas que se muestran al desplegarse.
Text	string	R+W	Texto visualizado actualmente.
DefaultText	string	R+W	Opción predeterminada
ListSource	string	R+W	Nombre de la fuente de datos que proporciona las opciones de la lista.

Nombre	Tipo	R/W	Propiedad
ListSourceType	integer	R+W	Tipo de fuente de datos. Las mismas que para los listados (solo se ignora la elección de la lista de valores).

Casillas de verificación (CheckBox) y Botón de opción (RadioButton)

Nombre	Tipo	R/W	Propiedad
Label	string	R+W	Título (etiqueta)
State	short	R+W	Estado 0 = no seleccionado 1 = seleccionado 2 = indefinido
MultiLine	boolean	R+W	Indica si hay más de una línea.

Campo enmascarado (PatternField)

Además de las propiedades de cuadro de texto tiene >las siguientes:

Nombre	Tipo	R/W	Propiedad
EditMask	string	R+W	Máscara de entrada.
LiteralMask	string	R+W	Máscara de caracteres
StrictFormat	boolean	R+W	Comprobación de formato durante la entrada.

Control de Tablas (GridControl)

Nombre	Tipo	R/W	Propiedad
Count	long	R	Número de columnas.
ElementNames	array of string	R	Lista de nombres de columna.
HasNavigation Bar	boolean	R+W	Barra de navegación.
RowHeight	long	R+W	Altura de la fila.

Etiqueta (Label)

Nombre	Tipo	R/W	Propiedad
Label	string	R+W	Texto mostrado.
MultiLine	boolean	R+W	Indica si hay más de una línea.

Cuadro de Grupo (GroupBox)

No hay propiedades para los cuadros de grupo que normalmente se procesen con macros. Lo que realmente importa es el estado de los *campos de opción* individuales.

Botones (Button)

Botón o Botón con imagen

Nombre	Tipo	R/W	Propiedad
Label	string	R+W	Título (texto de la etiqueta).

Nombre	Tipo	R/W	Propiedad
State	short	R+W	Estado predeterminado para una selección alternativa.
MultiLine	boolean	R+W	Indica si hay más de una línea.
DefaultButton	boolean	R+W	Si ha de ser el botón predeterminado.

Barra de navegación (NavigationBar)

Las propiedades y métodos adicionales asociados con la navegación, por ejemplo, filtros y cambio del puntero de registro, se controlan mediante el formulario.

Nombre	Tipo	R/W	Propiedad
IconSize	short	R+W	Tamaño de los iconos.
ShowPosition	boolean	R+W	Si la posición se puede ingresar y mostrar.
ShowNavigation	boolean	R+W	Si se permite la navegación.
ShowRecordActions	boolean	R+W	Si se permite grabar acciones.
ShowFilterSort	boolean	R+W	Si se permite la ordenación por filtros.

Métodos para formularios y controles

El tipo de datos del parámetro se indica mediante una abreviatura:

- número de columna para el campo deseado en el archivo de datos, contando desde 1
- valor numérico: podría ser un número entero o decimal
- s String (cadena de texto); La longitud máxima depende de la definición de la tabla.
- b booleano (lógico): verdadero o falso
- d Valor de fecha

Navegación en los conjuntos de datos

Estos métodos funcionan tanto en formularios como en el conjunto de resultados de una consulta.

Cursor en la descripción significa el puntero de registro (|).

Nombre	Tipo	Descripción
Probar la posición del cursor.		
isBeforeFirst	boolean	El cursor está antes del primer registro. Este es el caso si aún no se ha restablecido después de la entrada.
isFirst	boolean	Muestra si el cursor está en la primera entrada.
isLast	boolean	Muestra si el cursor está en la última entrada.
isAfterLast	boolean	El cursor está después de la última fila cuando se mueve hacia la siguiente.
getRow	long	Número de fila en que está el cursor.
Mover el cursor		
Para los tipos de datos booleanos, True significa que la navegación se realizó.		
beforeFirst	–	Se mueve antes de la primera fila.
first	boolean	Se mueve a la primera fila.

Nombre	Tipo	Descripción
previous	boolean	Retrocede una fila.
next	boolean	Avanza una fila.
last	boolean	Va al último registro.
afterLast	–	Va después del último registro.
absolute(n)	boolean	Va a la fila con número especificado.
relative(n)	boolean	Avanza o retrocede la cantidad específica. Hacia adelante para argumentos positivos y hacia atrás para argumentos negativos.
Métodos que afectan el estado actual del registro		
refreshRow	–	Lee de nuevo los valores originales de la fila.
rowInserted	boolean	Indica si es una fila nueva.
rowUpdated	boolean	Indica si la fila actual ha sido alterada.
rowDeleted	boolean	Indica si la fila actual se ha eliminado.

Edición de filas de datos (registros)

Los métodos utilizados para la lectura están disponibles para cualquier formulario o para un conjunto de resultados. Los métodos de alteración y almacenamiento solo se pueden usar para conjuntos de datos editables (generalmente tablas, no consultas).

Nombre	Tipo	Descripción
Métodos para toda la fila		
insertRow	–	Inserta un registro nuevo.
updateRow	–	Actualiza la alteración del registro en curso.
deleteRow	–	Elimina el registro indicado.
cancelRowUpdates	–	Deshace los cambios en el registro correspondiente.
moveToInsertRow	–	Mueve el cursor al registro que se ha insertado.
moveToCurrentRow	–	Tras insertar un nuevo registro, vuelve cursor a su posición anterior.
Valores de lectura		
getString(c)	string	Obtiene el contenido de la columna como una cadena de caracteres.
getBoolean(c)	boolean	Obtiene el contenido de la columna como un valor booleano.
getBytes(c)	byte	Obtiene el contenido de la columna como un solo byte.
getShort(c) getInt(c) getLong(c)	Short integer long	Obtiene el contenido de la columna como un entero.

Nombre	Tipo	Descripción
getFloat(c)	float	Obtiene el contenido de la columna como un único valor de precisión decimal.
getDouble(c)	double	Obtiene el contenido de la columna como un número decimal de doble precisión. Las conversiones automáticas hechas por Basic hacen de este un tipo adecuado para campos decimales y de moneda.
getBytes(c)	array of bytes	Obtiene el contenido de la columna como una matriz de bytes simples.
getDate(c)	Date	Obtiene el contenido de la columna como una fecha.
getTime(c)	Time	Obtiene el contenido de la columna como un valor de hora.
getTimestamp(c)	DateTime	Obtiene el contenido de la columna como marca de tiempo (fecha y hora).
<p>En Basic, los valores de fecha y hora reciben el tipo DATE. Para acceder a las fechas en los conjuntos de datos, hay varios tipos: <code>com.sun.star.util.Date</code> para una fecha, <code>com.sun.star.util.Time</code> para unidades horarias y <code>com.sun.star.util.DateTime</code> para una marca de tiempo (fecha y hora).</p>		
wasNull	boolean	Indica si el valor más reciente leído en una columna no tiene datos (NULL).
Almacenar valores		
updateNull(c)	–	Establece el contenido de la columna c como vacío (NULL).
updateBoolean(c,b)	–	Almacena el contenido booleano (b) en la columna c.
updateByte(c,x)	–	Almacena el byte (x) en la columna c.
updateShort(c,n) updateInt(c,n) updateLong(c,n)	–	Almacena el número entero (n) en la columna c.
updateFloat(c,n) updateDouble(c,n)	–	Almacena el número decimal (n) en la columna c.
updateString(c,s)	–	Almacena la cadena de texto (s) en la columna c.
updateBytes(c,x)	–	Almacena la matriz de bytes (x) en la columna c.
updateDate(c,d)	–	Almacena la fecha (d) en la columna c.
updateTime(c,d)	–	Almacena la hora (d) en la columna c.
updateTimestamp(c,d)	–	Almacena la marca de tiempo (d) en la columna c.

Edición individual de valores

Con estos métodos, el contenido de la columna relevante se lee o se cambia desde un campo de control mediante su enlace con un campo (`BoundField`). Es prácticamente el mismo que el método descrito en la sección anterior, excepto que no se utiliza el número de columna.

Nombre	Tipo	Descripción
Valores de lectura		
getString	string	Obtiene el contenido del campo como una cadena de texto.
getBoolean	boolean	Obtiene el contenido del campo como un valor lógico.
getBytes	byte	Obtiene el contenido del campo como un solo byte.
GetShort getInt getLong	Short integer long	Obtiene el contenido del campo como un entero.
getFloat	float	Obtiene el contenido como un valor decimal de precisión simple.
getDouble	double	Obtiene el contenido del campo como un número decimal de doble precisión. Las conversiones automáticas realizadas por Basic hacen de este un tipo adecuado para campos decimales y de moneda.
getBytes	array of bytes	Obtiene el contenido del campo como una matriz de bytes.
getDate	Date	Obtiene el contenido del campo como una fecha.
getTime	Time	Obtiene el contenido del campo como una hora.
getTimestamp	DateTime	Obtiene el contenido del campo como una marca de tiempo.
<p>En Basic, los valores de fecha y hora reciben el tipo DATE. Para acceder a las fechas, en los conjuntos de datos, hay varios tipos: <code>com.sun.star.util.Date</code> para una fecha, <code>com.sun.star.util.Time</code> para una expresión horaria y <code>com.sun.star.util.DateTime</code> para una marca de tiempo.</p>		
WasNull	boolean	Indica si el valor más leído en una columna no tiene datos (NULL).
Almacenar valores		
updateNull	–	Establece el contenido de la columna en vacío (NULL).
updateBoolean(b)	–	Establece el contenido de la columna en el valor lógico 'b'.
updateByte(x)	–	Almacena el byte 'x' en la columna.
updateShort(n)	–	Almacena el entero 'n' en la columna.
updateInt(n)	–	
updateLong(n)	–	
updateFloat(n) updateDouble(n)	– –	Almacena el número decimal 'n' en la columna.
updateString(s)	–	Almacena la cadena de caracteres 's' en la columna.
updateBytes(x)	–	Almacena la matriz de bytes 'x' en la columna.
updateDate(d)	–	Almacena la fecha 'd' en la columna.

Nombre	Tipo	Descripción
updateTime(d)	–	Almacena la hora 'd' en la columna.
updateTimestamp(d)	–	Almacena la marca de tiempo 'd' en la columna.

Parámetros para órdenes SQL preparadas

Los métodos que transfieren el valor de una orden SQL preparado previamente (consulte «Órdenes SQL previamente preparadas con parámetros» en la página 15) son similares a los de la sección anterior.

El primer parámetro (indicado por i) es una posición numerada dentro de la orden SQL.

Nombre	Tipo	Descripción
setNull(i, n)	–	Establece el contenido de la columna 'i' en NULL. 'n' es el tipo de datos SQL proporcionado en la Referencia de API.
setBoolean(i, b)	–	Pone el valor lógico 'b' en la orden SQL.
setByte(i, x)	–	Pone el byte 'x' en la orden SQL.
setShort(i, n)	–	Pone el entero 'n' en la orden SQL.
setInt(i, n)	–	
setLong(i, n)	–	
setFloat(i, n) setDouble(i, n)	–	Pone el número decimal 'n' en la orden SQL.
setString(i, s)	–	Pone la cadena de caracteres 's' en la orden SQL.
setBytes(i, x)	–	Pone el conjunto de bytes 'x' en la orden SQL.
setDate(i, d)	–	Pone la fecha 'd' en la orden SQL.
setTime(i, d)	–	Pone la hora 'd' en la orden SQL.
setTimestamp(i, d)	–	Pone la marca de tiempo 'd' en la orden SQL.
clearParameters	–	Elimina valores anteriores de todos los parámetros en la orden SQL.

Mejorar la facilidad de uso

Como primera categoría, se presentan varias opciones que sirven para mejorar la facilidad de uso de los formularios de Base.

Actualización automática de formularios

A menudo, se modifica algo en un formulario y se necesita que esta modificación aparezca en un segundo formulario de la misma página. El siguiente fragmento de código llama al método de actualización del segundo formulario.

Sub Update

Primero se nombra el procedimiento. La designación predeterminada para un procedimiento es Sub. Puede escribirse en mayúsculas o minúsculas. Sub incia un procedimiento que normalmente no devuelve ningún valor. Más abajo, se describe una función que puede devolver un valor para su uso posterior.

La macro tiene el nombre `Update`. No necesita declarar variables porque LibreOffice Basic declara automáticamente variables cuando se usan. Si escribe mal una variable, LibreOffice Basic crea silenciosamente una nueva variable sin quejarse.

Utilice `Option Explicit` para evitar que LibreOffice Basic declare automáticamente variables; (Recomendado por la mayoría de los programadores).

Generalmente se empieza declarando las variables. Todas las variables declaradas aquí son objetos (no números ni texto), por lo que agregamos `As Object` al final de la declaración. Para recordarnos más tarde el tipo de variables, anteponeamos a sus nombres una "o". Sin embargo, en principio, puede elegir cualquier nombre de variable que desee.

```
Dim oDoc As Object
Dim oDrawpage As Object
Dim oForm As Object
```

El formulario se encuentra en el documento actualmente activo. El contenedor, en el que se almacenan todos los formularios, se denomina `drawpage`. En el *Navegador de formularios*, este es el concepto de nivel superior, del que todos los formularios son subsidiarios.

En este ejemplo, el formulario se accede al formulario `Display`. Este es el nombre que se ha dado al formulario y se verá en el navegador de formularios. Para el primer formulario, si no se indica un nombre, Base le asigna el nombre predeterminado: `Form1`.

```
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oForm = oDrawpage.forms.getByName("Display")
```

Dado que el formulario se ha hecho accesible y el punto en el que se puede acceder se guarda en la variable `oForm`, ahora se vuelve a cargar (actualizar) con la instrucción `reload()`.

```
oForm.reload()
End Sub
```

Al ser un procedimiento (comienza con `SUB`, debe terminar con `End Sub`).

Se puede seleccionar este procedimiento para que se ejecute cuando se guarda un formulario.

Por ejemplo, en una caja registradora, si el número total de artículos vendidos y sus números de identificación en un inventario (leídos por un escáner de código de barras) se ingresan en un formulario, otro formulario en la misma ventana abierta (`drawpage`) puede mostrar los nombres de todos los artículos, y el costo total, cuando se actualice el formulario.

Filtrar registros

Un filtro puede funcionar perfectamente en la forma descrita en el «Capítulo 8, Tareas de base de datos». La variante que se muestra a continuación se encuentra en la base de datos de ejemplo `Example_Search_and_Filter.odt` reemplaza al botón `Guardar` utilizado en el formulario `filter_without_macros` (de este archivo) y vuelve a leer los controles de listado, de modo que un filtro elegido para un control de listado puede restringir las opciones disponibles en el otro control de listado.

```
Sub Filter
Dim oDoc As Object
Dim oDrawpage As Object
Dim oForm1 As Object
Dim oForm2 As Object
Dim oFieldList1 As Object
Dim oFieldList2 As Object
```

```
oDoc = thisComponent
oDrawpage = oDoc.drawpage
```

Primero, las variables se definen y configuran para acceder al conjunto de formularios. Este conjunto comprende los dos formularios *filter* y *display*. Los controles de listado están en el formulario *filter* y tienen los nombres *listbox1* y *listbox2*.

```
oForm1 = oDrawpage.forms.getByName("filter")
oForm2 = oDrawpage.forms.getByName("display")
oFieldList1 = oForm1.getByName("listbox1")
oFieldList2 = oForm1.getByName("listbox2")
```

Primero se transfiere el contenido de los Listados al formulario subyacente mediante `commit()`. La transferencia es necesaria, porque de lo contrario el cambio en un listado no se tendrá en cuenta cuando se actualice. La instrucción `commit()` solo debe aplicarse al listado al que se acaba de activar. Después de eso, el registro se guarda usando `updateRow()`.

En principio, nuestra tabla de filtros contiene un único registro, que se escribe una vez al principio. Este registro se sobrescribe continuamente con una instrucción de actualización.

```
oFieldList1.commit()
oFieldList2.commit()
oForm1.updateRow()
```

Los Listados están destinados a influenciarse entre sí. Por ejemplo, si se usa un listado para restringir los artículos visualizados a *CD*, el otro listado no debe incluir a los escritores de libros en su lista de autores. Una selección demasiado frecuente en el segundo listado daría como resultado un filtro vacío. Es por eso que los Listados deben leerse nuevamente. En sentido estricto, la instrucción `refresh()` solo debe aplicarse en el listado al que no se ha accedido.

Después de esto, se lee nuevamente *form2*, que debería mostrar el contenido filtrado.

```
oFieldList1.refresh()
oFieldList2.refresh()
oForm2.reload()
End Sub
```

Mediante varias consultas se puede suministrar el contenido a los listados (influenciados con este método).

La variante más sencilla es hacer que el listado obtenga su contenido de los resultados de un filtro. Posteriormente, otro filtro determinará qué datos serán nuevamente filtrados.

```
SELECT "Field_1" || ' - ' || "Count" AS "Display", "Field_1"
FROM ( SELECT COUNT( "ID" ) AS "Count", "Field_1" FROM "searchtable" GROUP BY
"Field_1" )
ORDER BY "Field1"
```

Se muestra el contenido del campo y el número de resultados. Para obtener el número de resultados, se utiliza una subconsulta. Esto es necesario, ya que de lo contrario solo se mostrará el número de resultados, sin más información que el campo relacionado, en el listado.

Mediante este procedimiento, la macro crea rápidamente listados que solo se llenan con un valor. Si un listado no está vacío (NULL), se tiene en cuenta durante el filtrado. Después de activar el segundo listado, solo están disponibles los campos vacíos y un valor mostrado para ambos listados.

Eso puede parecer práctico para una búsqueda limitada. Pero, ¿qué sucede si el catálogo de una biblioteca muestra claramente la ordenación de un elemento, pero no estuviera claro si se trata de un libro, un CD o un DVD?

Una vez seleccionado el sistema de ordenación en el primer listado, si en el segundo listado se selecciona *CD*, debe restablecerse a *NULL* para realizar una búsqueda posterior que incluya libros. Sería más práctico si el segundo Listado mostrara directamente los diversos tipos de artículos disponibles, con el cómputo de resultados correspondiente.

Para lograr este objetivo, se construye la siguiente consulta, que ya no se alimenta directamente de los resultados del filtro. El número de resultados debe obtenerse de una manera diferente.

```
SELECT
IFNULL( "Field_1" || ' - ' || "Count", 'empty - ' || "Count" ) AS "Display",
"Field_1"
FROM
( SELECT COUNT( "ID" ) AS "Count", "Field_1" FROM "Table"
  WHERE "ID" IN
    ( SELECT "Table"."ID" FROM "Filter", "Table"
      WHERE "Table"."Field_2" = IFNULL( "Filter"."Filter_2",
        "Table"."Field_2" ) )
  GROUP BY "Field_1" )
ORDER BY "Field_1"
```

Esta consulta tan compleja puede desglosarse. En la práctica, es común usar una consulta en modo vista para la subconsulta. El listado recibe su contenido de una consulta relacionada con esta consulta(vista) .

La consulta en detalle: la consulta presenta dos columnas. La primera columna contiene la vista visualizada por una persona que tiene el formulario abierto. Esta vista muestra el contenido del campo y, separados por un guion, la cantidad de resultados de este campo. La segunda columna transfiere su contenido a la tabla subyacente del formulario. Aquí solo tenemos el contenido del campo. Los listados extraen su contenido de la consulta, que se presenta como el resultado del filtro en el formulario. Solo estos campos están disponibles para su posterior filtrado.

La tabla de la que se extrae esta información es en realidad una consulta. En esta consulta, se cuentan los campos de la clave primaria (`SELECT COUNT("ID") AS "Count"`). Se agrupa por el término de búsqueda en el campo (`GROUP BY "Field_1"`). Esta consulta presenta el término en el campo mismo como la segunda columna y a su vez se basa en una subconsulta adicional:

```
SELECT "Table"."ID" FROM "Filter", "Table"
WHERE "Table"."Field_2" =
  IFNULL( "Filter"."Filter_2", "Table"."Field_2" )
```

Esta subconsulta trata con el otro campo a filtrar. En principio, este campo también debe coincidir con la clave primaria. Si se utilizan más filtros, esta consulta se puede extender:

```
SELECT "Table"."ID" FROM "Filter", "Table" WHERE
"Table"."Field_2" = IFNULL( "Filter"."Filter_2", "Table"."Field_2" )
AND
"Table"."Field_3" = IFNULL( "Filter"."Filter_3", "Table"."Field_3" )
```

Esto permite que cualquier otro campo que se filtre controle lo que finalmente aparece en el listado del primer campo, *Field_1*.

Finalmente, toda la consulta se ordena por el campo subyacente.

En el «Capítulo 8, Tareas de base de datos», se puede ver cómo se muestra realmente la consulta final subyacente al formulario mostrado.

La siguiente macro puede controlar mediante un listado qué listados deben guardarse y cuáles deben leerse nuevamente.

El siguiente procedimiento asume que la propiedad de *Información adicional* de cada listado contiene una lista separada por comas de todos los nombres de listado sin espacios. El primer nombre en la lista debe ser el nombre de ese listado.

```
Sub Filter_more_info(oEvent As Object)
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oForm1 As Object
    Dim oForm2 As Object
    Dim sTag As String
    sTag = oEvent.Source.Model.Tag
```

Se establece una matriz (una colección de datos accesibles a través de un número de índice) y se llena con los nombres de campo de los listados. El primer nombre en la lista es el nombre del listado vinculado al suceso.

```
aList() = Split(sTag, ",")
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oForm1 = oDrawpage.forms.getByName("filter")
oForm2 = oDrawpage.forms.getByName("display")
```

La matriz se ejecuta desde su límite inferior (Lbound()) hasta su límite superior (Ubound()) en un solo bucle. Todos los valores que estaban separados por comas en la información adicional, ahora se transfieren sucesivamente.

```
For i = LBound(aList()) To UBound(aList())
    If i = 0 Then
```

El listado que activó la macro debe guardarse. Se encuentra en la variable `aList(0)`. Primero, la información del listado se transfiere a la tabla subyacente y luego se guarda el registro.

```
        oForm1.getByName(aList(i)).commit()
        oForm1.updateRow()
    Else
```

Los otros listados tienen que actualizarse, ya que ahora contienen valores diferentes dependientes del primer listado.

```
        oForm1.getByName(aList(i)).refresh()
    End If
Next
oForm2.reload()
End Sub
```

Las consultas para esta macro más operativa (mayor facilidad de uso) son naturalmente las mismas que las presentadas en la sección anterior.

Preparar datos para campos de texto que se ajusten a convenciones SQL

Cuando los datos se almacenan en una orden SQL, los apóstrofes en nombres como "O'Connor" pueden causar problemas. Esto se debe a que las comillas simples (' ') se usan para encerrar el texto que se ingresará en los registros. En estos casos, necesitamos una función intermedia para preparar los datos adecuadamente.

```
Function String_to_SQL(st As StringString)
```

```

If InStr(st,"'") Then
    st = Join(Split(st,"'"),"''")
End If
String_to_SQL = st
End Function

```

Tenga en cuenta que esta es una función, no un procedimiento. Una función puede tomar un valor como argumento y luego devolver un valor procesado.

Primero se busca el texto a transferir para ver si contiene un apóstrofo. Si este es el caso, el texto se divide en este punto, el apóstrofo en sí mismo es el delimitador de la división, y se une nuevamente con dos apóstrofes. Esto enmascara el código SQL. La función produce su resultado a través de la siguiente llamada:

```
stTextnew = String_to_SQL(stTextold)
```

Esto significa simplemente que la variable `stTextold` se vuelve a procesar y el resultado se almacena en `stTextnew`. Las dos variables en realidad no necesitan tener nombres diferentes. La llamada se puede hacer con:

```
stText = String_to_SQL(stText)
```

Esta función se usa repetidamente en las siguientes macros para que los apóstrofes también se puedan almacenar utilizando órdenes SQL.

Calcular valores anticipadamente

Normalmente, los valores que se pueden calcular utilizando funciones de la base de datos no se almacenan en la base de datos. El cálculo no se lleva a cabo durante la entrada de datos en el formulario, sino al guardar el registro.

Si el formulario consta de un control de tabla, es un poco diferente. El valor calculado se puede leer inmediatamente después de la entrada de datos. Pero cuando los formularios tienen un conjunto de campos individuales, el cálculo correspondiente puede no ser visible.

En estos supuestos, es lógico que los valores que se calculan mediante fórmulas en la base de datos se reflejen en los controles apropiados del formulario. Los siguientes procedimientos que aparecen en la base de datos de ejemplo `Example_direct_Calculation_Form.odt` muestran cómo se pueden hacer visibles estos cálculos.

El primer procedimiento está vinculado a la pérdida del foco del campo *price*.

```

Sub Calculation_without_Tax(oEvent As Object)
    Dim oForm As Object
    Dim oField As Object
    Dim oField2 As Object
    oField = oEvent.Source.Model
    oForm = oField.Parent
    oField2 = oForm.GetByName("price_without_tax")
    oField2.BoundField.UpdateDouble(oField.GetCurrentValue / 1.19)
    If Not IsEmpty(oForm.GetByName("quantity").GetCurrentValue()) Then
        total_calc2(oForm.GetByName("quantity"))
    End If
End Sub

```

Si se ingresa un valor en el control *price*, la macro se inicia al salir de ese campo. En el mismo formulario, el control *price_without_tax* se actualiza mediante `BoundField.UpdateDouble` calculando el precio sin impuestos. El campo de datos se deriva de la consulta (vinculada al

formulario), que, en principio, realiza el mismo cálculo. De esta manera, el valor calculado es visible, durante la entrada de datos (mediante la macro), y también más tarde (con la consulta), durante la navegación a través de los registros almacenados.

Si el control de cantidad (*quantity*) contiene un valor, se debe realizar un cálculo adicional para los campos vinculados *total* y *total_without_tax*.

```
Sub Calculation_Total(oEvent As Object)
    oField = oEvent.Source.Model
    Calculation_Total2(oField)
End Sub
```

Este breve procedimiento sirve para transmitir el valor de la cantidad al siguiente procedimiento, cuando se abandona el control *quantity* del formulario.

```
Sub Calculation_Total2(oFeld As Object)
    Dim oForm As Object
    Dim oField2 As Object
    Dim oField3 As Object
    Dim oField4 As Object
    oForm = oFeld.Parent
    oField2 = oForm.getByName("price")
    oField3 = oForm.getByName("total")
    oField4 = oForm.getByName("tax_total")
    oField3.BoundField.UpdateDouble(oField.getCurrentValue *
oField2.getCurrentValue)
    oField4.BoundField.UpdateDouble(oField.getCurrentValue *
oField2.getCurrentValue -
    oField.getCurrentValue * oField2.getCurrentValue / 1.19)
End Sub
```

Este procedimiento afecta a varios campos a la vez. Se inicia desde el campo *quantity* que contiene la cantidad de productos. Con este campo y el campo de precio, se calcula el precio total y el total con Impuestos y se transfieren a los campos correspondientes.

Este diseño tiene un inconveniente: la tasa de Impuestos está formulada dentro de los procedimientos y la consulta. Sería mejor utilizar un argumento, relacionado con el precio del producto, ya que los impuestos pueden variar o no ser los mismos para todos los productos. El valor apropiado para los Impuestos debería leerse de un control en el formulario y estar reflejado en la tabla correspondiente.

Proporcionar la versión actual de LibreOffice

LibreOffice en su versión 4.1 trajo algunos cambios, en los campos de lista y los valores de fecha, que hacen necesario determinar cuál es la versión que se está usando al ejecutar macros en estas circunstancias. El siguiente código sirve para este propósito:

```
Function OfficeVersion()
    Dim aSettings, aConfigProvider
    Dim aParams2(0) As New com.sun.star.beans.PropertyValue
    Dim sProvider$, sAccess$
    sProvider = "com.sun.star.configuration.ConfigurationProvider"
    sAccess = "com.sun.star.configuration.ConfigurationAccess"
    aConfigProvider = createUnoService(sProvider)
    aParams2(0).Name = "nodepath"
```

```

    aParams2(0).Value = "/org.openoffice.Setup/Product"
    aSettings = aConfigProvider.CreateInstanceWithArguments(sAccess,
aParams2())
    OfficeVersion() =
Array(aSettings.ooName, aSettings.ooSetupVersionAboutBox)
End Function

```

Esta función devuelve una matriz en la que el primer elemento es LibreOffice y el segundo es el número de versión completo, por ejemplo 4.1.5.2.

Obtener el valor devuelto por los listados

Desde LibreOffice 4.1, el valor devuelto por un listado a la base de datos se almacena en CurrentValue. Este no era el caso en versiones anteriores, ni en OpenOffice ni en Apache OpenOffice. La siguiente función determinará como usar esta característica para que sea compatible con las distintas versiones. Para ello verifica la versión de LibreOffice.

```

Function ID_Determination(oField As Object) As Integer
    a() = OfficeVersion()
    If a(0) = "LibreOffice" And (LEFT(a(1),1) = 4 And
RIGHT(LEFT(a(1),3),1) > 0) Or LEFT(a(1),1) > 4 Then
        stContent = oField.CurrentValue
    Else

```

Antes de LibreOffice 4.1, el valor que se pasaba se leía de la lista de valores del listado. El registro visiblemente elegido es SelectedItems (0). Se utilizaba 0 porque podrían seleccionarse varios valores adicionales en un listado.

```

        stContent = oField.ValueItemList(oField.SelectedItems(0))
    End If
    If IsEmpty(stContent) Then

```

-1 es un valor que no se utiliza como Valor Automático y, por lo tanto, no existirá en la mayoría de las tablas como clave foránea.

```

        ID_Determination = -1
    Else
        ID_Determination = Cint(stContent) 'convertir a entero
    End If
End Function

```

La función transmite el valor como un entero. La mayoría de las claves primarias son enteros de incremento automático. Cuando una clave foránea no cumple este criterio, el valor de retorno debe ajustarse al tipo apropiado.

El valor visualizado de un listado se puede determinar adicionalmente utilizando la propiedad de presentación del campo.

```

Sub Listfielddisplay
    Dim oDoc As Object
    Dim oForm As Object
    Dim oListbox As Object
    Dim oController As Object
    Dim oView As Object
    oDoc = thisComponent

```



```

oForm = oDoc.Drawpage.Forms(0)
oListBox = oForm.getByName("ListBox")
oController = oDoc.getCurrentController()
oView = oController.getControl(oListBox)
print "Displayed content: " & oView.SelectedItem
End Sub

```

El controlador *oController* se utiliza para acceder a la vista del formulario. Esto determina lo que aparece en la interfaz visual. El valor seleccionado es `SelectedItem`.

Limitar el contenido de los listados ingresando letras iniciales

A veces el contenido de los listados es demasiado grande para manejarlo. Para agilizar la búsqueda en estos casos, es útil limitar el contenido del listado a los valores indicados al ingresar uno o más caracteres iniciales. El listado en sí se proporciona con una orden SQL que sirve como marcador de posición. Podría ser:

```
SELECT "Name", "ID" FROM "Table" ORDER BY "Name" LIMIT 5
```

Evita que Base tenga que leer una gran lista de valores cuando se abre el formulario.

La siguiente macro está vinculada a las propiedades del listado **Sucesos > Clave liberada**.

```

Global stListStart As String
Global lTime As Long

```

Primero, se crean variables globales. Estas variables son necesarias para permitir la búsqueda no solo de una sola letra, sino también, tras pulsar más teclas, para combinaciones de letras.

Las letras ingresadas se almacenan secuencialmente en la variable global `stListStart`.

La variable global `lTime` se usa para almacenar la hora actual en segundos. Si hay una pausa larga entre las pulsaciones de teclas, la variable `stListStart` debe restablecerse. Por esta razón, se consulta la diferencia horaria entre entradas sucesivas.

```

Sub ListFilter(oEvent As Object)
oField = oEvent.Source.Model
If oEvent.KeyCode < 538 Then

```

La macro se inicia con una pulsación de tecla. Dentro de la interfaz de programación (API), cada tecla tiene un código numérico (`KeyCode`) que se puede buscar en el siguiente enlace de internet: com::sun::star::awt::Key.

Los caracteres especiales como ä, ö y ü tienen el `KeyCode` 0. Todas las demás letras y números tienen un `KeyCode` menor que 538.

Es importante verificar el `KeyCode` porque al presionar la tecla *Tab* para moverse a otro campo también se iniciará la macro. El `KeyCode` para la tecla *Tab* es 1282, por lo que no se ejecutará ningún código adicional en la macro.

```
Dim stSql(0) As String
```

El código SQL para el listado se almacena en una matriz. Sin embargo, las órdenes SQL cuentan como elementos de datos únicos, por lo que la matriz se dimensiona como `stSql(0)`.

Cuando lea el código SQL del listado, tenga en cuenta que el código SQL no es accesible directamente como texto. En cambio, el código está disponible como un único elemento de matriz: `oField.ListSource(0)`.

Después de declarar variables para uso futuro, la orden SQL se divide. Para obtener el campo que se va a filtrar, dividimos el código en la primera coma. Por lo tanto, el campo debe colocarse al

principio de la orden. Luego, este código se divide nuevamente en el primer carácter de comillas dobles, que introduce el nombre del campo. Se hace usando matrices simples. La variable `stField` necesita que las comillas vuelvan a aparecer al principio. Además se usa `Rtrim` para evitar que se produzca un espacio al final de la expresión.

```
Dim stText As String
Dim stField As String
Dim stQuery As String
Dim ar0()
Dim ar1()
ar0() = Split(oField.ListSource(0), ",", 2)
ar1() = Split(ar0(0), "\"", 2)
stField = "\"" & Rtrim(ar1(1))
```

Se espera una instrucción de ordenación a continuación en el código SQL. Sin embargo, las órdenes en SQL pueden estar en mayúsculas, minúsculas o mixtas, por lo que se usa la función `inStr` en lugar de `Split` para encontrar la cadena de caracteres `ORDER`. El último parámetro para esta función es `1`, lo que indica que la búsqueda no distingue entre mayúsculas y minúsculas. Todo a la izquierda de la cadena `ORDER` se debe utilizar para construir el nuevo código SQL. Esto garantiza que el código también pueda servir para listados provenientes de diferentes tablas o que se han definido en el código SQL utilizando condiciones.

```
stQuery = Left(oField.ListSource(0), inStr(1,oField.ListSource(0),
"ORDER",1)-1)
If inStr(stQuery, "LOWER") > 0 Then
    stQuery = Left(stQuery, inStr(stQuery, "LOWER")-1)
ElseIf inStr(1,stQuery, "WHERE",1) > 0 Then
    stQuery = stQuery & " AND "
Else
    stQuery = stQuery & " WHERE "
End If
```

Si la consulta contiene el término `LOWER`, significa que se creó utilizando este procedimiento `ListFilter`. Por lo tanto, al construir la nueva consulta, necesitamos ir solo hasta esta posición.

Si este no es el caso, y la consulta ya contiene el término `WHERE` (en mayúscula o minúscula), cualquier condición adicional a la consulta debe anteponerse con `AND`.

Si no se cumple ninguna de las dos condiciones, se añade un `WHERE` al código existente.

```
If lTime > 0 And Timer() - lTime < 5 Then
    stListStart = stListStart & oEvent.KeyChar
Else
    stListStart = oEvent.KeyChar
End If
lTime = Timer()
```

Si se ha almacenado un valor de hora en la variable global, y la diferencia entre este valor y la hora actual es inferior a 5 segundos, la letra ingresada se une a la anterior. De lo contrario, la letra se trata como una nueva entrada de una sola letra. El control de listado se volverá a filtrar de acuerdo con esta entrada. Después de esto, la hora actual se almacena en `lTime`.

```
stText = LCase( stListStart & "%")
stSql(0) = stQuery + "LOWER("+stField+") LIKE '"+stText+"' ORDER
BY "+stField+""
```

```

        oFeld.ListSource = stSql
        oField.refresh
    End If
End Sub

```

El código SQL finalmente se une. La versión en minúsculas del contenido del campo se compara con la versión en minúsculas de las letras ingresadas. El código se inserta en el listado y el campo se actualiza para que solo se pueda buscar el contenido filtrado.

Convertir fechas de un formulario en una variable de tipo fecha

```

Function DateValue(oField As Object) As Date
    a() = OfficeVersion()
    If a(0) = "LibreOffice" And (LEFT(a(1),1) = 4 And
RIGHT(LEFT(a(1),3),1) > 0)
    Or LEFT(a(1),1) > 4 Then

```

Aquí se interceptan todas las versiones de LibreOffice desde la 4.1 en adelante. Para este propósito, el número de versión se divide en sus elementos individuales, y se verifican los números de versión mayor y menor. Esto funcionará hasta LibreOffice 9.

```

        Dim stMonth As String
        Dim stDay As String
        stMonth = Right(Str(0) & Str(oField.CurrentValue.Month),2)
        stDay = Right(Str(0) & Str(oField.CurrentValue.Day),2)
        Datumswert = CDateFromIso(oField.CurrentValue.Year & stMonth &
stDay)
    Else
        DateValue = CDateFromIso(oField.CurrentValue)
    End If
End Function

```

Desde LibreOffice 4.1.2, las fechas se han almacenado como matrices dentro de los controles de formulario. Esto significa que el valor actual del control no se puede usar para acceder a la fecha en sí. La fecha debe recrearse a partir del día, mes y año si se va a seguir usando en macros.

Buscar registros de datos

Se pueden buscar registros en la base de datos sin usar una macro. Sin embargo, configurar una consulta compleja puede llegar a ser una tarea muy complicada. Una macro puede resolver este problema con un simple bucle.

El procedimiento siguiente lee los campos en una tabla, crea una consulta interna y, finalmente, escribe una lista de los números de las claves primarias de registros en la tabla que se recuperan con este término de búsqueda. En la siguiente descripción, hay una tabla llamada *Searchtmp*, que consiste en un campo de clave primaria (*ID*) de incremento automático y un campo llamado *Nr.* que contiene todas las claves primarias recuperadas de la tabla en la que se busca. El nombre de la tabla se proporciona inicialmente como una variable.

Para obtener un resultado correcto, la tabla tiene que incluir el contenido que está buscando como texto y no como claves foráneas. Si es necesario, puede crear una consulta en modo vista para que la macro la use.¹

```

Sub Searching(stTable As String)
    Dim oDataSource As Object

```

1 Consulte la base de datos >Example_Search_and_Filter.odt > asociada a esta guía.

```

Dim oConnection As Object
Dim oSQL_Command As Object
Dim stSql As String
Dim oResult As Object
Dim oDoc As Object
Dim oDrawpage As Object
Dim oForm As Object
Dim oForm2 As Object
Dim oField As Object
Dim stContent As String
Dim arContent() As String
Dim inI As Integer
Dim inK As Integer
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oForm = oDrawpage.forms.getByname("searchform")
oField = oForm.getByname("searchtext")
stContent = oField.getCurrentValue()
stContent = LCase(stContent)

```

El contenido del cuadro de texto de búsqueda se convierte inicialmente en minúsculas, de modo que la función de búsqueda posterior solo necesita comparar lo ingresado en minúsculas.

```

oDataSource = ThisComponent.Parent.DataSource
oConnection = oDataSource.GetConnection("", "")
oSQL_Command = oConnection.createStatement()

```

Primero se debe determinar si realmente se ha ingresado un término de búsqueda. Si el control está vacío, se supone que no se está haciendo una búsqueda y se mostrarán todos los registros.

Si se ingresó un término de búsqueda, los nombres de columna se leen de la tabla en la que se está buscando, de modo que la consulta pueda acceder a los campos.

```

If stContent <> "" Then
    stSql = "SELECT ""COLUMN_NAME"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_COLUMNS"" WHERE ""TABLE_NAME"" = '"
+ stTable + "' ORDER BY ""ORDINAL_POSITION""
    oResult = oSQL_Statement.executeQuery(stSql)

```

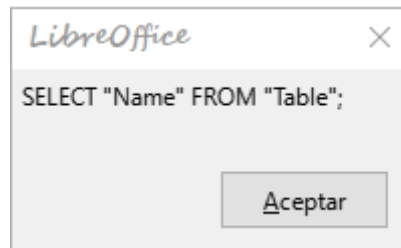


Nota

Las fórmulas SQL en macros primero deben colocarse entre comillas dobles como cadenas de caracteres normales. Los nombres de campo y de tabla ya están entre comillas dobles dentro de la fórmula SQL. Para crear un código final que transmita las comillas dobles correctamente, los nombres de campo y de tabla deben recibir dos conjuntos de estas comillas.

```
stSql = " SELECT ""Name"" FROM ""Table""; "
```

Esto convierte la variable *stSql* a la orden SQL correcta, se puede comprobar usando la función MsgBox de Basic: `MsgBox stSql` .



El índice de la matriz, en el que se escriben los nombres de los campos, se establece inicialmente en 0. Luego, la consulta comienza a leerse. Como se desconoce el tamaño de la matriz, debe ajustarse continuamente. Por eso el ciclo comienza con `ReDim Preserve arContent(inI)` para establecer el tamaño de la matriz y al mismo tiempo preservar el contenido existente. A continuación, se leen los campos y el índice de la matriz se incrementa en 1. Luego, la matriz se dimensiona nuevamente y se puede almacenar un valor adicional.

```
InI = 0
While oResult.next
    ReDim Preserve arContent(inI)
    arContent(inI) = oResult.getString(1)
    inI = inI + 1
Wend
stSql = "DROP TABLE ""searchtmp"" IF EXISTS"
oSQL_Command.executeUpdate (stSql)
```

La consulta se reúne dentro de un bucle y posteriormente se aplica a la tabla definida al principio. Se permiten todas las combinaciones de mayúsculas y minúsculas, ya que el contenido del campo en la consulta se convierte a minúsculas.

La consulta se construye de manera que los resultados terminen en la tabla *searchtmp*. Se supone que la clave primaria es el primer campo de la tabla (`arContent(0)`).

```
stSql = "SELECT """+arContent(0)+""" INTO ""searchtmp"" FROM """ +
stTable _
+ "" WHERE "
For inK = 0 To (inI - 1)
    stSql = stSql+"LCase("""+arContent(inK)+"") LIKE
'%" + stContent + "%'"
    If inK < (inI - 1) Then
        stSql = stSql+" OR "
    End If
Next
oSQL_Command.executeQuery(stSql)
Else
    stSql = "DELETE FROM ""searchtmp""
    oSQL_Command.executeUpdate (stSql)
End If
```

El formulario *display* tiene que recargarse. Su fuente de datos es una consulta, en este ejemplo *Searchquery*.

```
oForm2 = oDrawpage.forms.getByName("display")
oForm2.reload()
End Sub
```

Esto crea una tabla que debe ser evaluada por la consulta. En la medida de lo posible, la consulta debe construirse de modo que pueda editarse posteriormente. Se muestra una consulta como ejemplo:

```
SELECT * FROM "searchtable" WHERE "Nr." IN ( SELECT "Nr." FROM "searchtmp" ) OR "Nr." = CASE WHEN ( SELECT COUNT( "Nr." ) FROM "searchtmp" ) > 0 THEN '0' ELSE "Nr." END
```

Se incluyen todos los elementos de *searchtable*, incluida la clave primaria. Ninguna otra tabla aparece en la consulta directa; por lo tanto, no se necesita ninguna clave primaria de otra tabla y el resultado de la consulta sigue pudiéndose editar.

La clave primaria se guarda en este ejemplo con el nombre *Nr.* La macro lee precisamente este campo. Hay una verificación inicial para ver si el contenido del campo *Nr.* aparece en la tabla *searchtmp*. El operador *IN* es compatible con múltiples valores. La subconsulta también puede generar varios registros.

Para grandes cantidades de datos, la coincidencia de valores mediante el uso del operador *IN* se ralentiza notablemente. Por lo tanto, no es una buena idea usar un campo de búsqueda vacío simplemente para transferir todos los campos de clave primaria de *searchtable* a la tabla *searchtmp* y luego ver los datos de la misma manera. En cambio, un campo de búsqueda vacío crea una tabla *searchtmp* vacía, de modo que no hay registros disponibles. Este es el propósito de la segunda parte de la condición:

```
OR "Nr." = CASE WHEN ( SELECT COUNT( "Nr." ) FROM "searchtmp" ) > 0 THEN '-1' ELSE "Nr." END
```

Si se encuentra un registro en la tabla *searchtmp*, significa que el resultado de la primera consulta es mayor que 0. En este caso: *"Nr." = '-1'* (aquí necesitamos un número que no puede aparecer como una clave primaria, por eso *'-1'* es un buen valor).

Si la consulta arroja exactamente 0 (que será el caso si no hay registros presentes), entonces *"Nr." = "Nr."*. Esto enumerará cada registro que tenga un *Nr.*, al ser *Nr.* la clave primaria, esto significa todos los registros.

Resaltar términos de búsqueda en formularios y resultados

Con un cuadro de texto grande, a menudo no está claro dónde ocurren las coincidencias de un término de búsqueda. Sería más útil si el formulario pudiera resaltar las ocurrencias.

Buscar:	<input type="text" value="Office"/>	<input type="button" value="Mostrar"/>
ID	<input type="text" value="5"/>	
Memo	<div style="border: 1px solid black; padding: 5px;"><p>Introducción</p><p>Los conceptos básicos para crear una base de datos en LibreOffice se describen en el "Capítulo 8, Introducción a Base" de la Guía de primeros pasos. El componente de base de datos de LibreOffice, llamado Base, proporciona una interfaz gráfica para trabajar con bases de datos. Además, LibreOffice contiene una versión del motor de base de datos HSQL. Esta base de datos HSQLDB puede ser utilizada por un solo usuario. Todo el conjunto de datos se almacena en un archivo ODB que tiene un mecanismo de bloqueo de archivos en la carpeta o directorio de configuración cuando lo abre un usuario.</p></div>	

Para que un formulario funcione de esta manera, necesitamos un par de elementos adicionales en nuestra caja de trucos. Esta macro está incluida en las bases de datos de ejemplo, en el archivo *Example_autotext_Searchmark_Spelling.odt*.

El funcionamiento de un campo de búsqueda ya se ha explicado. Se crea una tabla de filtro y se usa un formulario para escribir los valores actuales de un único registro en esta tabla. Al >formulario principal se le proporciona contenido mediante una consulta:

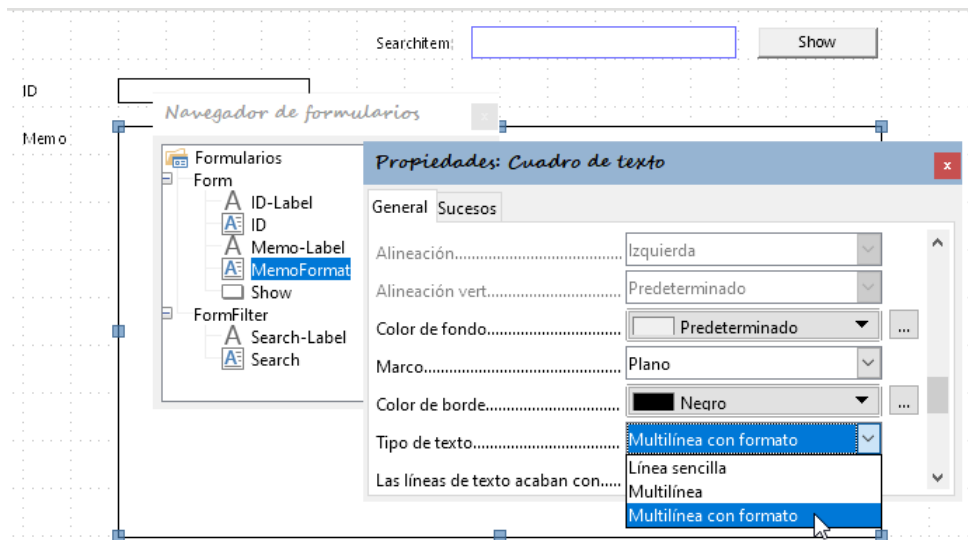
```
SELECT "ID", "memo"
```

```
FROM "table"
WHERE LOWER ( "memo" ) LIKE '%' || LOWER (
( SELECT "searchtext" FROM "filter" WHERE "ID" = TRUE ) ) || '%'
```

Cuando se ingresa el texto de búsqueda, y se pulsa el botón mostrar, se puede acceder a todos los registros en la tabla *table* que contienen el texto ingresado en el campo *memo*.

Se podrá usar los botones de navegación de formulario para acceder solo a los registros que cumplan esa condición. La búsqueda se configura para que no distinga entre mayúsculas y minúsculas.

Si no se ingresa ningún texto de búsqueda, se puede acceder a todos los registros de la tabla mediante los botones de navegación. Como la clave primaria de esta tabla se incluye en la consulta, esta consulta se puede editar.



En el formulario *Form*, además del campo *ID* para la clave primaria, hay un campo *MemoFormat* que se puede configurar para que muestre texto fomateado usando en sus propiedades generales: **Tipo de texto > Multirenglón con formato** dejaremos su color predeterminado (negro).

Al fijarnos en las propiedades del cuadro de texto, la pestaña *Datos* ahora ha desaparecido. Esto se debe a que no se pueden ingresar texto con formato en un campo de la base de datos, la base de datos no lo puede almacenar. Sin embargo, todavía es posible obtener texto en este campo, marcarlo y transferirlo a la base de datos después de una actualización mediante el uso de una macro.

El procedimiento `ContentRead` sirve para transferir el contenido del campo *memo* de la base de datos al cuadro de texto formateado *MemoFormat*, y para formatearlo, en este caso resaltando las coincidencias con el texto ingresado en el campo de búsqueda.

El procedimiento está ligado a las propiedades del Formulario **Sucesos > Tras el cambio de registro de datos**.

```
Sub ContentRead(oEvent As Object)
    Dim inMemo As Integer
    Dim oField As Object
    Dim stSearchtext As String
    Dim oCursor As Object
    Dim inSearch As Integer
    Dim inSearchOld As Integer
    Dim inLen As Integer
```



```

oForm = oEvent.Source
inMemo = oForm.findColumn("memo")
oField = oForm.getByName("MemoFormat")
oField.Text = oForm.getString(inMemo)

```

Primero se definen las variables. Luego, se busca el campo *memo* de tabla (mediante su relación con el formulario), la función `getString()` se usa para leer el texto de la columna numerada. Esto se transfiere al campo que puede formatearse pero que no tiene enlace con la base de datos (*MemoFormat*).

Las pruebas iniciales mostraron que el formulario se abrió, pero la barra de herramientas del formulario en la parte inferior ya no se creó. Para lo que se generó una espera muy corta de 5 milisegundos.

Después de esto, el contenido que se va a buscar se obtiene del formulario *FormFilter* (paralelo al formulario *Form* en la jerarquía de formularios).

```

Wait 5
stSearchtext =
oForm.Parent.getByName("FormFilter").getByName("Search").Text

```

Para poder formatear el texto, se tiene que crear un cursor de texto invisible (`TextCursor`) en el control que contiene el texto (*MemoFormat*). El formato predeterminado del campo utiliza una fuente serif de 12 puntos que puede no aparecer en otras partes del formulario y no puede personalizarse directamente con las propiedades de control del formulario. En este procedimiento, el texto se establece en la apariencia deseada desde el principio. Si esto no se hace, las diferencias en el formato pueden hacer que se corte el límite superior del texto en el campo. En las primeras pruebas, solo 2/3 de la primera línea eran legibles.

Para que el cursor marque el texto, se establece su inicio al principio del campo y luego al se selecciona hasta el final. El argumento en ambos casos es `true`. Después se establecen las propiedades de la fuente (tipo de fuente, color y tamaño) y luego, el cursor vuelve al principio.

```

oCursor = oField.createTextCursor()
oCursor.gotoStart(true)
oCursor.gotoEnd(true)
oCursor.CharHeight = 10
oCursor.CharFontName = "Arial, Helvetica, Tahoma"
oCursor.CharColor = RGB(0,0,0)
oCursor.CharWeight = 100.000000 'com::sun::star::awt::FontWeight
oCursor.gotoStart(false)

```

Si hay texto en el campo formateado y se ha realizado una entrada en el control de búsqueda solicitando una búsqueda, se recorre el contenido para encontrar la cadena de texto. El condicional `If` comprueba primero si se cumplen estas condiciones; el bucle `Do While` comprueba si la cadena buscada está realmente en el texto del campo *MemoFormat*. En realidad, esta configuración podría omitirse, ya que la consulta en la que se basa el formulario solo muestra texto que cumple estas condiciones.

```

If oField.Text <> "" And stSearchtext <> "" Then
  If inStr(oField.Text, stSearchtext) Then
    inSearch = 1
    inSearchOld = 0
    inLen = Len(stSearchtext)

```

Se busca en el texto la cadena introducida. Mediante el bucle que termina cuando no hay más coincidencias. La función `inStr()` devuelve la ubicación del primer carácter de la cadena de

búsqueda en el formato de visualización especificado. El ciclo está controlado por el requisito de que al final de cada ciclo, el valor de `inSearch` se haya incrementado en 1 (-1 en la primera línea del ciclo y +2 en la última línea). Para cada ciclo, el cursor se mueve a la posición inicial sin seleccionar el texto usando `oCursor.goRight(Posición_destino, false)`, y luego a la derecha seleccionando el texto (determinado por la longitud de la cadena de búsqueda) `oCursor.goRight(inLen, true)` y luego se resalta el texto encontrado con el formato deseado (color azul o algo más destacado) `oCursor.CharColor = RGB(102, 102, 255)`. El cursor se mueve de regreso a su nuevo punto de partida para la próxima ejecución.

```

Do While inStr(inSearch, oField.Text, stSearchtext) > 0
    inSearch = inStr(inSearch, oField.Text, stSearchtext) - 1
    oCursor.goRight(inSearch-inSearchOld, false)
    oCursor.goRight(inLen, true)
    oCursor.CharColor = RGB(102, 102, 255)
    oCursor.CharWeight = 110.000000
    oCursor.goLeft(inLen, false)
    inSearchOld = inSearch
    inSearch = inSearch + 2
Loop
End If
End If
End Sub

```

El procedimiento `ContentWrite` sirve para transferir el contenido del cuadro de texto formateado `MemoFormat` a la base de datos independientemente de si se produce o no alguna alteración.

El procedimiento está vinculado a las propiedades del formulario **Sucesos > Antes del cambio de registro**.

```

Sub ContentWrite(oEvent As Object)
    Dim oForm As Object
    Dim inMemo As Integer
    Dim loID As Long
    Dim oField As Object
    Dim stMemo As String
    oForm = oEvent.Source
    If InStr(oForm.ImplementationName, "ODatabaseForm") Then

```

El suceso desencadenante se implementa dos veces. Solo el nombre de implementación que termina con `OdatabaseForm` proporciona el acceso correcto al registro (las implementaciones se explican en la página 70).

```

    If Not oForm.isBeforeFirst() And Not oForm.isAfterLast() Then

```

Cuando el formulario se lee o se vuelve a cargar, el cursor se coloca antes del registro actual. Luego, si se realiza un intento, aparece el mensaje *"Estado de cursor no válido"*.

```

        inMemo = oForm.findColumn("memo")
        loID = oForm.findColumn("ID")
        oField = oForm.getByname("MemoFormat")
        stMemo = oField.Text
        If stMemo <> "" Then
            oForm.updateString(inMemo, stMemo)
        End If

```

```

        If stMemo <> "" And oForm.getString(loID) <> "" Then
            oForm.UpdateRow()
        End If
    End If
End If
End Sub

```

Desde la fuente de datos enlazada al formulario se obtienen los campos *memo* y el campo *ID*. Si el control *MemoFormat* contiene texto se transfiere al campo *memo* mediante `oForm.updateString()`. Solo si el control *ID* del formulario contiene un número (se ha establecido un dato para la clave primaria) se actualiza el registro. De lo contrario, se crea un nuevo registro mediante el funcionamiento normal del formulario. El formulario reconoce el cambio y lo almacena de manera independiente.

Revisar la ortografía durante la entrada de datos

Esta macro se puede usar para cuadros de texto *Multirenglón con formato*. Como en el capítulo anterior, está incluida en `Example_autotext_Searchmark_Spelling.odt`.

Primero se debe escribir el contenido de cada registro y luego se puede cargar el nuevo registro en el control de formulario. Los procedimientos *ContentRead* y *ContentWrite* difieren solo en el punto en el que la función de búsqueda se puede excluir.

ID	<input type="text" value="2"/>
Memo	<p><u>Introducción</u></p> <p>En la operación diaria de la oficina, las hojas de cálculo se usan regularmente para agregar conjuntos de datos y realizar algún tipo de análisis sobre ellos. Como los datos en una hoja de cálculo se presentan en una vista de tabla, claramente visibles y se pueden editar o agregar, muchos usuarios preguntan por qué deberían usar una base de datos en lugar de una hoja de cálculo. Este libro explica las diferencias entre los dos.</p> <p>Este capítulo presenta dos bases de datos de muestra y todo este libro se basa en ellas: <u>Media_without_macros.odt</u> y <u>Media_with_macros.odt</u>, ampliado con la inclusión de macros.</p>

El corrector ortográfico se inicia en el formulario anterior al pulsar la *barra espaciadora* o la tecla *Entrar* dentro del control de texto formateado del formulario. Se ejecuta al final de cada palabra. Y posiblemente se podría combinar con la pérdida de foco del control para garantizar que se verifique la última palabra.

El procedimiento está vinculado con las propiedades del cuadro de texto *Memoformat* **Sucesos > Despues de haber pulsado la tecla.**

```

SUB MarkWrongWordsDirect(oEvent As Object)
    GlobalScope.BasicLibraries.LoadLibrary("Tools") 'carga la biblioteca
    auxiliar "Tools" con funciones utilizadas en el procedimiento

```

La función `RTrimStr` se usa para eliminar cualquier signo de puntuación al final de la cadena, de lo contrario, todas las palabras que terminaban con una coma, punto final u otro signo de puntuación aparecerían como errores ortográficos. Además, `LTrimChar` se usa para eliminar paréntesis de apertura al comienzo de las palabras.

```

Dim aProp() As New com.sun.star.beans.PropertyValue
Dim oLinuSvcMgr As Object
Dim oSpellChk As Object
Dim oField As Object
Dim arText()
Dim stWord As String

```

```

Dim inlenWord As Integer
Dim ink As Integer
Dim i As Integer
Dim oCursor As Object
Dim stText As Object
oLinguSvcMgr =
createUnoService("com.sun.star.linguistic2.LinguServiceManager")
If Not IsNull(oLinguSvcMgr) Then
    oSpellChk = oLinguSvcMgr.getSpellChecker()
End If

```

Primero se declaran todas las variables. Luego se accede al módulo básico de corrección ortográfica `SpellChecker`. Será este módulo el que realmente verificará la corrección de las palabras individuales.

```

oField = oEvent.Source.Model
ink = 0
If oEvent.KeyCode = 1280 Or oEvent.KeyCode = 1284 Then

```

El suceso que inicia la macro es una pulsación de tecla. Este suceso incluye un código, `KeyCode`, para cada clave individual. El `KeyCode` para la tecla *Entrar* es 1280, el de *espacio* es 1284. Al igual que muchos otros datos, estos elementos se pueden inspeccionar mediante la herramienta *XrayTool*. Si se pulsa el *espacio* o el *retorno*, se verifica la ortografía. Se lanza, en otras palabras, al final de cada palabra. Solo la prueba de la última palabra no se produce automáticamente.

Cada vez que se ejecuta la macro, se verifican todas las palabras del texto. También se podría hacer una verificación individual de palabras, pero sería mucho más compleja.

El texto se divide en palabras individuales. El delimitador es el carácter del espacio. Antes de eso, las palabras divididas por saltos de línea deben unirse nuevamente, o los fragmentos de texto podrían confundirse con palabras completas.

```

stText = Join(Split(oField.Text,CHR(10))," ")
stText = Join(Split(stText,CHR(13))," ")
arText = Split(RTrim(stText)," ")
For i = LBound(arText) To Ubound(arText)
    stWord = arText(i)
    inlenWord = len(stWord)
    stWord = Trim( RtrimStr( RtrimStr( RtrimStr( RtrimStr(
RtrimStr(
RtrimStr(stWord,","),
"."),"?"),"!"),"."),")"))
    stWord =
Trim(LTrimChar(LTrimChar(LTrimChar(stWord,"("),")"),";"))

```

Se leen las palabras individuales, su longitud (sin recortar) es necesaria para el siguiente paso de edición. Solo así se puede determinar la posición de la palabra dentro del texto completo (necesario para el resaltado específico de los errores ortográficos).

La función `Trim` se usa para eliminar espacios, mientras que `RTrimStr` elimina comas y signos de puntuación al final del texto y `LTrimChar` cualquier signo de puntuación al principio.

```

If stWord <> "" Then
    oCursor = oField.createTextCursor()
    oCursor.gotoStart(false)
    oCursor.goRight(ink,false)

```

```

oCursor.goRight(inLenWord,true)
If Not oSpellChk.isValid(stWord, "en", aProp()) Then
oCursor.CharUnderline = 9
oCursor.CharUnderlineHasColor = True
oCursor.CharUnderlineColor = RGB(255,51,51)
Else
oCursor.CharUnderline = 0
End If
End If
ink = ink + inLenWord + 1
Next
End If
End Sub

```

Si la palabra no está vacía, se crea un cursor de texto. Este cursor se mueve sin resaltar al comienzo del texto en el campo de entrada. Luego salta hacia la derecha, aún sin resaltar, al término almacenado en la variable `ink`. Esta variable comienza con 0, pero después de que se haya ejecutado el primer bucle, es igual a la longitud de la palabra (+1 para el siguiente espacio).

Luego, el cursor se mueve hacia la derecha según la longitud de la palabra actual. Las propiedades de la fuente se modifican para crear el resaltado.

Se lanza el corrector ortográfico (`spellchecker`) que necesita la palabra a comprobar y el código del país como argumentos; si no se indica un código de país, lo tomará como correcto. El argumento de matriz generalmente está vacío.

Si la palabra no está en el diccionario, está subrayada con una línea roja ondulada. Este tipo de subrayado está representado por el número 9.

Si a palabra está en el diccionario, se elimina el subrayado (0). Este paso es necesario porque de lo contrario una palabra reconocida como falsa y luego corregida continuaría resaltada con la línea ondulada roja. Nunca se eliminaría si no se proporciona el formato para una ortografía correcta.

Cuadros combinados, como listados con opción de entrada

Mientras el control de *Listado* nos permite únicamente elegir un valor dentro de un listado, el control de *Cuadro combinado* funciona como un control de listado, pero además brinda la posibilidad de introducir datos que se pueden almacenar en una tabla.

Mediante el uso de cuadros combinados y un control de campo numérico (invisibles) se puede usar una tabla con dos campos: la clave primaria que servirá como referente de la elección del usuario y el otro para mostrar las entradas correspondientes y almacenar nuevos datos.

En nuestro ejemplo, `Example_Combobox_Listfield.odt`, usamos una tabla principal (*name*) para todos los datos de unos clientes y dos cuadros combinados que obtienen y actualizan la información de otras tablas: la calle y código postal y ciudad. Los cuadros combinados del formulario deben transferir la claves primarias y el contenido relacionado las tablas subsidiarias y además almacenar esa clave primaria en la tabla principal para, con una consulta, obtener todos los datos del cliente. Las claves primarias que se deben transferir se obtienen mediante los controles de campos numéricos invisibles.

Los valores de los controles invisibles se leen cuando se carga el formulario y en el cuadro combinado se puede presentar el contenido relacionado con ese valor. Si se cambia el contenido del cuadro combinado se guarda el contenido mostrado y el valor relacionado (clave primaria).

Si se utilizan consultas que puedan ser editadas en lugar de tablas, el texto que se mostrará en los campos combinados se puede obtener directamente a partir de la consulta, con lo que no se requiere una macro para este paso.

Para el funcionamiento de la macro se entiende que la clave primaria de fuente de datos para el cuadro combinado es un entero de incremento automático y tiene el nombre *ID*.

Visualizar texto en cuadros combinados

El siguiente procedimiento sirve para mostrar texto en el cuadro combinado de acuerdo con el valor de la clave proporcionado por el campo invisible. También se puede usar para listados que hacen referencia a dos tablas diferentes. En este caso, el código postal de una ciudad se almacena en una tabla separada de la tabla de la ciudad. El código postal se lee de una tabla que contiene tres campos: la clave principal, el código postal y una clave foránea para la ciudad. El cuadro combinado debe mostrar el código postal y la ciudad juntos.

```
Sub ShowText(oEvent As Object)
```

Este procedimiento se debe vincular al suceso de formulario *Tras el cambio de registro de datos*.

La macro se llama directamente desde el formulario. El suceso desencadenante es el cambio de datos en los cuadros combinados. Algunas variables ya se han declarado globalmente en un módulo separado y no se declaran aquí nuevamente.

```
Dim oForm As Object
Dim oFieldList As Object
Dim stFieldValue As String
Dim inCom As Integer
Dim stQuery As String
Dim stFieldValue AS String
```

En el formulario hay un control oculto desde el cual se pueden obtener los nombres de los cuadros combinados, estos cuadros combinados son procesados por la macro.

La *Información adicional* (Tag) de las propiedades del control oculto contiene esta lista de nombres de cuadros combinados, separados por comas. Los nombres se escriben en una matriz y todo el proceso tiene lugar dentro de un bucle que finaliza con `NEXT inCom`.

```
oForm = oEvent.Source
aComboboxes() = Split(oForm.getByname("combofields").Tag, ",")
For inCom = LBound(aComboboxes) TO Ubound(aComboboxes)
    ...
Next inCom
```

El cuadro combinado se identifica con `oFieldList`. Para obtener el valor de la clave foránea, necesitamos la columna correcta en la tabla correspondiente. Se puede acceder a él utilizando el nombre del campo de la tabla `stFieldID`, que se almacena en la información adicional del cuadro combinado.

```
oFieldList = oForm.getByname(Trim(aComboboxes(inCom)))
stFieldID = oForm.getString(oForm.findcolumn(oFieldList.Tag))
oFieldList.Refresh()
```

El cuadro combinado se vuelve a leer usando `Refresh()` en caso de que el contenido del campo haya cambiado por la entrada de datos nuevos.

La consulta necesaria para proporcionar el contenido visible al cuadro combinado se basa en el campo subyacente al control y el valor determinado por la clave foránea. Para que el código SQL sea utilizable, se procesa, eliminando cualquier operación de ordenación que pueda estar presente. Luego se realiza una comprobación para cualquier definición de relación (que comience con la palabra `WHERE`). De manera predeterminada, la función `InStr()` no distingue entre mayúsculas y minúsculas, por lo que cualquier combinación de mayúsculas y minúsculas están cubiertas. Si hay una relación, significa que la consulta contiene campos de dos tablas diferentes.

Necesitamos encontrar la tabla que proporciona la clave foránea para el enlace. La macro depende aquí del hecho de que la clave primaria en todas las tablas se llama *ID*.

Si no hay una relación definida, la consulta solo accede a una tabla. La información de la tabla se puede descartar y la condición se puede formular directamente utilizando el valor de la clave foránea.

```
If stFieldID <> "" Then
    stQuery = oFieldList.ListSource
    If InStr(stQuery,"order by") > 0 Then
        stSql = Left(stQuery, InStr(stQuery,"order by")-1)
    Else
        stSql = stQuery
    End If
    If InStr(stSql,"where") Then
        st = Right(stSql, Len(stSql)-InStr(stSql,"where")-4)
        If InStr(Left(st, InStr(st,"=")),"."ID") Then
            a() = Split(Right(st, Len(st)-InStr(st,"=")-1),".")
        Else
            a() = Split(Left(st, InStr(st,"=")-1),".")
        End If
        stSql = stSql + "AND "+a(0)+"."ID" = "+stFieldID
    Else
        stSql = stSql + "WHERE ""ID"" = "+stFieldID
    End If
```

Cada campo y nombre de tabla tiene que referenciarse en la orden SQL con dos conjuntos de comillas. Basic normalmente interpreta las comillas como delimitadores de cadena de texto, por lo que no aparecen cuando el código se pasa a SQL. Duplicar las comillas asegura que se pase un conjunto. ""ID"" significa que se accederá al campo "ID" en la consulta, (el conjunto único de comillas que utiliza SQL).

La consulta almacenada en la variable `stSql` y tras verificar la conexión con la base de datos se lleva a cabo. Su resultado se guarda en `oResult`.

```
oDatasource = ThisComponent.Parent.CurrentController
If Not (oDatasource.isConnected()) Then
    oDatasource.connect()
End If
oConnection = oDatasource.ActiveConnection()
oSQL_Command = oConnection.createStatement()
oResult = oSQL_Command.executeQuery(stSql)
```

El resultado de la consulta se lee en otro bucle. Al igual que cuando se hace una consulta en la interfaz, se pueden mostrar varios campos y registros. Pero la construcción de esta consulta requiere solo un resultado, que se encontrará en la primera columna (1) del conjunto de resultados de la consulta. Es el registro que proporciona el contenido visualizado del cuadro combinado. El contenido es texto (`getString()`), de ahí el comando `oResult.getString(1)`.

```
While oResult.next
    stFieldValue = oResult.getString(1)
Wend
```

El cuadro combinado ahora tiene que contener el valor del texto recuperado por la consulta.

```

    oFieldList.Text = stFieldValue
Else

```

Si no hay ningún valor en el campo para la clave foránea `oField`, la consulta ha fallado y en el cuadro combinado se escribe una cadena sin texto.

```

    oFieldList.Text = ""
End If
Next inCom
End Sub

```

Este procedimiento gestiona el contacto entre el cuadro combinado y la clave foránea disponible en un campo de la fuente de datos del formulario. Esto debería ser suficiente para mostrar los valores correctos en los cuadros combinados. El almacenamiento de nuevos valores requiere un procedimiento adicional.

Transferir un valor de clave foránea de un cuadro combinado a un campo numérico

Si se ingresa un nuevo valor en el cuadro combinado (propósito para el cual se construyó esta macro), la clave primaria correspondiente debe ingresarse en la tabla subyacente del formulario como una clave foránea.

```

Sub TextSelectionSaveValue(oEvent As Object)

```

Este procedimiento se vincula al suceso de formulario: *Antes de la acción en el registro de datos*.

Después de que declarar las variables (no se muestran aquí), Se tiene que determinar exactamente qué suceso debe iniciar la macro. El suceso que hemos vinculado, hace llamada a dos implementaciones en sucesión. Es importante que el procedimiento identifique al formulario para obtener y procesar sus elementos. Se puede hacer con ambas implementaciones pero de diferentes maneras. Aquí se utiliza la implementación `OdatabaseForm`. Para más información sobre las implementaciones >consulte «Un suceso: varias implementaciones» en la página 70.

```

    If InStr(oEvent.Source.ImplementationName, "ODatabaseForm") Then
    ...
End If
End Sub

```

El siguiente bucle se construye con el mismo principio que el procedimiento empleado para mostrar texto en cuadros combinados:

```

oForm = oEvent.Source
aComboboxes() = Split(oForm.GetByName("comboxfields").Tag, ",")
For inCom = LBound(aComboboxes) To UBound(aComboboxes)
    ...
Next inCom

```

El objeto `oFieldList` contiene el texto que se mostrará. Si el cuadro combinado está dentro de un control de tabla, no es posible acceder directamente desde el formulario. En ese supuesto, la información adicional del control oculto debe contener la ruta al campo mediante el cuadro combinado del control de tabla (control de tabla > campo). Con la función `Split` conseguiremos acceder al cuadro combinado.

```

a() = Split(Trim(aComboboxes(inCom)), ">")
If UBound(a) > 0 Then
    oFieldList = oForm.GetByName(a(0)).GetByName(a(1))
Else
    oFieldList = oForm.GetByName(a(0))

```


End If

A continuación, la consulta se lee desde el cuadro combinado y se divide en partes individuales. En un cuadro combinado sencillo, los elementos de información necesarios son el nombre del campo y el nombre de la tabla:

```
SELECT "Field" FROM "Table"
```

En algunos casos, esto podría aumentarse mediante una instrucción de ordenación. Siempre que se junten dos campos en el cuadro combinado, se requerirá más trabajo para separarlos.

```
SELECT "Field1"||' ||'"Field2" FROM "Table"
```

Esta consulta une dos campos con un espacio entre ellos. Como el separador es un espacio, la macro lo buscará y dividirá el texto en dos partes en consecuencia. Naturalmente, esto solo funcionará de manera fiable si *Field1* no contiene espacios en el texto. Como ejemplo, si el primer nombre es "Anne Marie" y el apellido "Müller", la macro tratará a "Anne" como el primer nombre y "Marie Müller" como el apellido. Para esto, se debe utilizar un separador más adecuado, que pueda procesar la macro. En el caso de los nombres, esto podría ser "Apellido, Nombre de pila".

Las cosas se complican aún más si los dos campos provienen de tablas diferentes:

```
SELECT "Table1"."Field1"||' > ||'"Table2"."Field2"  
FROM "Table1", "Table2"  
WHERE "Table1"."ID" = "Table2"."ForeignID"  
ORDER BY "Table1"."Field1"||' > ||'"Table2"."Field2" ASC
```

Los campos se tienen que separar uno de otro, se tiene que indicar la tabla a la que pertenece cada uno y determinar las claves foráneas correspondientes.

```
stQuery = oFieldList.ListSource  
aFields() = Split(stQuery, "||")  
stContent = ""  
For i=LBound(aFields)+1 To UBound(aFields)
```

El contenido de la consulta es despojado de lastre innecesario. Las partes se vuelven a montar en una matriz con una combinación de caracteres inusual como separador. FROM separa la visualización del campo visible de los nombres de las tablas. WHERE separa la condición de los nombres de las tablas. Las uniones mediante JOIN no son compatibles.

```
If Trim(UCASE(aFields(i))) = "ORDER BY" Then  
Exit For  
ElseIf Trim(UCASE(aFields(i))) = "FROM" Then  
stContent = stContent+" §§ "  
ElseIf Trim(UCASE(aFields(i))) = "WHERE" Then  
stContent = stContent+" §§ "  
Else  
stContent = stContent+Trim(aFields(i))  
End If  
Next i  
aContent() = Split(stContent, " §§ ")
```

En algunos casos, el contenido de la visualización del campo visible proviene de diferentes campos:

```
aFirst() = Split(aContent(0),"||")  
If UBound(aFirst) > 0 Then  
If UBound(aContent) > 1 Then
```


La primera parte contiene al menos dos campos. Los campos comienzan con un nombre de tabla. La segunda parte contiene dos nombres de tabla, que se pueden determinar a partir de la primera. La tercera parte contiene una relación con una clave foránea, separada por un igual (=):

```

aTest() = Split(aFirst(0),".")
NameTable1 = aTest(0)
NameTableField1 = aTest(1)
Erase aTest
stFieldSeparator = Join(Split(aFirst(1),""),",")
aTest() = Split(aFirst(2),".")
NameTable2 = aTest(0)
NameTableField2 = aTest(1)
Erase aTest
aTest() = Split(aContent(2),"=")
aTest1() = Split(aTest(0),".")
If aTest1(1) <> "ID" Then
    NameTab12ID = aTest1(1)
    IF aTest1(0) = NameTable1 Then
        Position = 2
    Else
        Position = 1
    End If
Else
    Erase aTest1
    aTest1() = Split(aTest(1),".")
    NameTab12ID = aTest1(1)
    If aTest1(0) = NameTable1 Then
        Position = 2
    Else
        Position = 1
    End If
End If
Else

```

La primera parte contiene dos nombres de campo sin nombres de tabla, posiblemente con separadores. La segunda los nombres de las tablas. No hay una tercera parte:

```

If UBound(aFirst) > 1 Then
    NameTableField1 = aFirst(0)
    stFieldSeparator = Join(Split(aFirst(1),""),",")
    NameTableField2 = aFirst(2)
Else
    NameTableField1 = aFirst(0)
    NameTableField2 = aFirst(1)
End If
NameTable1 = aContent(1)
End If
Else

```

Solo hay un campo de una tabla:

```

    NameTableField1 = aFirst(0)
    NameTable1 = aContent(1)
End If

```

La longitud máxima de caracteres que puede tener una entrada viene dada por la función ColumnSize. El cuadro combinado no se puede usar para limitar el tamaño, ya que puede necesitar contener dos campos al mismo tiempo.

```

LengthField1 = ColumnSize(NameTable1,NameTableField1)
If NameTableField2 <> "" Then
    If NameTable2 <> "" Then
        LengthField2 = ColumnSize(NameTable2,NameTableField2)
    Else
        LengthField2 = ColumnSize(NameTable1,NameTableField2)
    End If
Else
    LengthField2 = 0
End If

```

El contenido del cuadro combinado se lee:

```
stContent = oFieldList.getCurrentValue()
```

Los espacios iniciales y finales y los caracteres de control se eliminan si es necesario.

```

stContent = Trim(stContent)
If stContent <> "" Then
    If NameTableField2 <> "" Then

```

En algunos casos no se detecta el espacio, con las siguientes tres líneas se corrige este fallo

```

    IF ASC(stFieldSeparator) = 32 Then
        stFieldSeparator = " "
    End If

```

Si existe un segundo campo de tabla, el contenido del cuadro combinado tiene que dividirse. Para determinar dónde se producirá la división, usamos el separador de campo como argumento para la función.

```
a_stParts = Split(stContent, stFieldSeparator, 2)
```

El último parámetro significa que el número máximo de partes es 2.

Dependiendo de qué entrada corresponde al campo 1 y cuál al campo 2, el contenido del cuadro combinado ahora se asigna a las variables individuales. `Position = 2` > sirve aquí como una señal de que la segunda parte del contenido representa el Campo 2.

```

If Position = 2 Then
    stContent = Trim(a_stParts(0))
    If UBound(a_stParts()) > 0 Then
        stContentField2 = Trim(a_stParts(1))
    Else
        stContentField2 = ""
    End If
    stContentField2 = Trim(a_stParts(1))
Else
    stContentField2 = Trim(a_stParts(0))

```

```

    If UBound(a_stParts()) > 0 Then
        stContent = Trim(a_stParts(1))
    Else
        stContent = ""
    End If
    stContent = Trim(a_stParts(1))
End If
End If

```

Puede suceder que con dos contenidos separables, el tamaño previsto del cuadro combinado (longitud del texto) no se ajuste a los campos de la tabla que deben ser guardados. Para los cuadros combinados que representan un solo campo, esto normalmente se maneja configurando adecuadamente el control de formulario. Aquí, por el contrario, necesitamos alguna forma de detectar estos errores. Se verifica la longitud máxima permitida del campo relevante.

```

    If (LengthField1 > 0 And Len(stContent) > LengthField1) Or
        (LengthField2 > 0 And Len(stContentField2) > LengthField2) Then

```

Si la longitud del campo de la primera o segunda parte es demasiado grande, se almacena una cadena predeterminada en una de las variables. El carácter Chr (13) se usa para insertar un salto de línea.

```

        stmsgbox1 = "El campo " + NameTableField1 + " no debe exceder de
" + Field1Length + "caracteres de longitud." + Chr(13)
        stmsgbox2 = "El campo " + NameTableField2 + " no debe exceder de
" + Field2Length + "caracteres de longitud." + Chr(13)

```

Si ambos contenidos de campo son demasiado largos, se muestran ambos textos.

```

    If (LengthField1 > 0 And Len(stContent) > LengthField1) And
        (LengthField2 > 0 And Len(stContentField2) > LengthField2) Then
        MsgBox("El texto introducido es demasiado largo. " + Chr(13) +
stmsgbox1 + stmsgbox2 + "Reduzca su longitud.",64,"Entrada
Inválida")

```

El procedimiento utiliza la función MsgBox() para presentar un mensaje. Esta función necesita como primer argumento una cadena de texto (mensaje a mostrar), luego, opcionalmente, un número (que determina el tipo de diálogo de mensaje que se muestra) y, finalmente, una cadena de texto opcional como título para la ventana. Por lo tanto, el diálogo del mensaje tendrá el título *Entrada Inválida*. El número 64 proporciona un icono con el símbolo de Información.

El siguiente código cubre cualquier otro caso de texto excesivamente largo que pueda surgir.

```

    ElseIf (Field1Length > 0 And Len(stContent) > Field1Length) Then
        MsgBox("El texto introducido es demasiado largo. " + Chr(13) +
stmsgbox1 + "Reduzca su longitud.",64,"Entrada Inválida")
    Else
        MsgBox("El texto introducido es demasiado largo. " + Chr(13) +
stmsgbox2 + "Reduzca su longitud.",64,"Entrada inválida")
    End If
Else

```

Si no hay texto excesivamente largo, el procedimiento continúa sin presentar el mensaje.

Ahora las entradas están enmascaradas para que las comillas que puedan estar presentes no generen un error.

```

stContent = String_to_SQL(stContent)
If stContentField2 <> "" Then
    stContentField2 = String_to_SQL(stContentField2)
End If

```

Las primeras variables se asignan previamente y la consulta puede modificarlas posteriormente. Las variables `inID1` e `inID2` almacenan el contenido de los campos de clave primaria de las dos tablas.

Las variables se establecen en -1

Al iniciar variables numéricas (de tipo integer), Basic les asigna el valor de 0. Si una consulta no produce resultados el valor devuelto por Basic sería 0, lo que también podría indicar que la consulta ha tenido éxito. (la consulta devuelve una clave primaria 0). Al establecer la variable en -1 se desambigua esta salida, puesto que HSQLDB no puede establecer este valor para un campo de clave primaria con numeración automática.

A continuación, se establece la conexión con la base de datos, si no se hubiera establecido.

```

inID1 = -1
inID2 = -1
oDatasource = ThisComponent.Parent.CurrentController
If Not (oDatasource.isConnected()) Then
    oDatasource.connect()
End If
oConnection = oDatasource.ActiveConnection()
oSQL_Command = oConnection.createStatement()
If NameTableField2 <> "" And Not IsEmpty(stContentField2) And
NameTable2 <> "" Then

```

Si existe un segundo campo de tabla, se tiene que declarar una segunda dependencia.

```

stSql = "SELECT ""ID"" FROM "" + NameTable2 + "" WHERE "" +
NameTableField2 + ""="" + stContentField2 + ""
oResult = oSQL_Command.executeQuery(stSql)
While oResult.next
    inID2 = oResult.getInt(1)
Wend
If inID2 = -1 Then
    stSql = "INSERT INTO "" + NameTable2 + "" ("" +
NameTableField2 + "" ) VALUES ('" + stContentField2 + "') "
    oSQL_Command.executeUpdate(stSql)
    stSql = "CALL IDENTITY()"

```

Si el contenido dentro del cuadro combinado no está presente en la tabla correspondiente, se inserta allí. Luego se lee el valor de la clave primaria resultante. Si está presente, la clave primaria >se lee de la misma manera. La función utiliza los campos de clave primaria generados automáticamente (`IDENTITY`).

```

oResult = oSQL_Command.executeQuery(stSql)
While oResult.next
    inID2 = oResult.getInt(1)
Wend
End If

```

La clave primaria para el segundo valor se almacena temporalmente en la variable `inID2` y luego se escribe como clave primaria en la tabla correspondiente al primer valor. En función de si el registro de la primera tabla ya estaba disponible, el contenido se guarda (INSERT) o se altera (UPDATE).

```

If inID1 = -1 Then
    stSql = "INSERT INTO "" + NameTable1 + "" (" + NameTableField1 + "", "" + NameTab12ID + "") VALUES (' + stContent + ', ' + inID2 + ')"
    oSQL_Command.executeUpdate(stSql)

```

Y el correspondiente *ID* se lee directamente:

```

stSql = "CALL IDENTITY()"
oResult = oSQL_Command.executeQuery(stSql)
While oResult.next
    inID1 = oResult.getInt(1)
Wend

```

La clave primaria para la primera tabla se tiene que leer nuevamente para que se pueda transferir a la tabla subyacente del formulario.

```

Else
    stSql = "UPDATE "" + NameTable1 + "" SET "" + NameTab12ID + ""=' + inID2 + ' WHERE "" + NameTableField1 + "" = ' + stContent + '"
    oSQL_Command.executeUpdate(stSql)
End If
End If

```

En el caso de que ambos campos subyacentes al cuadro combinado estén en la misma tabla (por ejemplo, *Apellido* y *Nombre* de la tabla *Nombre*), se necesita una consulta diferente:

```

If NameTableField2 <> "" And NameTable2 = "" Then
    stSql = "SELECT ""ID"" FROM "" + NameTable1 + "" WHERE "" + NameTableField1 + ""=' + stContent + ' AND "" + NameTableField2 + ""=' + stContentField2 + '"
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        inID1 = oResult.getInt(1)
    Wend
    If inID1 = -1 Then

```

y no existe una segunda tabla:

```

stSql = "INSERT INTO "" + NameTable1 + "" (" + NameTableField1 + "", "" + NameTableField2 + "") VALUES (' + stContent + ', ' + stContentField2 + ')"
oSQL_Command.executeUpdate(stSql)

```

La clave primaria se lee de nuevo.

```

stSql = "CALL IDENTITY()"
oResult = oSQL_Command.executeQuery(stSql)
While oResult.next

```

```

        inID1 = oResult.getInt(1)
    Wend
End If
End If
IF NameTableField2 = "" Then

```

Ahora se considera el caso más simple: el segundo campo de la tabla no existe y la entrada aún no está presente en la tabla. En otras palabras, se ha ingresado un nuevo valor en el cuadro combinado.

```

    stSql = "SELECT ""ID"" FROM "" + NameTable1 + "" WHERE "" +
NameTableField1 + ""=''" + stContent + ""'"
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        inID1 = oResult.getInt(1)
    Wend
    If inID1 = -1 Then

```

Si no hay un segundo campo, el contenido del cuadro combinado se inserta como un nuevo registro.

```

    stSql = "INSERT INTO "" + NameTable1 + "" ("" +
NameTableField1 + """) VALUES ('" + stContent + "'" ) "
    oSQL_Command.executeUpdate(stSql)

```

y el ID resultante se lee directamente.

```

    stSql = "CALL IDENTITY()"
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        inID1 = oResult.getInt(1)
    Wend
End If
End If

```

El valor del campo de clave primaria tiene que determinarse, de modo que se pueda transferir a la parte principal del formulario.

A continuación, el valor de la clave primaria resultante de todos estos bucles se transfiere al campo invisible en la tabla principal y la base de datos subyacente. El campo de tabla vinculado al campo de formulario se obtiene mediante `BoundField` y mediante `updateInt` se inserta un número entero en este campo (consulte las definiciones de tipo numérico).

```

    oForm.updateLong(oForm.findColumn(oFeldList.Tag),inID1)
End If
ELSE

```

Si no se va a ingresar una clave primaria, porque no hubo entrada en el cuadro combinado o esa entrada se eliminó, el contenido del campo invisible también debe eliminarse. La instrucción `updateNull()` se usa para llenar el campo con la expresión para un campo vacío (NULL).

```

    oForm.updateNULL(oForm.findColumn(oFeldList.Tag),NULL)
End If
NEXT inCom
End If
End Sub

```

Función para medir la longitud de la entrada del cuadro combinado

La siguiente función proporciona el número de caracteres en la columna de la tabla respectiva, de modo que las entradas que son demasiado largas no se trunquen. Se elige una función para proporcionar valores de retorno, puesto que un procedimiento no proporciona ningún valor de retorno que pueda procesarse en otro lugar.

```
Function ColumnSize(TableName As String, Fieldname As String) As Integer
    oDatasource = ThisComponent.Parent.CurrentController
    If Not (oDatasource.isConnected()) Then
        oDatasource.connect()
    End If
    oConnection = oDataSource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    stSql = "SELECT ""COLUMN_SIZE"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_COLUMNS"" WHERE ""TABLE_NAME"" = '"
+ TableName + "' AND ""COLUMN_NAME"" = '" + Fieldname + "'"
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        i = oResult.getInt(1)
    Wend
    ColumnSize = i
End Function
```

Generar acciones de base de datos

```
Sub GenerateRecordAction(oEvent As Object)
```

Esta macro debe estar vinculada al suceso *Texto modificado* del cuadro combinado. Es necesario que en todos los casos en que se cambia el contenido ese cambio se almacene. Sin esta macro, no habría cambios en la tabla real que Base pudiera reconocer, ya que el cuadro combinado no está vinculado al formulario.

Esta macro altera directamente las propiedades del formulario:

```
    Dim oForm As Object
    oForm = oEvent.Source.Model.Parent
    oForm.IsModified = TRUE
End Sub
```

Esta macro no es necesaria para formularios que utilizan consultas para el contenido de cuadros combinados. Los cambios en los cuadros combinados se registran directamente.

Navegar de un formulario a otro

Se abrirá un formulario cuando ocurra un suceso particular.

En las propiedades de control de formulario, en la línea *Información adicional* (Tag), ingrese el nombre del formulario. También se puede ingresar más información aquí, y luego separarla mediante la función `Split()`.

```
Sub From_form_to_form(oEvent As Object)
    Dim stTag As String
    stTag = oEvent.Source.Model.Tag
    aForm() = Split(stTag, ",")
End Sub
```

La matriz se declara y se llena con los nombres de los formularios, primero el formulario que se abrirá y, en segundo lugar, el formulario actual, que se cerrará después de que se haya abierto el otro.

```
ThisDatabaseDocument.FormDocuments.GetByName(  
Trim(aForm(0)) ).open  
ThisDatabaseDocument.FormDocuments.GetByName(  
Trim(aForm(1)) ).close  
End Sub
```

Si, en cambio, el otro formulario solo se debe abrir cuando el actual está cerrado, por ejemplo, cuando existe un formulario principal y todos los demás formularios se controlan a través de botones, la siguiente macro debe vincularse al formulario con **Herramientas > Personalizar > Sucesos > Documento cerrado**:

```
Sub Mainform_open  
ThisDatabaseDocument.FormDocuments.GetByName( "Mainform" ).open  
End Sub
```

Si los documentos del formulario se ordenan dentro del archivo ODB en directorios, el procedimiento para cambiar el formulario debe ser más extenso:

```
Sub From_form_to_form_with_folders(oEvent As Object)  
REM El formulario que se debe abrir se indica en primer lugar.  
REM Si un formulario está en una carpeta, use "/" para definir la  
relación  
REM para que se pueda encontrar la subcarpeta.  
Dim stTag As String  
stTag = oEvent.Source.Model.Tag 'La etiqueta se ingresa en la  
información adicional  
aForms() = Split(stTag, ",") 'Aquí viene primero el nombre del  
formulario nuevo, luego el del formulario anterior  
aForms1() = Split(aForms(0),"/")  
aForms2() = Split(aForms(1),"/")  
If UBound(aForms1()) = 0 Then  
  
ThisDatabaseDocument.FormDocuments.GetByName( Trim(aForms1(0)) ).ope  
n  
Else  
ThisDatabaseDocument.FormDocuments.GetByName(  
Trim(aForms1(0)) ).GetByName( Trim(aForms1(1)) ).open  
End If  
If UBound(aForms2()) = 0 Then  
  
ThisDatabaseDocument.FormDocuments.GetByName( Trim(aForms2(0)) ).clo  
se  
Else  
ThisDatabaseDocument.FormDocuments.GetByName(  
Trim(aForms2(0)) ).GetByName( Trim(aForms2(1)) ).close  
End If
```


End Sub

Los documentos de formulario que se encuentran en un directorio se ingresan en el campo Información adicional como directorio / formulario. Tienen que convertirse a:

```
...getByName("Directorio").getByName("Form")
```

Listados jerárquicos

La configuración en un campo de listado tiene la intención de influir directamente en la configuración de otro. Para casos simples, esto ya se ha descrito anteriormente en la sección sobre filtrado de registros. Pero supongamos que el primer listado está destinado a afectar el contenido del segundo listado, que luego afecta el contenido de un tercer listado, y así sucesivamente.

Años	Clase	Nombre
1	a	Karl Müller
2	b	Evelyn Maier
3	c	Maria Gott
4	d	Eduard Abgefahren
5	e	Kurt Drechsler
6	f	Kunigunde Schimmel
7		
8		
9		
10		
11		
12		
13		

Ejemplos de campos de lista para un ordenamiento jerárquico

En este ejemplo, el primer listado *Años* contiene todos los años escolares. El segundo, *Clase* las clases que en cada año están representadas por letras. El tercero *Nombre* contiene los nombres de los miembros de la clase.

En circunstancias normales, el listado *Años* mostraría los 13 años, el listado *Clases* todas las letras de clase y el listado *Nombres* todos los alumnos de la escuela.

Si se trata de listado jerárquicos, la elección de clases se restringe una vez que se ha seleccionado un año. Solo se muestran las letras de clase que están realmente presentes en ese año. Esto puede variar porque, si el número de alumnos aumenta, el número de clases en un año también podría aumentar. El último listado, *Nombres*, es muy restringido. En lugar de más de 1000 alumnos, mostraría solo 30.

Al principio, solo se puede seleccionar el año. Una vez hecho esto, la lista (restringida) de clases estará disponible. Solo al final se da la lista de nombres.

Si se modifica el listado *Años*, la secuencia debe comenzar de nuevo. Si solo se modifica el listado *Clases*, el número de año sigue siendo válido. Para crear dicha función, el formulario debe poder almacenar una variable intermedia. Esto tiene lugar en un control oculto.

La macro está vinculada a un cambio en el contenido de un listado: **Propiedades Listado > Sucesos > Modificado**. Las variables necesarias se almacenan en la información adicional del listado.

He aquí un ejemplo de la información adicional para el control oculto:

```
MainForm,Year,hidden_control,Listbox_2
```

El formulario se llama *MainForm*. El listado actual se llama *Listbox1*. Este listado muestra el contenido del campo de la tabla *Año* y los siguientes listados se tienen que filtrar de acuerdo con

esta entrada. El control oculto se designa mediante `hidden_control` y la existencia de un segundo listado (`Listbox_2`) se pasa al procedimiento de filtrado.

```
Sub Hierarchical_control(oEvent As Object)
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oForm As Object
    Dim oFieldHidden As Object
    Dim oField As Object
    Dim oField1 As Object
    Dim stSql As String
    Dim acontent()
    Dim stTag As String
    oField = oEvent.Source.Model
    stTag = oField.Tag
    oForm = oField.Parent

    REM la etiqueta va en el campo de información adicional
    REM Contiene:
    REM 0. Nombre del campo para el filtrado de la tabla
    REM 1. Nombre del campo del control oculto que almacenará el valor
    filtrado
    REM 2. Un posible listado adicional
    REM La etiqueta se lee del elemento que inicia la macro. La
    variable se pasa
    REM al procedimiento y si es necesario a todos los listados
    adicionales
    aFilter() = Split(stTag, ",")
    stFilter = ""
```

Una vez que se han declarado los variables, el contenido de la etiqueta se pasa a una matriz, para que se pueda acceder a los elementos individuales. Luego se declara el acceso a los distintos controles del formulario.

Se determina el listado que llamó a la macro y se lee su valor. Solo si este valor no está vacío (NULL) se combinará con el nombre del campo que se va a filtrar >para hacer una orden SQL (Año en nuestro ejemplo). De lo contrario, el filtro permanecerá vacío. Si los listados están destinados a filtrar un formulario, no hay ningún control oculto disponible. En este caso, el valor del filtro se almacena directamente en el formulario.

```
    If Trim(aFilter(1)) = "" Then
        If oField.GetCurrentValue <> "" Then
            stFilter = """"+Trim(aFilter(0))
+""""=''+oField.GetCurrentValue()+"""
```

Si ya existe un filtro (por ejemplo, uno relacionado con `Listbox_2`, al que ahora se está accediendo), el nuevo contenido se adjunta al contenido anterior almacenado en el control oculto.

```
    If oForm.Filter <> ""
```

Solo puede darse cuando el mismo campo aún no se ha filtrado. Por ejemplo, si se está filtrando por Año, una repetición del filtro no encontrará registros adicionales para el listado `Nombre`. Solo

se puede encontrar a una persona en un año. Por lo tanto, se tiene que excluir la posibilidad de que el nombre del filtro ya se haya utilizado.

```
And InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='') = 0
Then
    stFilter = oForm.Filter + " AND " + stFilter
```

Si existe un filtro y el campo que se utilizará para el filtrado ya está presente en el filtro, se tiene que eliminar el filtrado anterior en este nombre de campo y crear un nuevo filtro.

```
ElseIf oForm.Filter <> "" Then
    stFilter = Left(oForm.Filter,
        InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='')-1) +
stFilter
End If
End If
```

Luego, el filtro se ingresa en el formulario. Este filtro también puede estar vacío si se seleccionó el primer listado y no tiene contenido.

```
oForm.Filter = stFilter
oForm.reload()
```

El mismo procedimiento >se ejecutará si el formulario no necesita ser filtrado inmediatamente. En este caso, el valor del filtro se almacena en el control oculto.

```
Else
    oFieldHidden = oForm.getByname(Trim(aFilter(1)))
    If oField.getCurrentValue <> "" Then
        stFilter = """"+Trim(aFilter(0))
+"""=''+oField.getCurrentValue()+"""
        If oFieldHidden.HiddenValue <> ""
            And InStr(oFieldHidden.HiddenValue, """"+Trim(aFilter(0))
+"""='') = 0 Then
                stFilter = oFieldHidden.HiddenValue + " AND " + stFilter
            ElseIf oFieldHidden.HiddenValue <> "" Then
                stFilter = Left(oFieldHidden.HiddenValue,
                    InStr(oFieldHidden.HiddenValue, """"+Trim(aFilter(0))
+"""='')-1) +
                    stFilter
            End If
        End If
    End If
    oFieldHidden.HiddenValue = stFilter
End If
```

Si la información adicional tiene una entrada numerada 4 (la numeración comienza en 0), el siguiente listado debe establecerse en la entrada correspondiente del listado que hace la llamada.

```
If UBound(aFilter()) > 1 Then
    oField1 = oForm.getByname(Trim(aFilter(2)))
    aFilter1() = Split(oField1.Tag, ",")
```

Los datos necesarios para el filtrado se leen de la Información adicional (*Tag*) en el listado correspondiente. Desafortunadamente, no es posible escribir solo el código SQL nuevo en el listado y luego leer los valores del listado. Los valores correspondientes a la consulta tienen que escribirse directamente en el listado.

La creación del código se basa en que la tabla a la que se refiere el formulario es la misma a la que se refieren los listados. Tal listado no está diseñado para transferir claves foráneas a la tabla.

```
If oField.GetCurrentValue <> "" Then
    stSql = "SELECT DISTINCT """+Trim(aFilter1(0))+"" FROM
"""+oForm.Command+ "" WHERE "+stFilter+" ORDER BY
"""+Trim(aFilter1(0))+""
    oDatasource = ThisComponent.Parent.CurrentController
    If Not (oDatasource.isConnected()) Then
        oDatasource.connect()
    End If
    oConnection = oDatasource.ActiveConnection()
    oSQL_Statement = oConnection.createStatement()
    oQuery_result = oSQL_Statement.executeQuery(stSql)
```

Los valores se leen en una matriz. La matriz se transfiere directamente al listado. Los índices correspondientes para la matriz se incrementan dentro de un bucle.

```
inIndex = 0
While oQuery_result.next
    ReDim Preserve aContent(inIndex)
    acontent(inIndex) = oQuery_result.getString(1)
    inIndex = inIndex+1
WEnd
Else
    aContent(0) = ""
End If
oField1.StringItemList = aContent()
```

El contenido del listado se ha modificado y tiene que ser leído de nuevo.

Luego, utilizando la propiedad de información adicional del listado que se ha actualizado, se vacía cada uno de los listados dependientes siguientes, iniciando un bucle para todos los listados siguientes hasta que se llegue a uno que no tenga un cuarto término en su información adicional.

```
oField1.refresh()
While UBound(aFilter1()) > 1
    Dim aLeer()
    oField2 = oForm.getByName(Trim(aFilter1(2)))
    Dim aFilter1()
    aFilter1() = Split(oField2.Tag, ",")
    oField2.StringItemList = aEmpty()
    oField2.refresh()
Wend
End If
End Sub
```

El contenido visible de los listados se almacena en `oField1.StringItemList`. Si algún valor adicional necesita ser almacenado para su transmisión a la tabla subyacente como una clave foránea, (habitual para los listados en formularios), este valor tiene que pasarse a la consulta por separado y luego almacenarse con `oField1.ValueItemList`.

Dicha extensión requiere variables adicionales, además de la tabla en la que se almacenarán los valores del formulario, la tabla de la que se extraen los contenidos del listado.

Se tiene que tener especial cuidado al formular la consulta de filtro.

```
stFilter = """+Trim(aFilter(1))+""="'+oField.GetCurrentValue()  
+"""
```

solo funcionará si la versión de LibreOffice es 4.1 o posterior, ya que es el valor que se debe almacenar lo que se proporciona como `CurrentValue()`, y no el valor que se muestra. Para asegurarse de que funciona en diferentes versiones, establezca la propiedad del listado **Datos > Campo enlazado > '0'**.

Ingresar horas con milisegundos

Para almacenar tiempos a una precisión de milisegundos se requiere un campo de marca de tiempo en la tabla, adaptado por separado por SQL para este propósito (consulte «Creación de tablas» en el «Capítulo 3»). Dicho campo se puede representar en un formulario mediante un campo formateado con el formato `MM:SS.00`. Sin embargo, en el primer intento de escribir en él, la entrada de registro fallará. Esto se puede corregir con el siguiente procedimiento, que debe estar vinculado a la propiedad del formulario **Sucesos > Antes de la acción en el registro de datos**:

```
SUB Timestamp  
Dim unoStmp As New com.sun.star.util.DateTime  
Dim oDoc As Object  
Dim oDrawpage As Object  
Dim oForm As Object  
Dim oFeld As Object  
Dim stZeit As String  
Dim ar()  
Dim arMandS()  
Dim loNano As Long  
Dim inSecond As Integer  
Dim inMinute As Integer  
oDoc = thisComponent  
oDrawpage = oDoc.Drawpage  
oForm = oDrawpage.Forms.getByName("MainForm")  
oField = oForm.getByName("Time")  
stTime = oField.Text
```

Las variables se declaran primero. El resto del código se ejecuta solo cuando el campo Hora tiene contenido. De lo contrario, el mecanismo interno del formulario actuará para establecer el campo vacío (NULL).

```
If stTime <> "" Then  
ar() = Split(stTime, ".")  
loNano = CLng(ar(1)&"0000000")  
arMandS() = Split(ar(0), ":")  
inSecond = CInt(arMandS(1))  
inMinute = CInt(arMandS(0))
```

La entrada en el campo Hora se divide en sus elementos:

Primero, la parte decimal se separa y se rellena a la derecha con caracteres nulos hasta un total de nueve dígitos. Un número tan alto solo puede almacenarse en una variable de tipo `Long`.

Luego, el resto del tiempo se divide en minutos y segundos, utilizando los dos puntos como separador, y estos se convierten en enteros.

```
With unoStmp
    .NanoSeconds = loNano
    .Seconds = inSecond
    .Minutes = inMinute
    .Hours = 0
    .Day = 30
    .Month = 12
    .Year = 1899
End With
```

Los valores de la marca de tiempo se asignan a la fecha estándar de LibreOffice (30.12.1899). Por supuesto, la fecha presente real se puede almacenar junto a ella.



Nota

Obtener y almacenar la fecha presente:

```
Dim now As Date
now = Now()
With unoStmp
    .NanoSeconds = loNano
    .Seconds = inSecond
    .Minutes = inMinute
    .Hours = Hour(now)
    .Day = Day(now)
    .Month = Month(now)
    .Year = Year(now)
End With    oField.BoundField.updateTimestamp(unoStmp)
End If
End Sub
```

Ahora la marca de tiempo que hemos creado se transfiere al campo usando `updateTimestamp` y se almacena en el formulario.

En tutoriales anteriores, los `NanoSeconds` se llamaban `HundrethSeconds`. Esto no coincide con la Interfaz de Programación (API) de LibreOffice y causará un mensaje de error.

Un suceso: varias implementaciones

Al usar formularios, puede ocurrir que una macro vinculada a un solo suceso se ejecute dos veces. Esto ocurre porque más de un proceso está vinculado simultáneamente a, por ejemplo, el almacenamiento de un registro modificado. Las diferentes causas de tal suceso se pueden determinar de la siguiente manera:

```
Sub Determine_eventcause(oEvent As Object)
    Dim oForm As Object
    oForm = oEvent.Source
    MsgBox oForm.ImplementationName
End Sub
```

Cuando se almacena un registro modificado, hay dos implementaciones involucradas llamadas `org.openoffice.comp.svx.FormController` y `com.sun.star.comp.forms.ODatabaseForm`. Con estos nombres, podemos asegurarnos de

que una macro solo se ejecute a través de su código una vez. Una ejecución duplicada generalmente causa una pausa (pequeña) en la ejecución del programa, pero puede también llevar a situaciones extrañas, como que un cursor devuelva dos registros en lugar de uno. Cada implementación solo permite comandos específicos, por lo que conocer el nombre de la implementación es importante.

Guardar con confirmación

Para alteraciones complicadas de registros, lo lógico es preguntar al usuario antes si el cambio realmente debe llevarse a cabo. Si la respuesta en el diálogo es *No*, se cancela el proceso de guardado, se descarta el cambio y el cursor permanece en el registro actual.

```
Sub Save_confirmation(oEvent As Object)
    Dim oFormFeature As Object
    Dim oFormOperations As Object
    Dim inAnswer As Integer
    oFormFeature = com.sun.star.form.runtime.FormFeature
    Select Case oEvent.Source.ImplementationName
        Case "org.openoffice.comp.svx.FormController"
            inAnswer = MsgBox("¿Desea modificar el registro?" ,4,
"Change_record")
            Select Case inAnswer
                Case 6 ' Yes, no hay acción posterior
                Case 7 ' No, se deshace el cambio del registro
                    oFormOperations = oEvent.Source.FormOperations
                    oFormOperations.execute(oFormFeature.UndoRecordChanges)
                Case Else
            End Select
        Case "com.sun.star.comp.forms.ODatabaseForm"
    End Select
End Sub
```

Hay dos momentos de activación con diferentes nombres de implementación. Estas dos implementaciones se distinguen en `SELECT CASE`. El código se ejecutará solo para la implementación de `FormController`. Esto se debe a que solo `FormController` tiene la variable `FormOperations`.

Además de *Sí* y *No*, el usuario también puede hacer clic en el botón *Cerrar*. Sin embargo, esto produce el mismo valor que *No*, es decir, **7**.

Si se navega por el formulario con la tecla de tabulación, el usuario solo ve el diálogo con el mensaje de confirmación. Sin embargo, los usuarios que usan la barra de navegación también verán un mensaje que indica que el registro no se alterará.

Clave primaria con año en curso y número

Cuando se preparan las facturas, los saldos anuales se ven afectados. Esto a menudo lleva a un deseo de separar las tablas de facturas de una base de datos por año y comenzar una nueva tabla cada año.

La siguiente solución macro utiliza un método diferente. Escribe automáticamente el valor del campo *ID* en la tabla, pero también tiene en cuenta el campo *año* que existe en la tabla como otra clave. La combinación de estas claves forman la clave primaria de la tabla. Por lo tanto, los campos siguientes deben aparecer en la tabla:

<i>year</i>	<i>ID</i>
2014	1
2014	2
2014	3
2015	1
2015	2

De esta forma, se obtiene una descripción general del año más fácilmente para documentos.

En la base de datos de ejemplo: `Example_serial_Number_Year.odt` puede comprobar su funcionamiento.

```

Sub Current_Date_and_ID
  Dim oDatasource As Object
  Dim oConnection As Object
  Dim oSQL_Command As Object
  Dim stSql As String
  Dim oResult As Object
  Dim oDoc As Object
  Dim oDrawpage As Object
  Dim oForm As Object
  Dim oField1 As Object
  Dim oField2 As Object
  Dim oField3 As Object
  Dim inIDnew As Integer
  Dim inYear As Integer
  Dim unoDate
  oDoc = thisComponent
  oDrawpage = oDoc.drawpage
  oForm = oDrawpage.forms.getByName("MainForm")
  oField1 = oForm.getByName("fmt_year")
  oField2 = oForm.getByName("fmtID")
  oField3 = oForm.getByName("dat_date")
  If IsEmpty(oField2.getCurrentValue()) Then
    If IsEmpty(oField3.getCurrentValue()) Then
      unoDate = createUnoStruct("com.sun.star.util.Date")
      unoDate.Year = Year(Date)
      unoDate.Month = Month(Date)
      unoDate.Day = Day(Date)
      inYear = Year(Date)
    Else
      inYear = oField3.CurrentValue.Year
    End If
    oDatasource = ThisComponent.Parent.CurrentController
    If Not (oDatasource.isConnected()) Then
      oDatasource.connect()
    End If
    oConnection = oDatasource.ActiveConnection()

```



```

oSQL_Command = oConnection.createStatement()
stSql = "SELECT MAX( ""ID"" )+1 FROM ""orders"" WHERE ""year"" =
""
    + inYear + ""
oResult = oSQL_Command.executeQuery(stSql)
While oResult.next
    inIDnew = oresult.getInt(1)
Wend
If inIDnew = 0 Then
    inIDnew = 1
End If
oField1.BoundField.updateInt(inYear)
oField2.BoundField.updateInt(inIDnew)
If IsEmpty(oField3.getCurrentValue()) Then
    oField3.BoundField.updateDate(unoDate)
End If
End If
End Sub

```

Todas las variables están declaradas. Los controles del formulario principal son accesibles. El resto del código solo se ejecuta si la entrada en el campo *fmtID* todavía está vacía. Si no se ha ingresado una fecha, se crea una estructura de fecha para que, fecha y año presentes se trasladen a los campos relevantes. Y se realiza una conexión a la base de datos, si aún no existe.

El valor más alto del campo *ID* para el año en curso se incrementa en 1. Si el conjunto de resultados está vacío, es que no hay entradas en el campo *ID*. En este punto, se podría ingresar 0 en el control *fmtID*, pero la numeración de las órdenes debe comenzar en 1, por lo que la variable *inIDnew* recibe el valor 1.

El valor devuelto para el año, *ID* y *fecha* (si no se ha ingresado ninguna fecha) se transfieren al formulario. En el formulario, los campos de claves primarias para el *ID* y el *año* están protegidos contra escritura. En consecuencia, solo se les puede dar valores usando esta macro.

Ampliación de tareas de bases de datos mediante macros

Crear una conexión a una base de datos

```

oDataSource = ThisComponent.Parent.DataSource
If Not oDataSource.IsPasswordRequired Then
    oConnection = oDataSource.GetConnection("", "")

```

Sería posible proporcionar un nombre de usuario y una contraseña, si fuera necesario. En ese caso, los paréntesis contendrían ("Nombre de usuario", "Contraseña"). En lugar de incluir el nombre de usuario y una contraseña visibles, se abre el diálogo de protección de contraseñas:

```

Else
    oAuthentication =
    createUnoService("com.sun.star.sdb.InteractionHandler")
    oConnection = oDataSource.ConnectWithCompletion(oAuthentication)
End If

```

Sin embargo, si un formulario dentro del mismo archivo Base está accediendo a la base de datos, solo necesita:

```

oDataSource = ThisComponent.Parent.CurrentController
If Not (oDataSource.isConnected()) Then
    oDataSource.connect()
End If
oConnection = oDataSource.ActiveConnection()

```

Aquí se conoce la base de datos, por lo que no es necesario un nombre de usuario y una contraseña, ya que estos están desactivados en la configuración básica de HSQLDB para la versión interna.

Para formularios fuera de Base, la conexión se realiza a través del primer formulario:

```

oDataSource = ThisComponent.Drawpage.Forms(0)
oConnection = oDataSource.ActiveConnection

```

Copiar datos de una base de datos a otra

La base de datos interna es una base de datos de usuario único. Los registros se almacenan dentro del archivo *.odb. El intercambio directo de datos entre diferentes archivos de base de datos no estaba permitido pero, sin embargo, es posible mediante la exportación e importación.

Los archivos *.odb se configuran frecuentemente para permitir el intercambio automático de datos entre bases de datos.

El siguiente procedimiento puede ser útil.

Después de declarar las variables, la ruta a la base de datos actual se lee desde un botón en el formulario. El nombre de la base de datos está separado del resto de la ruta. El archivo de destino para los registros también está presente en esta carpeta. El nombre de este archivo se adjunta a la ruta para permitir que se realice una conexión a la base de datos de destino.

La conexión a la base de datos de origen se determina en relación con el formulario que contiene el botón: `ThisComponent.Parent.CurrentController`.

La conexión a la base de datos externa se configura utilizando `DatabaseContext` y la ruta.

```

Sub DataCopy
    Dim oDatabaseContext As Object
    Dim oDataSource As Object
    Dim oDataSourceZiel As Object
    Dim oConnection As Object
    Dim oConnectionZiel As Object
    Dim oDB As Object
    Dim oSQL_Command As Object
    Dim oSQL_CommandTarget As Object
    Dim oResult As Object
    Dim oResultTarget As Object
    Dim stSql As String
    Dim stSqlTarget As String
    Dim inID As Integer
    Dim inIDTarget As Integer
    Dim stName As String
    Dim stTown As String
    oDB = ThisComponent.Parent
    stDir = Left(oDB.Location, Len(oDB.Location) - Len(oDB.Title))
    stDir = ConvertToUrl(stDir & "TargetDB.odb")

```

```

oDatasource = ThisComponent.Parent.CurrentController
If Not (oDatasource.isConnected()) Then
    oDatasource.connect()
End If
oConnection = oDatasource.ActiveConnection()
oDatabaseContext =
createUnoService("com.sun.star.sdb.DatabaseContext")
oDatasourceTarget = oDatabaseContext.getByName(stDir)
oConnectionTarget = oDatasourceTarget.GetConnection("", "")
oSQL_Command = oConnection.createStatement()
stSql = "SELECT * FROM ""table""
oResult = oSQL_Command.executeQuery(stSql)
While oResult.next
    inID = oResult.getInt(1)
    stName = oResult.getString(2)
    stTown = oResult.getString(3)
    oSQL_CommandTarget = oConnectionTarget.createStatement()
    stSqlTarget = "SELECT ""ID"" FROM ""table"" WHERE ""ID"" =
""+inID+""
    oResultTarget = oSQL_CommandZiel.executeQuery(stSqlTarget)
    inIDZiel = - 1
    While oResultTarget.next
        inIDTarget = oResultTarget.getInt(1)
    Wend
    If inIDTarget = - 1 Then
        stSqlTarget = "INSERT INTO ""table""
(""ID"", ""name"", ""town"") VALUES
    ('"+inID+"', '"+stName+"', '"+stTown+"')"
        oSQL_CommandTarget.executeUpdate(stSqlZiel)
    End If
Wend
End Sub

```

Las tablas completas de la base de datos de origen se leen e insertan, línea por línea, en las tablas de la base de datos de destino utilizando la conexión que se ha configurado. Antes de la inserción, se realiza una comprobación para ver si se ha establecido un valor para la clave primaria. Si es así, el registro no se copia.

También se puede organizar que, en lugar de copiar un nuevo registro, se actualice un registro existente. En todos los casos, esto asegura que la base de datos de destino contiene registros con la clave primaria correcta de la base de datos de origen.

Acceso a consultas

Es más fácil crear consultas en la interfaz gráfica de usuario que transferir su texto a macros, con la complicación adicional de las dobles comillas duplicadas para todos los nombres de tablas y campos.

```

Sub aQueryContent
    Dim oDatabaseFile As Object
    Dim oQuery As Object

```

```

Dim stQuery As String
oDatabaseFile = ThisComponent.Parent.CurrentController.DataSource
oQuery = oDatabaseFile.getQueryDefinitions()
stQuery = oQuery.getByName("Query").Command
MsgBox stQuery
End Sub

```

Así se accede al contenido de un archivo *.odb desde un formulario. A la consulta se accede usando `getQueryDefinitions()`. El código SQL para la consulta está en su campo *Command*. Se puede obtener para utilizar la orden SQL dentro de una macro.

Cuando utiliza el código SQL de la consulta, debe tener cuidado de que el código no haga referencia a otra consulta. Lo que conlleva inevitablemente al mensaje de que la tabla de la base de datos es desconocida (aparentemente). Debido a esto, es más sencillo crear vistas a partir de consultas y luego acceder a las vistas en la macro.

Asegurar la base de datos

En algunas ocasiones puede ocurrir que el archivo *.odb se trunque inesperadamente, y de manera especial al crear una base de datos. Por eso, es recomendable guardar con cierta frecuencia la base de datos después de una edición, sobre todo cuando se utilizan Informes.

El archivo de base de datos puede dañarse por un fallo del sistema operativo en el momento en que la base de datos está en uso. Particularmente, cuando se termina el archivo Base, momento en el que el contenido de la base de datos tiene que escribirse en el archivo.

Además, existen los habituales supuestos de archivos que repentinamente no se pueden abrir, por ejemplo por un fallo del disco duro. Por lo tanto, es conveniente tener una copia de seguridad lo más actualizada posible. El estado de los datos no cambia mientras el archivo *.odb permanezca abierto.

Por esta razón, los procedimientos de seguridad se pueden vincular directamente a la apertura del archivo. Simplemente copie el archivo utilizando la ruta de respaldo proporcionada en **Herramientas > Opciones > LibreOffice > Rutas**. Esta macro sobrescribe la versión más antigua de la copia de seguridad después de un número específico de copias (`inMax`).

```

Sub Databasebackup(inMax As Integer)
  Dim oPath As Object
  Dim oDoc As Object
  Dim sTitle As String
  Dim sUrl_End As String
  Dim sUrl_Start As String
  Dim i As Integer
  Dim k As Integer
  oDoc = ThisComponent
  sTitle = oDoc.Title
  sUrl_Start = oDoc.URL
  Do While sUrl_Start = ""
    oDoc = oDoc.Parent
    sTitle = oDoc.Title
    sUrl_Start = oDoc.URL
  Loop

```

Si la macro se ejecuta al iniciar el archivo *.odb, `sTitle` y `sUrl_Start` serán correctos. En cambio, si la macro se ejecuta mediante un formulario, primero debe determinar si hay una URL

disponible para la copia de seguridad. Si la URL está vacía, se busca en un nivel superior (oDoc.Parent).

```
oPath = createUnoService("com.sun.star.util.PathSettings")
For i = 1 To inMax + 1
    If Not FileExists(oPath.Backup & "/" & i & "_" & sTitle) Then
        If i > inMax Then
            For k = 1 To inMax - 1 To 1 Step -1
                If FileDateTime(oPath.Backup & "/" & k & "_" & sTitle) <=
FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitle) Then
                    If k = 1 Then
                        i = k
                        Exit For
                    End If
                Else
                    i = k + 1
                    Exit For
                End If
            Next
        End If
        Exit For
    End If
Next
sUrl_End = oPath.Backup & "/" & i & "_" & sTitle
FileCopy(sUrl_Start,sUrl_End)
End Sub
```

También puede hacer una copia de seguridad mientras Base se está ejecutando, siempre que los datos se puedan escribir en el archivo antes de que se lleve a cabo el procedimiento de copia de seguridad DatabaseBackup. Esto se puede hacer mediante una secuencia de tiempo programada o al presionar un botón en el formulario. Este borrado de memoria intermedia se maneja con el siguiente procedimiento:

```
Sub Write_data_out_of_cache
    Dim oData As Object
    Dim oDataSource As Object
    oData = ThisDatabaseDocument.CurrentController
    If Not ( oData.isConnected() ) Then oData.connect()
    oDataSource = oData.DataSource
    oDataSource.flush
End Sub
```

Si todo esto se inicia desde un solo botón en un formulario, se tiene que programar otro procedimiento que inicie ambos procedimientos:

```
Sub BackupNow
    Write_data_out_of_cache
    DatabaseBackup(10)
End Sub
```

Especialmente para una macro de seguridad, sería más lógico hacer que la macro fuera accesible a través de la barra de herramientas de la base de datos. Esto se hace en la ventana principal del archivo Base usando **Herramientas > Personalizar > Barras de herramientas**.

En el diálogo *Personalizar* que se muestra en la figura 1, en debe usar el nombre del archivo de la base de datos como *Ámbito*, para que la *Función* se guarde en el archivo, que en este caso es *Media_with_Macros.odb*.

Para que siempre sea visible en la base de datos es recomendable seleccionar la barra de herramientas *Estándar* como *Destino*, puesto que está disponible en todas las ventanas de Base.

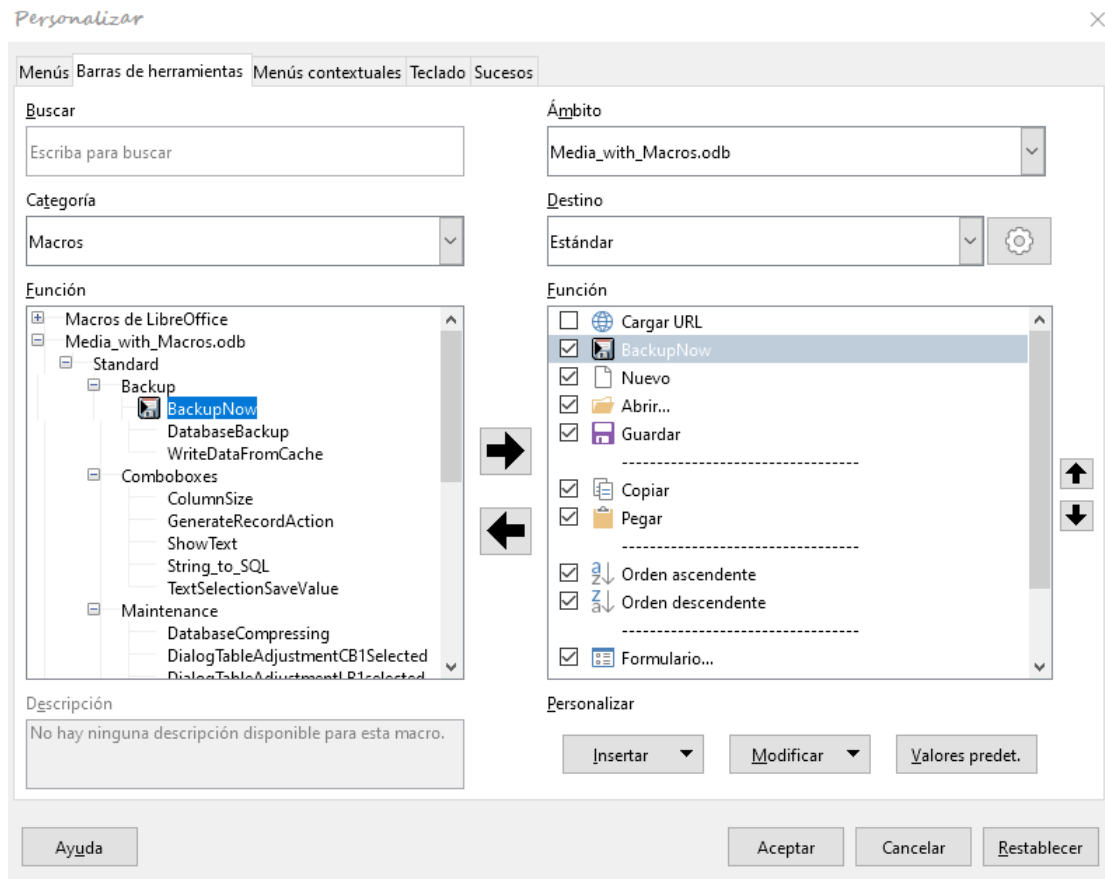
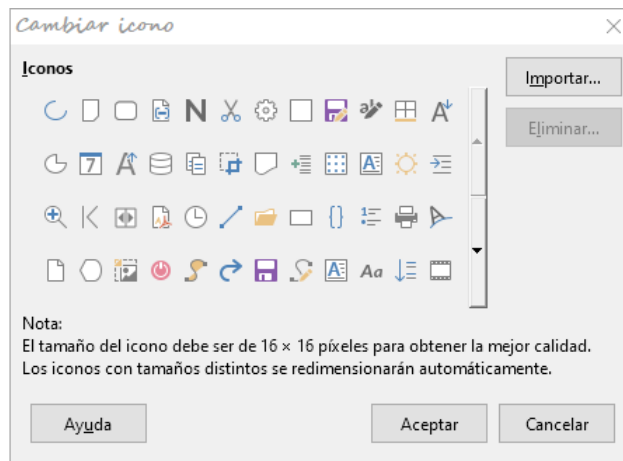


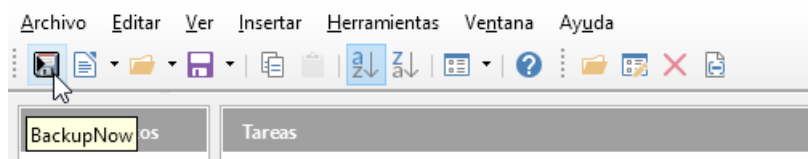
Figura 1: Personalización de la barra de herramientas

El diálogo ahora muestra funciones relevantes, en la lista de la derecha. Seleccione la *categoría >Macros* y elija el procedimiento *BackupNow*.

El comando estará disponible para su uso en la barra de herramientas. Para asignar un icono al comando, elija el nombre que ha asignado al nuevo botón y mediante **Modificar > Cambiar icono** se abrirá el siguiente diálogo.



Seleccione un icono adecuado para el botón (también puede crear y agregar su propio icono).



El icono ahora aparece en lugar del nombre del procedimiento. El nombre se convierte en una información emergente cuando se sitúa el ratón sobre el icono. Para ejecutar el procedimiento, simplemente haga clic en el icono en la barra de herramientas.

Compactar la base de datos

Esto es simplemente una orden SQL (SHUTDOWN COMPACT), que debe ejecutarse de vez en cuando, especialmente después de que se hayan eliminado muchos datos. La base de datos almacena datos nuevos, pero aún reserva el espacio de los datos eliminados. En los casos en que los datos se hayan alterado sustancialmente, necesita por tanto compactar la base de datos.



Nota

Desde LibreOffice versión 3.6, esta orden se ejecuta automáticamente para el HSQLDB interno cuando se cierra la base de datos. Por lo tanto, esta macro ya no es necesaria para una base de datos interna.

Una vez que se lleva a cabo la compactación, las tablas no son accesibles. Para poder acceder al contenido, el archivo tiene que ser reabierto. Por lo tanto, esta macro cierra el formulario desde el que se la llama. Lamentablemente, no puede cerrar el documento en sí mismo sin provocar una recuperación cuando se abre de nuevo. Por eso esta instrucción está comentada.

```
Sub Database_compaction
  Dim stMessage As String
  oDataSource = ThisComponent.Parent.CurrentController ' Accesible
desde el formulario
  If Not (oDataSource.isConnected()) Then
    oDataSource.connect()
  End If
  oConnection = oDataSource.ActiveConnection()
  oSQL_Statement = oConnection.createStatement()
  stSql = "SHUTDOWN COMPACT" ' La base de datos se está compactando y
cerrando
  oSQL_Statement.executeQuery(stSql)
```

```

    stMessage = "Se va a compactar la base de datos." + Chr(13) + "Se
cerrará el formulario."
    stMessage = stMessage + Chr(13) + "Después de esto, la base de
datos debe cerrarse"
    stMessage = stMessage + Chr(13) + "Solo tendrá acceso a la base de
datos después de la reapertura del archivo de base de datos ."
    MsgBox stMessage

```

```

ThisDatabaseDocument.FormDocuments.getByName( "Maintenance" ).close
    REM El cierre del archivo de base de datos mediante el siguiente
comando provoca una operación de recuperación al abrirla nuevamente.
Por eso se comenta el comando de cierre.
' ThisDatabaseDocument.close(True)
End Sub

```

Disminuir el índice de la tabla para los campos automáticos

Si se eliminan muchos datos de una tabla, a los usuarios a menudo les preocupa que la secuencia de claves primarias generadas automáticamente continúe aumentando desde el último número asignado en lugar de comenzar a partir del último valor más alto de los registros que quedan (si en una tabla tenemos 60 registros y se borran los 10 últimos, el siguiente registro que se inserte tendrá el número 61, en lugar del esperado 51). El siguiente procedimiento lee el valor más alto existente del campo *ID* en una tabla y establece el siguiente valor sumando 1 a este máximo.

Si el campo de la clave primaria tiene un nombre distinto de *ID*, se tiene que editar la macro para utilizar ese nombre.

```

Sub Table_index_down(stTable As String)
    REM Se establece el valor siguiente de la clave primaria ID con el
mínimo valor posible.
    Dim inCount As Integer
    Dim inSequence_Value As Integer
    oDataSource = ThisComponent.Parent.CurrentController ' Accessible
mediante el formulario
    If Not (oDataSource.isConnected()) Then
        oDataSource.connect()
    End If
    oConnection = oDataSource.ActiveConnection()
    oSQL_Statement = oConnection.createStatement()
    stSql = "SELECT MAX(""ID"") FROM ""+stTable+"" " ' Se obtiene el
valor más alto de "ID"
    oQuery_result = oSQL_Statement.executeQuery(stSql) ' Se inicia la
consulta y el valor de retorno se almacena en la variable
oQuery_result
    If Not IsNull(oQuery_result) Then
        While oQuery_result.next
            inCount = oQuery_result.getInt(1) ' Se lee el primer campo de
datos
        Wend ' siguiente registro, en este caso ninguno, ya que solo
existe un registro

```



```

    If inCount = "" Then ' Si la tabla está vacía no se obtiene
ningún valor, y el valor >más alto se establece en -1
        inCount = -1
    End If
    inSequence_Value = inCount+1 ' El valor más alto se incrementa
en 1
REM Se prepara una nueva orden para la base de datos. La ID del
siguiente registro comenzará ahora desde inCount + 1.
REM Esta operación no tiene valor de retorno, ya que no se está
leyendo ningún registro
    oSQL_statement = oConnection.createStatement()
    oSQL_statement.executeQuery("ALTER TABLE "" + stTable + ""
ALTER COLUMN ""ID"" RESTART WITH " + inSequence_Value + "")
    End If
End Sub

```

Imprimir desde Base

La forma estándar de imprimir un documento desde Base es usar un informe. Alternativamente, las tablas y consultas se pueden copiar y preparar su impresión desde Calc. Por supuesto, también es posible la impresión directa de un formulario desde la pantalla.

Imprimir un informe desde un formulario interno

Normalmente, la generación de informes se realiza desde la interfaz Base de usuario. Un clic en el nombre del informe inicia la preparación del informe. Sería más fácil, por supuesto, si el informe se pudiera lanzar directamente desde un formulario.

```

Sub Reportlaunch
    ThisDatabaseDocument.ReportDocuments.getByName("Report").open
End Sub

```

Se accede a todos los informes por nombre desde su contenedor ReportDocuments. Se abren con Open. Si un informe está vinculado a una consulta que se filtra a través del formulario, este método permite imprimir el registro seleccionado.

Lanzar, formatear, imprimir directamente y cerrar un informe

Sería aún mejor si el informe se pudiera enviar directamente a la impresora. La siguiente combinación de procedimientos agrega algunos pequeños detalles. Primero selecciona el registro activo en el formulario, vuelve a formatear el informe para que los cuadros de texto se configuren automáticamente para la altura correcta y luego inicia el informe. Finalmente, el informe se imprime y, opcionalmente, se guarda como un pdf. Y todo esto sucede casi por completo en segundo plano, ya que el informe se oculta después de que se abre el formulario y se cierra tras la impresión. Andrew Pitonyak, Thomas Krumbein y Lionel Elie Mamane hicieron sugerencias para los diversos procedimientos.

```

Sub ReportStart(oEvent As Object)
    Dim oForm As Object
    Dim stSql As String
    Dim oDatasource As Object
    Dim oConnection As Object
    Dim oSQL_command As Object
    Dim oReport As Object

```

```

Dim oReportView As Object
oForm = oEvent.Source.model.parent
stSql = "UPDATE ""Filter"" SET ""Integer"" = '" +
    oForm.getInt(oForm.findColumn("ID")) + "' WHERE ""ID"" = TRUE"
oDatasource = ThisComponent.Parent.CurrentController
If Not (oDatasource.isConnected()) Then
    oDatasource.connect()
End If
oConnection = oDatasource.ActiveConnection()
oSQL_command = oConnection.createStatement()
oSQL_command.executeUpdate(stSql)
oReport =
ThisDatabaseDocument.ReportDocuments.getByname("Reportname").open
oReportView = oReport.CurrentController.Frame.ContainerWindow
oReportView.Visible = False
ReportLineHeightAuto(oReport)
End Sub

```

El procedimiento `ReportStart` está vinculado a un botón en el formulario. Usando este botón, se puede leer la clave primaria del registro seleccionado. Desde el suceso que inicia la macro, podemos llegar al formulario (`oForm`). El nombre del campo de clave primaria en este caso es `ID`. Usando `oForm.getInt(oForm.findColumn("ID"))`, la clave se lee desde el campo como un entero. Este valor se almacena en una tabla de filtro. La tabla de filtro controla una consulta para garantizar que solo se utilizará el registro seleccionado para el informe.

El informe se puede abrir sin referencia al formulario. Se hace accesible como un objeto (`oReport`). La ventana del informe se vuelve invisible. Desafortunadamente, no puede ocultarse durante la activación, por lo que aparece brevemente y se llena con el contenido apropiado en segundo plano.

A continuación, se inicia el procedimiento `ReportLineHeightAuto` que recibe como argumento una referencia al informe abierto.

La altura de la línea del registro se puede configurar automáticamente en el momento de la impresión. Si hay demasiado texto en un campo en particular, el texto se trunca y el resto se indica con un triángulo rojo. Cuando esto no funciona, el siguiente procedimiento garantizará que en todas las tablas con el nombre `Detail`, se active el control automático de altura.

```

Sub ReportLineHeightAuto(oReport As Object)
    Dim oTables As Object
    Dim oTable As Object
    Dim inT As Integer
    Dim inI As Integer
    Dim oRows As Object
    Dim oRow As Object
    oTables = oReport.getTextTables()
    For inT = 0 To oTables.count() - 1
        oTable = oTables.getByIndex(inT)
        If Left$(oTable.name, 6) = "Detail" Then
            oRows = oTable.Rows
            For inI = 0 To oRows.count - 1
                oRow = oRows.getByIndex(inI)
            Next inI
        End If
    Next inT
End Sub

```

```

        oRow.IsAutoHeight = True
    Next inI
End If
Next inT
PrintCloseReport(oReport)
End Sub

```

Cuando se crea el informe, se debe tener cuidado de que todos los campos en la misma línea de la sección *Detail* tengan la misma altura. De lo contrario, el control automático de altura puede establecer una línea a doble altura.

Una vez que en todas las tablas con el nombre *Detail* se ha establecido el control automático de altura, el informe se envía a la impresora mediante el procedimiento `PrintCloseReport`.

La matriz *Props* contiene los ajustes de los valores asociados con la impresión del documento. Para el comando de impresión, es necesario el nombre de la impresora de salida. El informe debe permanecer abierto hasta que se complete la impresión. Esto se garantiza dando el nombre de la impresora y `Wait` como argumentos.

```

Sub PrintCloseReport(oReport As Object)
    Dim Props
    Dim stPrinter As String
    Props = oReport.getPrinter()
    stPrinter = Props(0).value
    Dim arg(1) As New com.sun.star.beans.PropertyValue
    arg(0).name = "Name"
    arg(0).value = "<" & stPrinter & ">"
    arg(1).name = "Wait"
    arg(1).value = True
    oReport.print(arg())
    oReport.close(true)
End Sub

```

Solo cuando la impresión se ha enviado completamente a la impresora se cierra el documento.

Para conocer la configuración de la impresora, consulte las secciones «Impresora» y «Configuración de impresión» de la ayuda.

Si, en lugar de (o además de) una copia impresa, desea obtener un una copia de seguridad en formato *pdf* del documento, se puede usar el método `storeToURL()`:

```

Sub ReportPDFstore(oReport As Object)
    Dim stUrl As String
    Dim arg(0) As New com.sun.star.beans.PropertyValue
    arg(0).name = "FilterName"
    arg(0).value = "writer_pdf_Export"
    stUrl = "file:///...."
    oReport.storeToURL(stUrl, arg())
End Sub

```

La URL tiene que ser una dirección URL completa (`file:///Ruta/Nombre_Archivo`).

Si además esta dirección está vinculada a un registro permanente de la base de datos o del documento impreso, como un número de factura se evita que la próxima impresión sobrescriba el archivo PDF de copia de seguridad.

Imprimir informes desde un formulario externo

Hay problemas cuando se utilizan formularios externos. Los informes se encuentran dentro del archivo *.odb y no están disponibles mediante el navegador de origen de datos.

```
Sub Reportstart(oEvent As Object)
    Dim oField As Object
    Dim oForm As Object
    Dim oDocument As Object
    Dim oDocView As Object
    Dim Arg()
    oField = oEvent.Source.Model
    oForm = oField.Parent
    sURL = oForm.DataSourceName
    oDocument = StarDesktop.LoadComponentFromURL(sURL, "_blank", 0,
Arg() )
    oDocView = oDocument.CurrentController.Frame.ContainerWindow
    oDocView.Visible = False
    oDocument.GetCurrentController().connect
    Wait(100)
    oDocument.ReportDocuments.GetByName("Report").open
    oDocument.close(True)
End Sub
```

El informe se inicia desde un botón en el formulario externo que traslada al formulario la ruta del archivo: `oForm.DataSourceName`. El archivo se abre mediante `LoadComponentFromURL`, y este debe permanecer en segundo plano, por lo que se accede a la vista del documento y la interfaz se establece en `Visible = False`. En principio, esto debería haberse hecho directamente usando la lista de argumentos `Arg()`, pero lamentablemente, esto no produce el resultado correcto.

El informe no se puede llamar inmediatamente desde el documento abierto, ya que la conexión aún no está establecida. (El informe aparece con un fondo gris y LibreOffice se bloquea). Con una breve espera de aproximadamente 100 milisegundos se puede resolver el problema. (Es posible que necesite hacer pruebas para determinar el tiempo mínimo de espera).

Después se lanza el informe, como el informe estará en un archivo de texto separado, el archivo *.odb abierto puede cerrarse nuevamente. El método `oDocument.close(True)` pasa esta instrucción al archivo *.odb, que solo se cerrará cuando ya no esté activo, es decir, cuando no se pasen más registros al informe.

Se puede iniciar un acceso similar desde los formularios dentro del archivo *.odb, pero en este caso el documento no debe cerrarse.

Utilizando macros combinadas con la función de combinación de correspondencia o cuadros de texto, puede obtener impresiones de buena calidad bastante más rápido que con el Generador de informes.

Combinación de correspondencia desde Base

Desde Base se puede hacer una *Combinación de correspondencia* mediante informes, pero los cuadros de texto de los informes son de uso limitado. Para crear una carta modelo de cierta calidad se puede utilizar Writer.

No es necesario abrir Writer, personalizar la carta y luego imprimir desde este programa, todo este proceso se puede automatizar mediante una macro y una plantilla.

```

Sub MailmergePrint
  Dim oMailMerge As Object
  Dim aProps()
  oMailMerge = CreateUnoService("com.sun.star.text.MailMerge")

```

La base de datos tiene que estar registrada en LibreOffice, y el nombre que se empleó para registrarla será el que se utilice para acceder a los datos. No es necesario que el nombre sea el mismo que el del archivo que se utilice para la combinación de correspondencia. En este ejemplo se utilizará *Addresses* como el nombre registrado de la base de datos

```
oMailMerge.DataSourceName = "Addresses"
```

La ruta al archivo de salida (*mailmerge.odt*) debe formatearse para poder usarlo en LibreOffice en este ejemplo se utiliza, una ruta absoluta en un sistema Linux .

```
oMailMerge.DocumentURL =
ConvertToUrl("home/user/Documentos/mailmerge.odt")
```

Se establece el tipo de comando: 0 representa una tabla, 1 una consulta y 2 una orden SQL.

```
oMailMerge.CommandType = 1
```

Se ha elegido una consulta con el nombre *MailmergeQuery*.

```
oMailMerge.Command = "MailmergeQuery"
```

Se utiliza un filtro para determinar qué registros se utilizarán para la impresión de la correspondencia. Este filtro podría, especificarse utilizando un control de formulario y acceder a él desde la macro. El uso de la clave primaria de un registro puede hacer que se imprima un solo documento.

En este ejemplo, se selecciona el campo *Gender* en *MailmergeQuery* y luego se buscan registros que tengan *m* en este campo.

```
oMailMerge.Filter = ""Gender"='m' "
```

Los tipos de salida disponibles son Impresora (1), Archivo (2) y Correo (3). Utilizaremos un archivo de salida para las pruebas que guardaremos en una ruta establecida. Por cada registro se creará un archivo y para evitar la sobrescritura utilizaremos el campo *Surname* (Apellido) que se agregará al nombre del archivo de salida.

```
oMailMerge.OutputType = 2
oMailMerge.OutputUrl = ConvertToUrl("home/user/Documents")
oMailMerge.FileNameFromColumn = True
oMailMerge.FileNamePrefix = "Surname"
oMailMerge.execute(aProps())
End Sub
```

Si el filtro se proporciona con sus datos a través del formulario posibilita la combinación de correspondencia sin abrir Writer.

Imprimir mediante campos de texto

Crearemos una plantilla en Writer para personalizar la carta modelo Para completar los datos a partir de la base de datos se deben usar campos, Estos campos se insertan usando **Insertar > Campo > Más campos > Funciones > Marcador de posición**.

Los marcadores de posición deben tener los mismos nombres que los nombres de los campos en la base de datos o consulta desde la que se llama a la macro y el tipo de marcador para un caso sencillo debe ser *Texto*.

Se debe indicar la ruta de la plantilla para que el procedimiento para crear las cartas rellene los marcadores de posición en la plantilla con el contenido de el/los registros elegidos por la consulta. La base de datos de ejemplo `Example_database_mailmerge_direct.odt` muestra cómo se puede generar una factura completa con la ayuda de campos de texto y el acceso a una tabla preparada dentro de la plantilla de documento. Esta manera de creación de documentos no tiene las limitaciones de altura de los campos que nos podemos encontrar en un informe creado con el generador de informes.

Aquí está parte del código, suministrado principalmente por DPunch:

<http://de.openoffice.info/viewtopic.php?f=8&t=45868#p194799>

```
Sub Filling_Textfields
  oForm = thisComponent.Drawpage.Forms.MainForm
  If oForm.RowCount = 0 Then
    MsgBox "Registro no disponible para la impresión"
    Exit Sub
  End If
```

El formulario principal está activo. El botón que inicia la macro también podría usarse para indicar el formulario. La macro comprueba que el formulario realmente contenga datos imprimibles.

```
oColumns = oForm.Columns
oDB = ThisComponent.Parent
```

El acceso directo a la URL de la plantilla desde el formulario no es posible. Tiene que hacerse utilizando la referencia de nivel superior a la base de datos.

```
stDir = Left(oDB.Location, Len(oDB.Location) - Len(oDB.Title))
```

Con esta instrucción obtenemos la URL del directorio donde se aloja la base de datos (sin el nombre del archivo de Base).

```
stDir = stDir & "Plantilla_con_Campos.odt" 'Se indica la plantilla y se
abre creando un nuevo documento.
Dim args(0) As New com.sun.star.beans.PropertyValue
args(0).Name = "AsTemplate"
args(0).Value = True
oNewDoc = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, args)
```

Los campos de texto se rellenan con los datos

```
oTextfields = oNewDoc.Textfields.createEnumeration
Do While oTextfields.hasMoreElements
  oTextfield = oTextfields.nextElement
  If
oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit")
Then
  stColumnName = oTextfield.PlaceHolder
```

El marcador de posición `oTextfield.PlaceHolder` representa el campo de texto.

```
If oColumns.hasByName(stColumnName) Then
```

Al especificar el mismo nombre del campo de texto en el documento que en de la columna del conjunto de datos, el contenido de la base de datos se transfiere al campo correspondiente del documento de texto. Se rellenan los todos datos mediante el bucle `Do While`.

```

        inIndex = oForm.findColumn(stColumnName)
        oTextField.Anchor.String = oForm.getString(inIndex)
    End If
End If
Loop
End Sub

```

Llamar a aplicaciones externas a LibreOffice

Mediante macros también es posible hacer llamadas a programas externos a LibreOffice para abrir archivos, >seguir los enlaces de Internet o iniciar un programa de correo electrónico. Para esta sección vea la base de datos de ejemplo [Example_Mail_File_activate.odb](#).

Llamada a aplicaciones para abrir archivos

Este procedimiento permite, con un solo clic en un botón, abrir el programa que está asociado al tipo de archivo en el sistema operativo. Se utiliza en el formulario *File_adtivate*

```

SUB File_activate
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oField AS OBJECT
    DIM oShell AS OBJECT
    DIM stField AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("form")
    oField = oForm.getByName("fileselection")
    REM Leer el nombre de archivo elegido por el control de selección
    stField = oField.Text
    REM Iniciar la llamada al programa mediante la conexión con el
    sistema operativo
    oShell =
createUnoService("com.sun.star.system.SystemShellExecute")
    stField = convertToUrl(stField)
    oShell.execute(stField,,0)
END SUB

```

En el ejemplo se utiliza el control de selección de archivo *fileselection* para elegir un archivo y un botón para iniciar el procedimiento.

Llamada a un programa en función del texto introducido por el usuario

Este procedimiento lanzará el programa asociado al tipo de archivo, el programa de correo electrónico o el navegador web en función del contenido del texto ingresado por el usuario. El siguiente procedimiento se utiliza en el formulario *Mail_Website_activate*

```

Sub Website_Mail_activate
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oForm As Object
    Dim oField As Object

```



```

Dim oShell As Object
Dim stField As String
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oForm = oDrawpage.Forms.getByName("form")
oField = oForm.getByName("url_mail")

```

Se lee el contenido del control *url_mail*. Puede ser una dirección web que comience, una dirección de correo electrónico o una ruta a un documento (por ejemplo, una imagen o un archivo almacenado externamente).

```

stFeld = oField.Text
If stField = "" Then
    Exit Sub
End If

```

Al pulsar el botón, si el control de cuadro de texto está vacío, el procedimiento se interrumpe, en caso de que tenga contenido se busca un carácter @ Esto indicaría una dirección de correo electrónico. El programa de correo electrónico debe iniciarse para enviar correo a esta dirección.

```

If InStr(stField,"@") Then
stField = "mailto:"+stField

```

Si no hay un carácter @, se busca el identificador de una URL. Si comienza con `http://` se trata de un recurso de Internet que debe buscarse con un navegador web. De lo contrario, la ruta comenzará con el término `file:///` que corresponde a un archivo en el sistema.

```

ElseIf InStr(stFeld,"http://") Then
    stFeld = convertToUrl(stField)
Else
    stField = "file:///"+stField
End If

```

Se busca el programa asignado por el sistema operativo a dichos archivos. Para la palabra clave `'mailto:'` es el programa de correo, para `'http://'` el navegador, en cualquier otro caso el sistema tiene que decidir en función del tipo de archivo y su configuración.

```

oShell =
createUnoService("com.sun.star.system.SystemShellExecute")
oShell.execute(stField,,0)
End Sub

```

Llamada a un programa de correo con contenido predefinido

El ejemplo anterior se puede ampliar para iniciar un programa de correo con un asunto y contenido predefinidos. El siguiente procedimiento se utiliza en el formulario *Mail_activate*

El programa de correo se inicia usando `'mailto:recipient?subject= &body= &cc= &bcc= '`. Las dos últimas entradas no están presentes en el formulario. Los archivos adjuntos no están previstos en la definición de `'mailto'` pero a veces funciona `'attachment='`.

```

Sub Mail_activate
Dim oDoc As Object
Dim oDrawpage As Object
Dim oForm As Object
Dim oField1 As Object

```



```

Dim oField2 As Object
Dim oField3 As Object
Dim oField4 As Object
Dim oShell As Object
Dim stField1 As String
Dim stField2 As String
Dim stField3 As String
Dim stField4 As String
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oForm = oDrawpage.Forms.getByName("form")
oField1 = oForm.getByName("mail_to")
oField2 = oForm.getByName("mail_subject")
oField3 = oForm.getByName("mail_body")
stField1 = oField1.Text
If stField1 = "" Then
    MsgBox "Sin dirección de correo electrónico" & Chr(13) & _
        "No se iniciará el programa de correo electrónico" , 48,
    "Enviar mensaje"
    Exit Sub
End If

```

La conversión a URL en el asunto (*mail_subject*) y el cuerpo del mensaje (*mail_body*) es necesaria para evitar que caracteres especiales y saltos de línea causen errores en el código. Sin embargo, >esta conversión añade `file:///` al principio del texto que se elimina con la función `Mid()`.

```

stField2 = Mid(ConvertToUrl(oField2.Text),9)
stField3 = Mid(ConvertToUrl(oField3.Text),9)

```

En contraste a una simple llamada a la ejecución de un programa, Los detalles de la invocación al correo se dan aquí como parte de la llamada de ejecución.

```

oShell =
createUnoService("com.sun.star.system.SystemShellExecute")
oShell.execute("mailto:" + stField1 + "?subject=" + stField2 +
"&body=" + stField3,,0)
End Sub

```



Nota

El envío de correo electrónico con la ayuda de un programa de correo también se puede hacer usando el siguiente código, pero el contenido real del correo electrónico no se puede insertar de esta manera.

```

Dim attaches(0)
oMailer =
createUnoService("com.sun.star.system.SimpleSystemMail")
oMailProgram = oMailer.querySimpleMailClient()
oNewmessage = oMailProgram.createSimpleMailMessage()
oNewmessage.setRecipient(stField1)
oNewmessage.setSubject(stField2)
attachs(0) = "file:///..."
oNewmessage.setAttachement(attachs())
oMailProgram.sendSimpleMailMessage(oNewmessage, 0 )

```

Para posibles parámetros, consulte:

http://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailMessage.html

Cambiar el puntero del ratón al situarse sobre un enlace

En el navegador de internet y en otros programas, al situar el puntero del ratón sobre un enlace externo el puntero del ratón se transforma en una mano, además el texto del enlace suele estar resaltado. Base también emplea esta característica, y su semejanza es perfecta. Cualquier usuario entenderá que puede hacer un clic para abrir un programa externo.

Para esta sección, consulte vea el formulario *Mail_Website_activate_directly* en la base de datos de ejemplo *Example_Mail_File_activate.odb*.

Este breve procedimiento debe estar vinculado al suceso `mouse_inside` de un control.

```

Sub mouse_pointer(Event As Object)
    REM Consulte también SwitchmousePointer en el módulo
    «ModuleControls» de la biblioteca de macros de LibreOffice «Tools»
    Dim oPointer As Object
    oPointer = createUnoService("com.sun.star.awt.Pointer")
    oPointer.setType(27) 'Tipos (consulte
com.sun.star.awt.SystemPointer en la API)
    Event.Source.Peer.SetPointer(oPointer)
End Sub

```

Mostrar formularios sin una barra de herramientas

Los nuevos usuarios de Base a menudo se irritan porque aparecen algunas barras de herramientas cuyo uso no es útil dentro de un formulario. Estas barras de herramientas se pueden ocultar de varias maneras. Las dos maneras, más comunes para ocultar una barra de herramientas en todas las versiones de LibreOffice, se describen a continuación.

Los tamaños de las ventanas y las barras de herramientas generalmente están controlados por una macro que se inicia desde un formulario usando **Herramientas > Personalizar > Sucesos > Abrir documento**. Esto se aplica a todo el documento, no a uno principal o subformulario individual.

Formularios sin barra de herramientas en la ventana

El tamaño de una ventana puede variar. Usando el botón apropiado también se puede cerrar. Estas tareas las lleva a cabo el gestor de ventanas o entorno de escritorio de su sistema operativo. La posición y el tamaño de una ventana en la pantalla se pueden establecer con una macro cuando se inicia el programa.

```

Sub Hide_toolbar

```

```

Dim oFrame As Object
Dim oWin As Object
Dim oLayoutMng As Object
Dim aElements()
oFrame = StarDesktop.GetCurrentFrame()

```

El título del formulario se mostrará en la barra de título de la ventana.

```

oFrame.setTitle "Mi Formulario"
oWin = oFrame.GetContainerWindow()

```

La ventana está maximizada. Esto no es lo mismo que el modo de pantalla completa, ya que la barra de tareas todavía está visible. La ventana tiene una barra de título, que se puede usar para cambiar su tamaño o cerrarla.

```

oWin.IsMaximized = true

```

Es posible crear una ventana con un tamaño y posición específicos. Esto se lleva a cabo con `oWin.SetPosSize(0, 0, 600, 400, 15)`. Aquí la ventana aparece en la esquina superior izquierda de la pantalla con un ancho de 600 píxeles y una altura de 400. El último número indica los valores introducidos que se tendrán en cuenta. Se llama `Flag`. Este valor se calcula a partir de la suma de los siguientes valores (internos de la API): `x = 1`, `y = 2`, `ancho = 4`, `alto = 8`. Como todos los parámetros >están expresados, `Flag` tiene que tener el valor 15 (1+2+4+8)

```

oLayoutMng = oFrame.LayoutManager
aElements = oLayoutMng.getElements()
For i = LBound(aElements) To UBound(aElements)
    If aElements(i).ResourceURL =
        "private:resource/toolbar/formsnavigationbar" Then
    Else
        oLayoutMng.HideElement(aElements(i).ResourceURL)
    End If
Next
End Sub

```

En el caso de una barra de navegación de formulario, no se debe ocultar, el formulario debe poderse usar en el caso en que no se haya incorporado un control de barra de navegación (lo que provocaría que la barra de navegación esté oculta de todos modos). Solo se deben ocultar las barras de herramientas que no sean la barra de navegación.

Si las barras de herramientas se ocultan, después de abandonar el formulario, seguirán ocultas. Por supuesto, pueden visualizarse usando **Ver > Barras de herramientas**. Pero sería bastante molesto si faltara la barra de herramientas estándar (**Ver > Barras de herramientas > Estándar**) o la barra de estado (**Ver > Barra de estado**).

Este procedimiento muestra (`ShowElement`) las barras de herramientas que estaban ocultas (`HideElement`). Los comentarios aluden a las barras cuya ausencia puede ser más notable.

```

Sub Show_toolbar
Dim oFrame As Object
Dim oLayoutMng As Object
Dim aElements()
oFrame = StarDesktop.GetCurrentFrame()
oLayoutMng = oFrame.LayoutManager
aElements = oLayoutMng.getElements()
For i = LBound(aElements) To UBound(aElements)

```

```

        oLayoutMng.showElement(aElements(i).ResourceURL)
    Next
    ' Barras importantes que pueden encontrarse ocultas:
    ' "private:resource/toolbar/standardbar"
    ' "private:resource/statusbar/statusbar"
End Sub

```

Las macros están vinculadas a: **Herramientas > Personalizar > Sucesos > Abrir documento** (*Hide_toolbar*) y **Cerrar documento** (*Show_Toolbar*).

A veces las barras de herramientas no vuelven. En algunos casos, puede ser más útil no utilizar las barras de herramientas > que el administrador de diseño ya conoce, sino primero crear barras de herramientas personalizadas y luego mostrarlas:

```

Sub Hide_toolbar
    Dim oFrame As Object
    Dim oLayoutMng As Object
    Dim i As Integer
    Dim aElements(5) As String
    oFrame = StarDesktop.getCurrentFrame()
    oLayoutMng = oFrame.LayoutManager
    aElements(0) = "private:resource/menubar/menubar"
    aElements(1) = "private:resource/statusbar/statusbar"
    aElements(2) = "private:resource/toolbar/formsnavigationbar"
    aElements(3) = "private:resource/toolbar/standardbar"
    aElements(4) = "private:resource/toolbar/formdesign"
    aElements(5) = "private:resource/toolbar/formcontrols"
    For Each i In aElemente()
        IF Not(oLayoutMng.requestElement(i)) Then
            oLayoutMng.createElement(i)
        End If
        oLayoutMng.showElement(i)
    Next i
End Sub

```

Las barras de herramientas que se crearán se nombran explícitamente. Si la barra de herramientas correspondiente no está disponible para el administrador de diseño, se crea usando `createElement` y luego se muestra usando `showElement`.

Formularios en modo de pantalla completa

En el modo de pantalla completa, el formulario cubre toda la pantalla. No hay barra de tareas u otros elementos que distraigan si se están ejecutando otros programas.

```

Function Fullscreen(boSwitch As Boolean)
    Dim oDispatcher As Object
    Dim Props(0) As New com.sun.star.beans.PropertyValue
    oDispatcher =
createUnoService("com.sun.star.frame.DispatchHelper")
    Props(0).Name = "FullScreen"
    Props(0).Value = boSwitch
    oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame,

```

```

        ".uno:FullScreen", "", 0, Props())
    End Function

```

Esta función se inicia utilizando el siguiente procedimiento (`Fullscreen_on`). En este procedimiento, se ejecuta simultáneamente el procedimiento anterior (`hide_toolbar`) para ocultar las barras de herramientas; de lo contrario, aparecerá la barra de herramientas, y al usarla se desactiva el modo de pantalla completa. En el modo de pantalla completa hay una barra de herramientas, aunque solo tiene un símbolo para volver al modo normal:

```

Sub Fullscreen_on
    Fullscreen(true)
    Hide_toolbar
End Sub

```

Se sale del modo de pantalla completa presionando la tecla **ESC**. También se puede usar un botón específico al que se asigne el siguiente procedimiento:

```

Sub Fullscreen_off
    Fullscreen(false)
    Show_toolbar
End Sub

```

Lanzar formularios directamente desde la apertura de la base de datos

Cuando las barras de herramientas desaparecen o se debe mostrar un formulario en modo de pantalla completa, el archivo de la base de datos debe iniciar el formulario directamente al abrirse. Desafortunadamente, un comando simple para abrir un formulario no funcionará, ya que la conexión de la base de datos aún no existe cuando se abre el archivo.

La siguiente macro se inicia desde **Herramientas > Personalizar > Sucesos > Abrir documento**. Use la opción *Guardar en...* `Databasefile.odt`.

```

Sub Form_Directstart
    Dim oDatasource As Object
    oDatasource = ThisDatabaseDocument.CurrentController
    If Not (oDatasource.isConnected()) Then
        oDatasource.connect()
    End If

    ThisDatabaseDocument.FormDocuments.getByName("Nombre_Formulario").open
End Sub

```

Primero se tiene que hacer una conexión a la base de datos. El controlador es parte de `ThisDatabaseDocument`, igual que el formulario. Tras hacer esta conexión, el formulario se puede iniciar y, al estar conectado, se tiene acceso a los datos.

Acceder a una base de datos MySQL con macros

Todas las macros mostradas hasta ahora han sido parte de una base de datos interna HSQLDB. Cuando se trabaja con bases de datos externas, son necesarios algunos cambios y extensiones.

Código MySQL en macros

Cuando se accede a la base de datos interna mediante macros las tablas y los campos tienen que estar entre comillas dobles duplicadas porque en Basic las comillas dobles se utilizan para delimitar un texto. Una instrucción en SQL:

```
SELECT "Field" FROM "Table"
```

Dentro de una macro tendrá el siguiente aspecto.:

```
stSQL = "SELECT ""Field"" FROM ""Table"""
```

Las consultas MySQL usan otra forma de señalar los campos y tablas: una comilla inclinada.

```
SELECT `Field` FROM `Database`.`Table`
```

Dentro del código Basic la comilla inclinada no necesita estar duplicada, pero sí hay que encerrar la instrucción entre comillas dobles, puesto que se trata de una cadena de texto. La instrucción MySQL anterior, en Basic debe escribirse así:

```
stMySQL = "SELECT `Field` FROM `Database`.`Table`"
```

Tablas temporales como almacenamiento intermedio individual

En el capítulo anterior, una tabla de un único registro se usaba con frecuencia para buscar o filtrar tablas. Esto no funcionará en un sistema multiusuario, ya que otros usuarios dependerían del valor del filtro de otra persona. Las tablas temporales en MySQL solo son accesibles para el usuario de la conexión activa, por lo que se pueden crear este tipo de tablas para buscar y filtrar.

Naturalmente, estas tablas no se pueden crear de antemano. Tienen que crearse cuando se abre el archivo Base. Por lo tanto, la siguiente macro debe estar vinculada a la apertura del archivo *.odb.

```
Sub CreateTempTable
    oDatasource = thisDatabaseDocument.CurrentController
    If Not (oDatasource.isConnected()) Then oDatasource.connect()
    oConnection = oDatasource.ActiveConnection()
    oSQL_Statement = oConnection.createStatement()
    stSql = "CREATE TEMPORARY TABLE IF NOT EXISTS `Searchtmp` (`ID`
INT PRIMARY KEY,
    `Name` VARCHAR(50))"
    oSQL_Statement.executeUpdate(stSql)
End Sub
```

Cuando se abre el archivo *.odb >no hay conexión a una base de datos MySQL externa. La conexión tiene que ser creada y posteriormente configurar una tabla temporal con los campos necesarios.

Diálogos

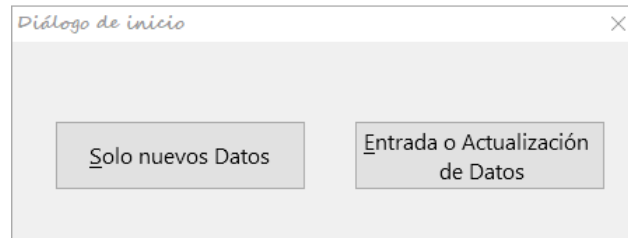
En Base, puede usar diálogos Basic en lugar de formularios para la entrada y modificación de datos o el mantenimiento de la base de datos. Los diálogos se pueden personalizar directamente para el entorno de aplicación en uso, pero, naturalmente, no se definen de manera tan cómoda como los formularios. Aquí se muestra una breve introducción que termina en un ejemplo bastante complicado para usar en el mantenimiento de la base de datos.

El ejemplo que utilizaremos se basa en la base de datos proporcionada `Example_Dialogs.odb` que contiene una tabla *name* con tres campos: *ID*, (clave primaria) *forename* (nombre) y *surname* (apellido)

Iniciar y finalizar diálogos

Primero, se debe crear el diálogo en la computadora en que está el archivo *.odb. Esto se hace usando **Herramientas > Macros > Organizar diálogos > Nombre_de_la_base_de_datos > Estándar > Nuevo**. El diálogo aparece con una superficie gris continua y una barra de título con un icono de cierre. El diálogo creado puede abrirse y luego cerrarse haciendo clic en el icono de cierre.

Cuando se selecciona el diálogo, existe la posibilidad de establecer su tamaño y posición en las propiedades generales. También se puede ingresar el contenido del título, *Diálogo de inicio* en este caso).



La barra de herramientas en el borde inferior de la ventana contiene varios controles para los diálogos. Desde ella se han seleccionado dos botones lo que permite iniciar distintos procedimientos u otros diálogos. La edición de contenido y la vinculación de macros a sucesos se hace igual que para los botones en los formularios. Las declaraciones de variables para los diálogos requiere un cuidado especial. El diálogo se declara como una variable global para que se pueda acceder a él mediante diferentes procedimientos. En este caso, el diálogo se llama oDialog0, porque en el ejemplo habrá más diálogos, aunque puede tener otro nombre.

Dim oDialog0 As Object

Primero se carga la biblioteca para el diálogo. Está en el directorio Estándar, si no se eligió otro nombre cuando se creó el diálogo. El diálogo en sí se puede alcanzar en esta biblioteca utilizando el nombre Dialog0. `Execute()` inicia el diálogo.

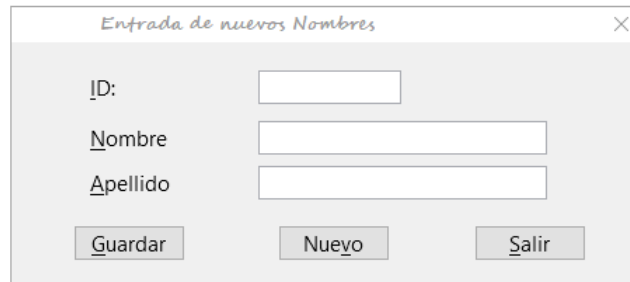
```
Sub Dialog0Start
    DialogLibraries.LoadLibrary("Standard")
    oDialog0 = createUnoDialog(DialogLibraries.Standard.Dialog0)
    oDialog0.Execute()
End Sub
```

En principio, un diálogo puede cerrarse usando el botón Cerrar en el marco. Sin embargo, si desea otro botón específico para esto, el comando `EndExecute()` debe usarse dentro de un procedimiento.

```
Sub Dialog0End
    oDialog0.EndExecute()
End Sub
```

Dentro de este marco, se puede iniciar y cerrar cualquier número de diálogos nuevamente.

Diálogo sencillo para ingresar nuevos registros



Este diálogo es un primer paso para el siguiente diálogo para editar registros. Primero se define el enfoque básico para administrar tablas. Aquí nos ocupamos del almacenamiento de registros con nuevas claves primarias o la nueva entrada completa de registros. ¿Hasta qué punto un pequeño diálogo como este puede ser suficiente para la entrada de dato en una base de datos? – depende de las necesidades del usuario –.

```
Dim oDialog1 As Object
```

Se debe crear una variable global para el diálogo en el nivel superior del módulo antes de cualquier procedimiento para que pueda ser accesible en cualquier momento.

El diálogo se abre y cierra de la misma manera que el diálogo anterior. Solo se cambia el nombre de Dialog0 a Dialog1. El botón *Salir* está vinculado al procedimiento para cerrar el diálogo.

El botón *Nuevo* se usa para borrar todos los controles del diálogo de entradas anteriores, utilizando el procedimiento `DataFieldsClear`.

```
Sub DataFieldsClear
    oDialog1.getControl("NumericField1").Text = ""
    oDialog1.getControl("TextField1").Text = ""
    oDialog1.getControl("TextField2").Text = ""
End Sub
```

Se puede acceder a cada control que se ha insertado en un diálogo por su nombre. La interfaz de Basic evita que los nombres estén duplicados, cosa que no ocurre con los controles en un formulario.

El método `getControl` se usa con el nombre del control. Los campos numéricos también tienen una propiedad `Text`, usando esta propiedad es la única forma en que se puede vaciar un campo numérico (existe texto vacío, pero no un número vacío). Se debe escribir un 0 en el campo de la clave primaria.

El botón **Guardar** inicia el procedimiento `Data1Save`:

```
Sub Data1Save
    Dim oDatasource As Object
    Dim oConnection As Object
    Dim oSQL_Command As Object
    Dim loID As Long
    Dim stForename As String
    Dim stSurname As String
    loID = oDialog1.getControl("NumericField1").Value
    stForename = oDialog1.getControl("TextField1").Text
    stSurname = oDialog1.getControl("TextField2").Text
    If loID > 0 And stSurname <> "" Then
        oDatasource = thisDatabaseDocument.CurrentController
```



```

If Not (oDatasource.isConnected()) Then
    oDatasource.connect()
End If
oConnection = oDatasource.ActiveConnection()
oSQL_Command = oConnection.createStatement()
stSql = "SELECT ""ID"" FROM ""name"" WHERE ""ID"" = '"+loID+'"'
oResult = oSQL_Command.executeQuery(stSql)
While oResult.next
    MsgBox ("Ya existe el valor para el campo 'ID'",16,"Valor
Duplicado")
    Exit Sub
Wend
stSql = "INSERT INTO ""name"" (""ID"", ""forename"",
""surname""
    VALUES ('"+loID+'', '"+stForename+'', '"+stSurname+'')"
oSQL_Command.executeUpdate(stSql)
DatafieldsClear
End If
End Sub

```

Como en el procedimiento `DatafieldsClear`, se accede a los campos de entrada. Esta vez el acceso es para lectura. Solo si el campo `ID` tiene una entrada mayor que 0 y el campo `surname` también contiene texto, se pasará el registro. Se puede excluir un valor nulo para el `ID` porque una variable numérica para números enteros siempre se inicializa a 0. Por lo tanto, un campo vacío se almacena con un valor cero.

Si ambos campos han recibido contenido, se establece una conexión con la base de datos. Como los controles no están en un formulario, la conexión de la base de datos debe realizarse utilizando `thisDatabaseDocument.CurrentController`.

Primero se consulta la base de datos para ver si ya existe un registro con la misma clave primaria que la ingresada. Si esto ocurre, se muestra un diálogo de mensaje que contiene un símbolo de *Stop* (código: 16) el mensaje *Ya existe valor para el campo 'ID' >y con título Valor Duplicado*. Al pulsar *Aceptar* el procedimiento se detiene con la instrucción `Exit SUB`.

Si la consulta no encuentra ningún registro con la misma clave primaria, el nuevo registro se inserta en la base de datos utilizando la orden SQL de inserción. Luego se llama a la rutina `DatafieldsClear` para limpiar el contenido de los controles y poder continuar ingresando datos.

Diálogo para editar registros en una tabla

Este diálogo ofrece más posibilidades que el anterior. Aquí se pueden mostrar todos los registros y puede navegar por ellos, crear o eliminar registros. Naturalmente, el código es más complicado.

El botón Salir está vinculado al procedimiento que se describió en el diálogo anterior, modificado para *Dialog2*. Aquí se describen los botones restantes y sus funciones. La entrada de datos en el diálogo está restringida porque el campo *ID* tiene que tener un valor mínimo de 1. Esta limitación está producida por el manejo de variables en Basic:

Las variables numéricas se inicializan de manera predeterminada en 0. Por lo tanto, si los valores numéricos de los campos vacíos y los de los campos que contienen un 0 se leen, Basic no puede detectar diferencias entre ellos. Esto significa que si se usara un 0 en el campo *ID*, tendría que leerse primero como texto y quizás convertirse más tarde a un número.

El diálogo se crea en las mismas condiciones que antes. Sin embargo, el procedimiento de carga depende de un valor cero para la variable pasada al procedimiento *DataLoad*.

```
Sub Dialog2Start
    DialogLibraries.LoadLibrary("Standard")
    oDialog2 = createUnoDialog(DialogLibraries.Standard.Dialog2)
    DataLoad(0)
End Sub
```

Con este procedimiento se crea el diálogo, y se hace la llamada al procedimiento *DataLoad()* para conectar con la base de datos y cargar los valores en los controles. Una vez cargados los valores se presentará el diálogo al usuario.

```
Sub DataLoad(loID As Long)
    Dim oDatasource As Object
    Dim oConnection As Object
    Dim oSQL_Command As Object
    Dim stForename As String
    Dim stSurname As String
    Dim loRow As Long
    Dim loRowMax As Long
    Dim inStart As Integer
    oDatasource = thisDatabaseDocument.CurrentController
    If Not (oDatasource.isConnected()) Then
        oDatasource.connect()
    End If
    oConnection = oDatasource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    If loID < 1 Then
        stSql = "SELECT MIN(""ID"") FROM ""name""
        oResult = oSQL_Command.executeQuery(stSql)
        While oResult.next
            loID = oResult.getInt(1)
        Wend
        inStart = 1
    End If
```

Las variables se declaran. La conexión de la base de datos para el diálogo se establece como se describe anteriormente. Al principio, *loID* es 0. Este caso proporciona el valor de clave primaria más bajo permitido por SQL. El registro correspondiente se mostrará más tarde en el diálogo. Al mismo tiempo, la variable *inStart* se establece en 1, de modo que el diálogo se pueda iniciar más tarde. Si la tabla no contiene ningún registro, *loID* seguirá siendo 0. En ese caso, no será necesario buscar el número y el contenido de los registros correspondientes.

Solo si loID es mayor que 0, se hará una consulta para ver qué registros están disponibles en la base de datos. Luego, una segunda consulta contará todos los registros que se mostrarán. La tercera consulta proporciona la posición del registro en uso contando todos los registros que tengan un número en la clave primaria menor o igual al número introducido.

```

If loID > 0 Then
    stSql = "SELECT * FROM ""name"" WHERE ""ID"" = '"+loID+'"'
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        loID = oResult.getInt(1)
        stForename = oResult.getString(2)
        stSurname = oResult.getString(3)
    Wend
    stSql = "SELECT COUNT(""ID"") FROM ""name""'"
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        loRowMax = oResult.getInt(1)
    Wend
    stSql = "SELECT COUNT(""ID"") FROM ""name"" WHERE ""ID"" <=
'+loID+'"'
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        loRow = oResult.getInt(1)
    Wend
    oDialog2.getControl("NumericField1").Value = loID
    oDialog2.getControl("TextField1").Text = stForename
    oDialog2.getControl("TextField2").Text = stSurname
End If
oDialog2.getControl("NumericField2").Value = loRow
oDialog2.getControl("NumericField3").Value = loRowMax
If loRow = 1 Then
    ' Fila (registro) anterior
    oDialog2.getControl("CommandButton4").Model.enabled = False
Else
    oDialog2.getControl("CommandButton4").Model.enabled = True
End If
If loRow <= loRowMax Then
    ' Fila (registro) siguiente| nueva fila | borrar
    oDialog2.getControl("CommandButton5").Model.enabled = True
    oDialog2.getControl("CommandButton2").Model.enabled = True
    oDialog2.getControl("CommandButton6").Model.enabled = True
Else
    oDialog2.getControl("CommandButton5").Model.enabled = False
    oDialog2.getControl("CommandButton2").Model.enabled = False
    oDialog2.getControl("CommandButton6").Model.enabled = False
End If
IF inStart = 1 Then
    oDialog2.Execute()

```

```
End If
End Sub
```

Los valores obtenidos se transfieren a los controles del diálogo. Las entradas para el número de registro actual y el número total de registros recuperados siempre se escriben, reemplazando el valor numérico predeterminado de 0.

Los botones de navegación (*CommandButton4* y *CommandButton5*) solo se pueden usar cuando se puede acceder al registro correspondiente. En caso contrario, se desactivan temporalmente con `enabled = False`. Lo mismo ocurre para los botones *Nuevo* y *Borrar*. No deberían estar disponibles cuando el número de registro mostrado sea mayor que el número máximo que se determinó. Esta es la configuración predeterminada de este diálogo al ingresar registros.

Una vez cargados los valores necesarios para los controles se presenta el diálogo al usuario con la instrucción `oDialog2.Execute`.

El botón `<` (*CommandButton4*) se usa para retroceder al registro anterior. Por lo tanto, este botón debe estar desactivado cuando el registro que se muestre sea el primero.

La navegación mediante este botón requiere que la clave primaria para el registro en curso se lea desde el control de cuadro de texto con etiqueta *ID* (*NumericField1*) (donde se refleja el registro en curso). Aquí hay dos casos posibles:

- 1) El control no tiene ningún valor, En este caso `loID` tiene el valor predeterminado suministrado por Basic a la variable de tipo integer (0)
- 2) O por el contrario, `loID` contiene un valor mayor que 0. Luego, una consulta puede determinar el valor de *ID* directamente inferior al `>actual`.

```
Sub PreviousRow
    Dim loID As Long
    Dim loIDnew As Long
    loID = oDialog2.getControl("NumericField1").Value
    oDatasource = thisDatabaseDocument.CurrentController
    If Not (oDatasource.isConnected()) Then
        oDatasource.connect()
    End If
    oConnection = oDatasource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    If loID < 1 Then
        stSql = "SELECT MAX(""ID"") FROM ""name""
    Else
        stSql = "SELECT MAX(""ID"") FROM ""name"" WHERE ""ID"" <
""+loID+""
    End If
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        loIDnew = oResult.getInt(1)
    Wend
    If loIDnew > 0 Then
        DataLoad(loIDnew)
    End If
End Sub
```

Si el control *ID* está vacío, la navegación por registros debería cambiar al valor más alto del número de clave primaria. Si, por otro lado, el control *>ID* se refiere a un registro, debe mostrar el valor anterior de *ID*.

El resultado de esta consulta se utiliza para ejecutar el procedimiento `DataLoad` nuevamente con el valor de la clave primaria correspondiente.

El botón `>` (`CommandButton5`) se usa para navegar al siguiente registro. Esta posibilidad debería existir solo cuando los datos del diálogo no se hayan vaciado para la entrada de un nuevo registro. Naturalmente, este será el caso cuando se inicie el diálogo y también con una tabla vacía.

Un valor en `NumericField1` es obligatorio. A partir de este valor, SQL puede determinar qué clave primaria es la siguiente más alta en la tabla. Si el conjunto de resultados de la consulta está vacío porque no hay un registro correspondiente, el valor de `loIDnew = 0`. De lo contrario, el contenido del siguiente registro se lee usando `DataLoad`.

```
Sub NextRow
    Dim loID As Long
    Dim loIDnew As Long
    loID = oDialog2.getControl("NumericField1").Value
    oDatasource = thisDatabaseDocument.CurrentController
    If Not (oDatasource.isConnected()) Then
        oDatasource.connect()
    End If
    oConnection = oDatasource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    stSql = "SELECT MIN(""ID"") FROM ""name"" WHERE ""ID"" >
    '"+loID+'""
    oResult = oSQL_Command.executeQuery(stSql)
    While oResult.next
        loIDnew = oResult.getInt(1)
    Wend
    If loIDnew > 0 Then
        DataLoad(loIDnew)
    Else
        Datafields2Clear
    End If
End Sub
```

Si se ha alcanzado el registro más alto, al navegar al siguiente registro, la tecla de navegación inicia el procedimiento `Datafields2Clear`, que sirve para preparar la entrada de un nuevo registro.

Este procedimiento no solo vacía los datos de los controles. El número del nuevo registro presentado en la zona de navegación se obtiene sumando un uno al número total de registros, dejando en claro que el registro en el que se está trabajando actualmente aún no está incluido en la base de datos.

Tan pronto como se haya lanzado `Datafields2Clear`, se activa la posibilidad de saltar al registro anterior, se salta al siguiente registro y se desactivan los botones *Nuevo* y *Borrar*, puesto que el diálogo ya está preparado para crear un nuevo registro y no se puede borrar un registro que todavía no existe.

```
Sub Datafields2Clear
    loRowMax = oDialog2.getControl("NumericField3").Value
```

```

oDialog2.getControl("NumericField1").Text = ""
oDialog2.getControl("TextField1").Text = ""
oDialog2.getControl("TextField2").Text = ""
oDialog2.getControl("NumericField2").Value = loRowMax + 1
oDialog2.getControl("CommandButton4").Model.enabled = True '
Registro anterior
oDialog2.getControl("CommandButton5").Model.enabled = False '
Registro siguiente
oDialog2.getControl("CommandButton2").Model.enabled = False '
Registro nuevo
oDialog2.getControl("CommandButton6").Model.enabled = False '
Borrar
End Sub

```

Guardar registros solo debería ser posible cuando los campos *ID* y *surname* contengan entradas. Si se cumple esta condición, el procedimiento comprueba si se trata de un nuevo registro haciendo uso del puntero de registro que está configurado para que el nuevo registro sean uno más alto que el número total de registros.

Para nuevos registros, se realizan comprobaciones para garantizar que se realice correctamente la operación de guardado. Si el número utilizado para la clave primaria se ha utilizado antes, se muestra una advertencia. Si la pregunta asociada se responde con el botón *Aceptar*, se sobrescribe el registro existente con este número. De lo contrario, se utilizará el número existente. Si no hay entradas existentes en la base de datos (`loRowMax = 0`), esta prueba es innecesaria y el nuevo registro se puede guardar directamente. Para un nuevo registro, el número de registros se incrementa en 1 y los controles se limpian para el siguiente registro.

Los registros existentes simplemente se sobrescriben con un comando de actualización.

```

Sub Data2Save(oEvent As Object)
Dim oDatasource As Object
Dim oConnection As Object
Dim oSQL_Command As Object
Dim oDlg As Object
Dim loID As Long
Dim stForename As String
Dim stSurname As String
Dim inMsg As Integer
Dim loRow As Long
Dim loRowMax As Long
Dim stSql As String
oDlg = oEvent.Source.getContext()
loID = oDlg.getControl("NumericField1").Value
stForename = oDlg.getControl("TextField1").Text
stSurname = oDlg.getControl("TextField2").Text
If loID > 0 And stSurname <> "" Then
oDatasource = thisDatabaseDocument.CurrentController
If Not (oDatasource.isConnected()) Then
oDatasource.connect()
End If
oConnection = oDatasource.ActiveConnection()

```

```

oSQL_Command = oConnection.createStatement()
loRow = oDlg.getControl("NumericField2").Value
loRowMax = oDlg.getControl("NumericField3").Value
If loRowMax < loRow Then
    If loRowMax > 0 Then
        stSql = "SELECT ""ID"" FROM ""name"" WHERE ""ID"" =
''+loID+''"
        oResult = oSQL_Command.executeQuery(stSql)
        While oResult.next
            inMsg = MsgBox ("Ya existe el valor para el campo 'ID'." &
CHR(13) & "¿Desea sobrescribir el registro?",20,"Valor Duplicado")
            If inMsg = 6 Then
                stSql = "UPDATE ""name"" SET
""forename""='"+stForename+"',
""surname""='"+stSurname+" WHERE ""ID"" =
''+loID+''"
                oSQL_Command.executeUpdate(stSql)
                DataLoad(loID) 'el contador de registros (Rowcount) se
debe actualizar.
            End If
        Exit Sub
    Wend
End If
stSql = "INSERT INTO ""name"" (""ID"", ""forename"",
""surname"") VALUES
('"+loID+"', '"+stForename+"', '"+stSurname+"')
oSQL_Command.executeUpdate(stSql)
oDlg.getControl("NumericField3").Value = loRowMax + 1
' Después de la inserción se suma un 1 al número de
registros.
Datafields2Clear
' Se limpian los datos de los controles para la siguiente
entrada.
Else
    stSql = "UPDATE ""name"" SET ""forename""='"+stForename+"',
""surname""='"+stSurname+" WHERE ""ID"" = ''+loID+''"
    oSQL_Command.executeUpdate(stSql)
End If
End If
End Sub

```

El procedimiento de borrado utiliza una pregunta adicional para evitar eliminaciones accidentales. Dado que el botón está desactivado cuando los campos de entrada están vacíos, nunca debe aparecer un *NumericField1* vacío. Por lo tanto, se puede omitir la verificación `IF loID > 0`.

La eliminación hace que el número de registros disminuya en 1. Esto debe corregirse usando `loRowMax - 1`. Luego se muestra el registro siguiente al que estaba en uso.

```

Sub DataDelete(oEvent As Object)
    Dim oDatasource As Object

```

```

Dim oConnection As Object
Dim oSQL_Command As Object
Dim oDlg As Object
Dim loID As Long
oDlg = oEvent.Source.getContext()
loID = oDlg.getControl("NumericField1").Value
If loID > 0 Then
    inMsg = MsgBox ("¿Desea borrar el registro actual?",20,"Borrar
registro")
    If inMsg = 6 Then
        oDatasource = thisDatabaseDocument.CurrentController
        If Not (oDatasource.isConnected()) Then
            oDatasource.connect()
        End If
        oConnection = oDatasource.ActiveConnection()
        oSQL_Command = oConnection.createStatement()
        stSql = "DELETE FROM ""name"" WHERE ""ID"" = '"+loID+'"'
        oSQL_Command.executeUpdate(stSql)
        loRowMax = oDlg.getControl("NumericField3").Value
        oDlg.getControl("NumericField3").Value = loRowMax - 1
        NextRow
    End If
ELSE
    MsgBox ("No se ha borrado ningún registro." & CHR(13) &
        "No se ha seleccionado ningún registro.",64,"No se puede
borrar")
    End If
End Sub

```

Este pequeño diálogo ha demostrado que el uso de código macro puede proporcionar una base para procesar registros. El acceso a través de formularios es mucho más fácil, pero un diálogo puede ser muy flexible para adaptarse a los requisitos del programa. Sin embargo, no es adecuado para la creación rápida de una interfaz de base de datos.

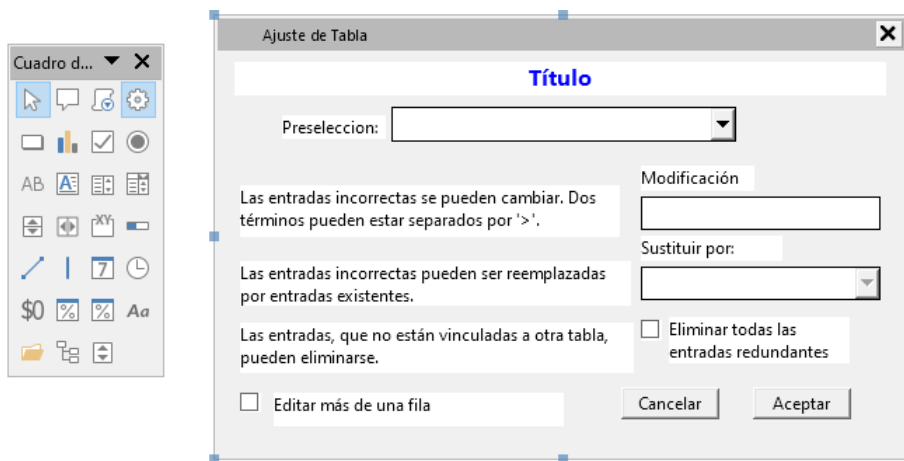
Usar un diálogo para limpiar entradas incorrectas en tablas

Este diálogo más complejo *Dialog_TableAdjustment* se encuentra en la base de datos de ejemplo *Media_with_macros.odt*

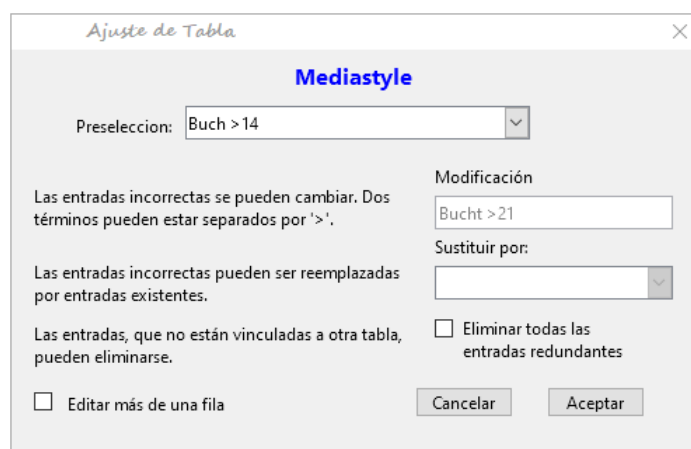
Los errores de entrada en los campos a menudo solo se notan más tarde. A menudo es necesario modificar entradas duplicada en varios registros al mismo tiempo. Es incómodo tener que hacer esto en la vista de tabla normal, especialmente cuando se tienen que editar varios registros, ya que cada registro requiere que se realice una entrada individual.

Los formularios pueden usar macros para hacer este tipo de cosas, pero para hacerlo en varias tablas, necesitaría formularios construidos de manera idéntica.

Los diálogos pueden hacer el trabajo de manera más eficaz. Al principio se puede proporcionar un diálogo con los datos necesarios para las tablas apropiadas y se puede invocar desde distintos formularios.



Los diálogos se guardan junto con los módulos para macros. Su creación es básicamente igual a la de un formulario. Los controles son muy similares, excepto el control de tabla que está presente en formularios pero no existe en los diálogos de Basic.



La apariencia de los diálogos está determinada por la configuración de la interfaz gráfica de usuario.

El diálogo que se muestra arriba sirve en la base de datos de ejemplo para editar tablas que no se utilizan directamente como base de un formulario.

Por ejemplo, solo se puede acceder al tipo de artículo a través de un listado (en la versión macro se utiliza un cuadro combinado). En la versión macro, el contenido del campo puede expandirse con nuevo contenido, pero no es posible alterar el contenido existente. En la versión sin macros, las modificaciones se llevan a cabo utilizando un control de tabla separado.

Si bien las alteraciones en este caso son fáciles de realizar sin macros, es bastante difícil cambiar el tipo de artículo de muchos artículos a la vez. Supongamos que están disponibles los siguientes tipos:

"Libro, tapa blanda", "Libro, tapa dura", "Libro en rústica" y "Libro espiral doble".

Ahora resulta que, después de que la base de datos ha estado en uso durante mucho tiempo, nos damos cuenta que tantos tipos de artículos han vuelto excesiva la tarea de ordenación. Por lo tanto, deseamos reducirlos, preferiblemente a un solo tipo. Sin macros, los registros en la tabla de artículos tendrían que ser encontrados (usando un filtro) y alterados individualmente. Si conoce SQL, puede hacerlo mucho mejor usando una orden SQL. Puede cambiar todos los registros en la tabla *Media* con una sola entrada. Una segunda orden SQL elimina los tipos de medios ya sobrantes que no tienen ningún enlace a la tabla de Medios.

Precisamente, este método se aplica utilizando el diálogo anterior con el control *>Sustituir por* : solo hay que adaptar una orden SQL a la tabla *Mediastyle* utilizando un procedimiento que también puede usarse en otras tablas.

A menudo, las entradas se trasladan a una tabla que, posteriormente, se puede cambiar en el formulario, por lo que ya no son necesarias. Eliminar estas entradas huérfanas puede ser beneficioso, pero son bastante difíciles de encontrar utilizando la interfaz gráfica de usuario. Aquí, de nuevo, es útil una orden SQL adecuada, junto con una instrucción de eliminación. Esta orden para las tablas afectadas se incluye en el diálogo con *Eliminar todas las entradas redundantes*.

Si el diálogo se va a utilizar para llevar a cabo varios cambios, se indica mediante la casilla de verificación *Editar más de una fila*. Entonces el diálogo no terminará simplemente cuando se haga clic en el botón *Aceptar*.

El código de macro para este diálogo se puede ver en su totalidad en la base de datos de ejemplo. Solo algunos extractos se explican a continuación.

```
Sub Table_purge(oEvent As Object)
```

La macro debe iniciarse ingresando en la sección Información adicional para los botones relevantes:

```
0: Form, 1: Subform, 2: SubSubform, 3: Combobox or table control, 4:  
Foreign key field in a form, empty for a table control, 5: Table name of  
auxiliary table, 6: Table field1 of auxiliary table, 7: Table field2 of  
auxiliary table, or 8: Table name of auxiliary table for table field2
```

Las entradas en esta área se enumeran al comienzo de la macro como comentarios. Los números vinculados a ellos se transfieren y la entrada relevante se lee desde una matriz. La macro puede editar listados, que tienen dos entradas, separadas por el signo mayor que (>). Estas dos entradas también pueden provenir de tablas diferentes y reunirse mediante una consulta, como por ejemplo en la tabla de Código postal, que solo tiene el campo de clave foránea *Town_ID* para la ciudad, que requiere que la tabla *Town* muestre los nombres de las ciudades.

```
Dim aForeignTable(0, 0 to 1)  
Dim aForeignTable2(0, 0 to 1)
```

Entre las variables definidas al principio hay dos matrices. Si bien el comando `Split()` puede crear matrices normales durante la ejecución del procedimiento, las matrices bidimensionales tienen que definirse de antemano. Las matrices bidimensionales son necesarias para almacenar varios registros de una consulta cuando la consulta se refiere a más de un campo. Las dos matrices declaradas anteriormente tienen que poder interpretar las consultas que hacen referencia a dos campos de tabla. Por lo tanto, se definen para dos contenidos diferentes utilizando 0 a 1 para la segunda dimensión.

```
stTag = oEvent.Source.Model.Tag  
aTable() = Split(stTag, ", ")  
For i = LBound(aTable()) To UBound(aTable())  
    aTable(i) = trim(aTable(i))  
Next
```

Se leen las variables proporcionadas. La secuencia es la establecida en el comentario anterior. Hay un máximo de nueve entradas y debe declarar si existe una octava entrada para el campo de `table2` y una novena entrada para una segunda tabla.

Si los valores se van a eliminar de una tabla, primero es necesario verificar que no existan como claves foráneas en alguna otra tabla. En estructuras de tabla simples, una tabla dada tendrá solo una conexión de clave foránea a otra tabla. Sin embargo, en la base de datos de ejemplo >hay una tabla *Town* que se usa tanto para el lugar de publicación de los artículos como para las

direcciones de la ciudad. Por lo tanto, la clave primaria de la tabla *Town* se ingresa dos veces en diferentes tablas.

Estas tablas y nombres de claves foráneas también se pueden ingresar naturalmente usando el campo *Información adicional*. Sin embargo, sería mejor si se pudieran proporcionar universalmente para todos los casos. Esto se puede hacer usando la siguiente consulta.

```
stSql = "SELECT ""FKTABLE_NAME"", ""FKCOLUMN_NAME"" FROM  
""INFORMATION_SCHEMA"". ""SYSTEM_CROSSREFERENCE"" WHERE  
""PKTABLE_NAME"" = '" + aTable(5) + "'"
```

En la base de datos, el área `INFORMATION_SCHEMA` contiene toda la información sobre las tablas de la base de datos, incluida información sobre claves foráneas. Se puede acceder a las tablas que contienen esta información utilizando `INFORMATION_SCHEMA`.

`SYSTEM_CROSSREFERENCE.PKTABLE_NAME` proporciona la tabla que proporciona su clave primaria para la conexión. `FKTABLE_NAME` proporciona la tabla que utiliza esta clave primaria como clave foránea. Finalmente, `FKCOLUMN_NAME` proporciona el nombre del campo de clave foránea.

La tabla que proporciona su clave primaria para usar como clave foránea se encuentra en la matriz creada previamente en la posición 6. Al recuento que comienza con 0, el valor se lee de la matriz usando `aTable(5)`.

```
inCount = 0  
stForeignIDTab1Tab2 = "ID"  
stForeignIDTab2Tab1 = "ID"  
stAuxiltable = aTable(5)
```

Antes de que comience la lectura de las matrices, se tienen que establecer algunos valores predeterminados. Estos son: el índice de la matriz en la que se escribirán los valores de la tabla auxiliar, la clave primaria predeterminada si no necesitamos la clave foránea para una segunda tabla y la tabla auxiliar predeterminada, vinculada a la tabla principal, para el código postal y ciudad, la tabla de códigos postales.

Cuando dos campos están vinculados para su visualización en un listado, pueden proceder de dos tablas diferentes. Para la visualización del código postal y la ciudad, la consulta es:

```
SELECT "Postcode"."Postcode" || ' > ' || "Town"."Town" FROM "Postcode", "Town" WHERE  
"Postcode"."Town_ID" = "Town"."ID"
```

La tabla para el primer campo (*Postcode*) está vinculada a la segunda tabla mediante una clave foránea. Solo la información de estas dos tablas y los campos *Postcode* y *Town* se pasa a la macro. Todas las claves primarias se llaman por defecto *ID* en la base de datos de ejemplo. Por lo tanto, la clave foránea de *Town* en el código postal tiene que determinarse utilizando el procedimiento.

Del mismo modo, el procedimiento tiene que acceder a cada tabla con la que el contenido del listado está conectado por una clave foránea.

```
oQuery_result = oSQL_Statement.executeQuery(stSql)  
If Not IsNull(oQuery_result) Then  
While oQuery_result.next  
ReDim Preserve aForeignTable(inCount,0 to 1)
```

La matriz tiene que estar recién dimensionada cada vez. Para mantener los contenidos existentes, se utiliza la instrucción `Preserve`.

```
aForeignTables(inCount,0) = oQuery_result.getString(1)
```

Lectura del primer campo con el nombre de la tabla que contiene la clave foránea. El resultado para la tabla de códigos postales es la tabla *Address*.

```
aForeignTables(inCount,1) = oQuery_result.getString(2)
```

Lectura del segundo campo con el nombre del campo de clave foránea. El resultado para la tabla de código postal es el campo `Postcode_ID` en la tabla *Address*.

En los casos en que una llamada al procedimiento incluye el nombre de una segunda tabla, se ejecuta el siguiente bucle. Solo cuando el nombre de la segunda tabla aparece como la tabla de clave foránea para la primera tabla, se cambia la entrada predeterminada. En nuestro caso, esto no ocurre, ya que la tabla *Town* no tiene clave foránea de la tabla de *Postcode*. Por lo tanto, la entrada predeterminada para la tabla auxiliar sigue siendo *Postcode*; finalmente, la combinación de código postal y ciudad es la base de la tabla de direcciones, que contiene una clave foránea de la tabla de códigos postales.

```
If UBound(aTable()) = 8 Then
  If aTable(8) = aForeignTable(inCount,0) Then
    stForeignIDTab2Tab1 = aForeignTable(inCount,1)
    stAuxiltable = aTable(8)
  End If
End If
inCount = inCount + 1
```

Como es posible que sea necesario leer más valores, el índice se incrementa para redimensionar las matrices y el ciclo termina.

```
Wend
End If
```

Si, cuando se llama al procedimiento, existe un segundo nombre de tabla, se inicia la misma consulta para esta tabla:

```
If UBound(aTable()) = 8 Then
```

Se ejecuta de manera idéntica, excepto que el bucle prueba si quizás el primer nombre de la tabla aparece como un nombre de tabla de clave foránea. Ese es el caso aquí: la tabla *Postcode1* contiene la clave foránea *Town_ID* de la tabla *Town*. Esta clave foránea ahora se asigna a la variable `stForeignIDTab1Tab2`, de modo que se puede definir la relación entre las tablas.

```
If aTable(5) = aForeignTable2(inCount,0) Then
  stForeignIDTab1Tab2 = aForeignTable2(inCount,1)
End If
```

Después de algunas configuraciones adicionales para garantizar un retorno al formulario correcto después de ejecutar el diálogo (determinando el número de línea del formulario, para que podamos volver a ese número de línea después de una nueva lectura), comienza el bucle, que recrea el diálogo cuando se completa la primera acción pero se requiere que el diálogo se mantenga abierto para acciones adicionales. La configuración para la repetición se lleva a cabo utilizando la casilla de verificación correspondiente.

```
Do
```

Antes de que se inicie el diálogo, primero se determina el contenido de los listados. Se debe tener cuidado si los listados representan dos campos de tabla o incluso están relacionados con dos tablas diferentes.

```
If UBound(aTable()) = 6 Then
```

El listado se relaciona solo con una tabla y un campo, ya que la matriz de argumentos termina en `Tablefield1` de la tabla auxiliar.

```
stSql = "SELECT "" + aTable(6) + "" FROM "" + aTable(5) +  
"" ORDER BY "" + aTable(6) + """  
ElseIf UBound(aTable()) = 7 Then
```

El listado se relaciona con dos campos de tabla pero solo con una tabla, ya que la matriz de argumentos termina en `Tablefield2` de la tabla auxiliar.

```
stSql = "SELECT "" + aTable(6) + ""||' > '||"" + aTable(7)  
+ "" FROM "" + aTable(5) + "" ORDER BY "" + aTable(6) + """  
Else
```

El listado se basa en dos campos de tabla de dos tablas. Esta consulta corresponde al ejemplo con el código postal y la ciudad.

```
stSql = "SELECT "" + aTable(5) + "".""" + aTable(6) + ""||'  
> '||"" + aTable(8) + "".""" + aTable(7) + "" FROM "" +  
aTable(5) + """, "" + aTable(8) + "" WHERE "" + aTable(8) +  
"".""" + stForeignIDTab2Tab1 + "" = "" + aTable(5) + "".""" +  
stForeignIDTab1Tab2 + "" ORDER BY "" + aTable(6) + """  
End If
```

Tenemos la primera evaluación para determinar las claves foráneas. Las variables `stForeignIDTab2Tab1` y `stForeignIDTab1Tab2` comienzan con el valor `ID`. Para `stForeignIDTab1Tab2`, la evaluación de la consulta anterior arroja un valor diferente, es decir, el valor de `Town_ID`. De esta manera, la construcción de la consulta anterior produce exactamente el contenido ya formulado para el código postal y la ciudad, solo mejorado por la ordenación.

Ahora se tiene que hacer enlace con los listados, para proporcionarles el contenido devuelto por las consultas. Estos listados aún no existen, ya que el diálogo en sí aún no se ha creado. El diálogo se crea primero en la memoria, usando las siguientes líneas, antes de que se visualice en la pantalla.

```
DialogLibraries.LoadLibrary("Standard")  
oDlg =  
CreateUnoDialog(DialogLibraries.Standard.Dialog_Table_purge)
```

Luego vienen las configuraciones para los campos del diálogo. Aquí, por ejemplo, está listado que se proporcionará con los resultados de la consulta anterior:

```
oCtlList1 = oDlg.GetControl("ListBox1")  
oCtlList1.addItem(aContent(),0)
```

El acceso a los controles del diálogo se logra utilizando `GetControl` con el nombre apropiado. En los diálogos no es posible que dos controles utilicen el mismo nombre, ya que esto crearía problemas al evaluar el diálogo.

El listado se suministra con el contenido de la consulta, que se ha almacenado en la matriz `aContent()`. El listado contiene solo el contenido que se mostrará como un campo, por lo que solo se ocupará la posición 0.

Después de que se hayan completado todos los campos con el contenido deseado, se inicia el diálogo.

```
Select Case oDlg.Execute()  
Case 1 'Case 1 significa que se ha pulsado el botón "Aceptar"  
Case 0 'Si se ha pulsado el botón "Cancelar"
```

```

    inRepetition = 0
End Select
Loop While inRepetition = 1

```

El diálogo se ejecuta repetidamente siempre que el valor de `inRepetition` sea 1. Esto se establece mediante la casilla de verificación correspondiente.

Aquí, en resumen, está el contenido después de hacer clic en el botón *Aceptar*:

```

Case 1
    stInhalt1 = oCtlList1.getSelectedItem() 'Lee valor de
Listbox1 ...
    REM ... y determina el valor de ID correspondiente.

```

El valor de ID del primer listado se almacena en la variable `inLB1`.

```

    stText = oCtlText.Text ' Lee el valor del campo .

```

Si el cuadro de texto no está vacío, se maneja la entrada en el cuadro de texto. No se consideran ni el listado para un valor de reemplazo ni la casilla de verificación para eliminar todos los registros huérfanos. Esto queda claro por el hecho de que la entrada de texto establece que estos otros campos estén inactivos.

```

If stText <> "" Then

```

Si el cuadro de texto no está vacío, el nuevo valor se escribe en lugar del anterior utilizando el control *ID* que ha tomado el valor de la tabla. Existe la posibilidad de dos entradas, como también es el caso en el listado. El separador es `>`. Para dos entradas en tablas diferentes, se tienen que iniciar dos comandos `UPDATE`, que se crean aquí simultáneamente y se reenvían, separados por un punto y coma.

```

ElseIf oCtlList2.getSelectedItem() <> "" Then

```

Si el cuadro de texto está vacío y el listado 2 contiene un valor, el valor del listado 1 tiene que reemplazarse por el valor en el listado 2. Esto significa que todos los registros en las tablas para las que los registros en los listados son claves foráneas deben ser verificados y, si es necesario, escrito con una clave foránea alterada.

```

    sContent2 = oCtlList2.getSelectedItem()
    REM Lee valor del listado.
    REM Determina el ID para el valor del listado.

```

El valor de *ID* del segundo listado se almacena en la variable `inLB2`. Aquí también, las cosas se desarrollan de manera diferente dependiendo de si uno o dos campos están contenidos en el listado, y también de si una o dos tablas son la base del contenido del listado.

El proceso de reemplazo depende de qué tabla se define como la tabla que proporciona la clave foránea para la tabla principal. Para el ejemplo anterior, esta es la tabla de códigos postales, ya que `Postcode_ID` es la clave foránea que se reenvía a través de *Listbox 1* y *Listbox 2*.

```

If stAuxilTable = aTable(5) Then
    For i = LBound(aForeignTables()) To UBound(aForeignTables())

```

Reemplazar el antiguo valor de ID por el nuevo valor de ID se vuelve problemático en las relaciones *n:m*, ya que en tales casos, el mismo valor se puede asignar dos veces. Eso puede ser lo que se desea, pero debe evitarse cuando la clave foránea forme parte de la clave primaria. Así, en la tabla *rel_Media_Author*, un artículo no puede tener el mismo autor dos veces porque la clave primaria se construye a partir de *Media_ID* y *Author_ID*. En la consulta, se buscan todos los campos clave que colectivamente tienen la propiedad `UNIQUE` o se definieron como claves foráneas con la propiedad `UNIQUE` utilizando un índice.

Si la clave foránea tiene la propiedad ÚNICA y ya está representada allí como se desea en el futuro en LB2, esa clave no se puede reemplazar.

```
stSql = "SELECT ""COLUMN_NAME"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" =
'" + aForeignTables(i,0) + "' AND ""NON_UNIQUE"" = False AND
""INDEX_NAME"" = (SELECT ""INDEX_NAME"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" =
'" + aForeignTables(i,0) + "' AND ""COLUMN_NAME"" = '" +
aForeignTables(i,1) + "' )"
```

'NON_UNIQUE' = False' proporciona los nombres de las columnas que son ÚNICAS. Sin embargo, no se necesitan todos los nombres de columna, sino solo aquellos que forman un índice con el campo de clave foránea. La Subselección maneja esto con los mismos nombres de tabla (que contienen la clave foránea) y los nombres de los campos de clave foránea.

Si ahora la clave foránea está presente en el conjunto, el valor de la clave solo se puede reemplazar si se utilizan otros campos para definir el índice correspondiente como UNIQUE. Cuando realice reemplazos, tiene que tener cuidado de que la singularidad de la combinación de índices no se vea comprometida.

```
If aForeignTables(i,1) = stFieldName Then
    inUnique = 1
Else
    ReDim Preserve aColumns(inCount)
    aColumns(inCount) = oQuery_result.getString(1)
    inCount = inCount + 1
End If
```

Todos los nombres de columna, aparte de los nombres de columna conocidos para campos de clave foránea como Índice con la propiedad UNIQUE, se almacenan en la matriz. Como el nombre de columna del campo de clave foránea también pertenece al grupo, se puede usar para determinar si se debe verificar la unicidad durante la modificación de datos.

```
If inUnique = 1 Then
    stSql = "UPDATE "" + aForeignTables(i,0) + "" AS ""a"" SET "" +
aForeignTables(i,1) + ""=' + inLB2 + "' WHERE "" +
aForeignTables(i,1) + ""=' + inLB1 + "' AND ( SELECT COUNT(*) FROM
"" + aForeignTables(i,0) + "" WHERE "" + aForeignTables(i,1) +
""=' + inLB2 + "' )"
    If inCount > 0 Then
        stFieldgroup = Join(aColumns(), "" || ||"" )
```

Si hay varios campos, aparte del campo de clave foránea, que juntos forman un índice UNIQUE, se combinan aquí para una agrupación SQL. De lo contrario, solo aColumns(0) aparece como stFieldgroup.

```
stFieldName = ""
For ink = LBound(aColumns()) To UBound(aColumns())
    stFieldName = stFieldName + " AND "" + aColumns(ink) + "" =
""a"". "" + aColumns(ink) + "" "
```

Las partes SQL se combinan para una subconsulta correlacionada.

```
Next ink
stSql = Left(stSql, Len(stSql) - 1)
```


La consulta anterior termina con un paréntesis. Ahora se debe agregar más contenido a la subconsulta, por lo que este cierre tiene que eliminarse. Después de eso, la consulta se expande con las condiciones adicionales.

```

    stSql = stSql + stFieldName + "GROUP BY (" + stFieldgroup + ")
) < 1"
End If

```

Si la clave foránea no tiene conexión con la clave primaria o con un índice UNIQUE, no importa si el contenido está duplicado.

```

Else
    stSql = "UPDATE " + aForeignTables(i,0) + " SET " +
aForeignTables(i,1) + "'=' + inLB2 + ' WHERE " +
aForeignTables(i,1) + "'=' + inLB1 + '"
End If
oSQL_Statement.executeQuery(stSql)
NEXT

```

La actualización se lleva a cabo mientras existan diferentes conexiones a otras tablas; es decir, siempre que la tabla actual sea la fuente de una clave foránea en otra tabla. Este es el caso dos veces para la tabla *Town*: en la tabla *Media* y en la tabla *Postcode*.

Posteriormente, el valor anterior se puede eliminar del listado 1, ya que ya no tiene ninguna conexión con otras tablas.

```

stSql = "DELETE FROM " + aTable(5) + " WHERE ""ID""=' + inLB1 +
'''"
oSQL_Statement.executeQuery(stSql)

```

En algunos casos, se tiene que llevar a cabo el mismo método para una segunda tabla que ha proporcionado datos para los listados. En nuestro ejemplo, la primera tabla es la tabla *Postcode* y la segunda es la tabla *Town*.

Si el cuadro de texto está vacío y el listado 2 tampoco contiene nada, verificamos si la casilla correspondiente indica que se deben eliminar todas las entradas excedentes. Esto significa las entradas que no están vinculadas a otras tablas por una clave foránea.

```

ElseIf oCtlCheck1.State = 1 Then
    stCondition = ""
    If stAuxilTable = aTable(5) Then
        For i = LBound(aForeignTables()) To UBound(aForeignTables())
            stCondition = stCondition + ""ID"" NOT IN (SELECT "" +
aForeignTables(i,1) + "" FROM "" + aForeignTables(i,0) + "") AND
"
        Next
    Else
        For i = LBound(aForeignTables2()) To UBound(aForeignTables2())
            stCondition = stCondition + ""ID"" NOT IN (SELECT "" +
aForeignTables2(i,1) + "" FROM "" + aForeignTables2(i,0) + "")
AND "
        Next
    End If

```

El último **AND** tiene que eliminarse, ya que de lo contrario la instrucción de eliminación terminaría con **AND**.


```

    stCondition = Left(stCondition, Len(stCondition) - 4) '
    stSql = "DELETE FROM "" + stAuxilTable + "" WHERE " +
stCondition + ""
    oSQL_Statement.executeQuery(stSql)

```

Como la tabla ya se ha purgado una vez, el índice de la tabla puede verificarse y, opcionalmente, corregirse hacia abajo. Consulte el procedimiento descrito en una de las secciones anteriores.

```

Table_index_down(stAuxilTable)

```

Posteriormente, si es necesario, se puede actualizar el listado en el formulario desde el que se llamó al diálogo *Table_purge*. En algunos casos, se debe volver a leer todo el formulario. Para este fin, el registro actual se determina al comienzo del procedimiento para que después de que se haya actualizado el formulario, el registro actual se pueda restablecer.

```

oDlg.EndExecute() 'Fin del dialogo ...
oDlg.Dispose() '... y liberación de memoria
End Sub

```

Los diálogos se terminan con el comando `EndExecute()` y se eliminan por completo de la memoria con `Dispose()`.

Escribir macros con Access2Base

Las versiones de LibreOffice de la 4.2 en adelante tienen Access2Base integrado. Esta biblioteca presenta una capa básica con su API (interfaz de programación de aplicaciones) específica entre el código del usuario y la interfaz UNO habitual. La API proporcionada no trae en sí nuevas funcionalidades pero, en muchos casos, es más legible, concisa y más fácil de usar que UNO.

La API se parece mucho a la diseñada por Microsoft para el software Access. Aunque Base y Access tienen mucho en común, no así sus estilos de programación nativos. Access2Base llena el vacío.

Puede encontrar una documentación en inglés con ejemplos en <http://www.access2base.com/access2base.html>

Para dar algunos ejemplos de cómo Access2Base oculta la complejidad de UNO:

- La propiedad de valor (Access2Base sencillo) de un control tiene en UNO como equivalentes, dependiendo del tipo de control o su ubicación en un formulario, un control de tabla o un diálogo: *CurrentValue*, *Date*, *EffectiveValue*, *HiddenValue*, *ProgressValue*, *RefValue*, *ScrollValue*, *SpinValue*, *State*, *StringItem*, *List*, *Text*, *Time*, *ValueItem*, *List* o ... *Value*.
- Para obtener los N primeros registros de una tabla o una consulta en una matriz Basic, se puede usar simplemente >el método *GetRows(N)* en un objeto *Recordset*. Comparado con los métodos *getString*, *getNull*, *getDouble*, *getLong*, ... en UNO que debe aplicar en los campos según su tipo y el sistema de base de datos utilizado.

Hay dos categorías principales de objetos manejados por Access2Base, que apuntan a:

- *La interfaz de usuario*: métodos utilizados normalmente desde una aplicación Base
- *Acceso a la base de datos*: métodos utilizados desde una aplicación Base o desde cualquier otra aplicación LibreOffice

Para acceder a la biblioteca, adjunte el siguiente procedimiento de una aplicación Base al suceso `OpenDocument` de su archivo Base:

```

Sub DBOpen(Optional oEvent As Object)
    If GlobalScope.BasicLibraries.HasByName("Access2Base") Then

```

```

GlobalScope.BasicLibraries.loadLibrary("Access2Base")
End If
Call Application.OpenConnection(ThisDatabaseDocument)
End Sub

```

Alternativamente, para obtener acceso a la base de datos desde una aplicación que no sea Base, ejecute:

```

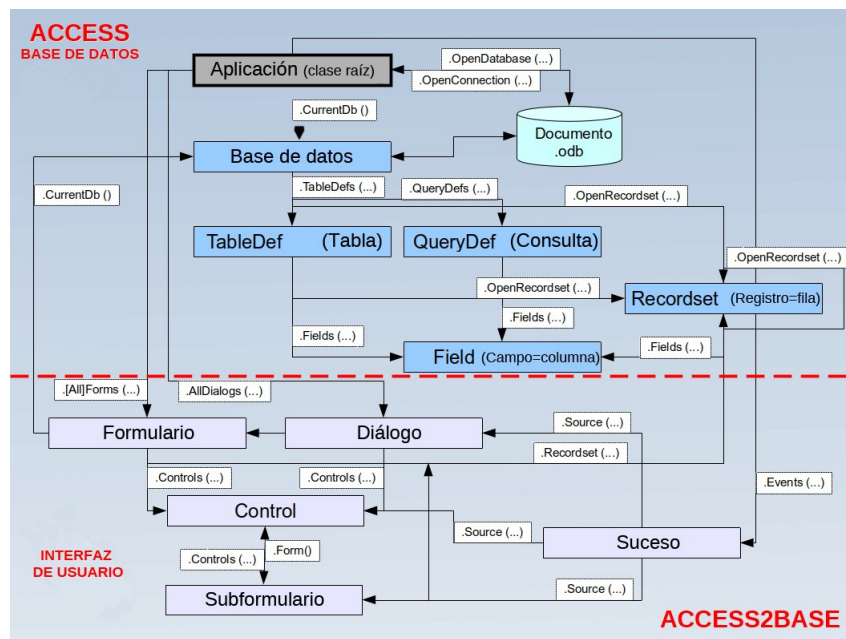
Sub DBOpen()
Dim myDb As Object
If GlobalScope.BasicLibraries.hasByName("Access2Base") then
GlobalScope.BasicLibraries.loadLibrary("Access2Base")
End If
Set myDb = Application.OpenDatabase(" ... Nombre_archivo_de_base de
datos ... ")
End Sub

```

No es la intención de esta guía replicar la documentación del sitio web mencionado anteriormente. Utilizaremos solo un resumen de los conceptos principales de la API.

El modelo Objeto

A continuación, a partir del *objeto raíz de la aplicación*, hay un esquema que describe la navegación a través de los objetos más utilizados:



Algunos ejemplos

Imprimir una lista de nombres de tablas y campos

```

Sub ScanTables()
Dim oDatabase As Object, oTable As Object, oField As Object
Dim i As Integer, j As Integer
Set oDatabase = Application.CurrentDb()
With oDatabase
For i = 0 To .TableDefs.Count - 1

```

```

        Set oTable = .TableDefs(i) 'Obtiene cada definición de tabla
individual
        DebugPrint oTable.Name
        For j = 0 To oTable.Fields.Count - 1
            Set oField = oTable.Fields(j) 'Obtiene cada campo individual
            DebugPrint "", oField.Name, oField.TypeName
        Next j
    Next i
End With
End Sub

```

Almacenar los datos producidos por una consulta en una matriz básica

```

Sub LoadQuery()
Dim oRecords As Object, vData As Variant
    Set oRecords = Application.CurrentDb().OpenRecordset("myQuery")
    vData = oRecords.GetRows(1000)
    orecords.mClose()
End Sub

```

Establecer valores predeterminados en entradas de formulario

Para hacer que después de cada entrada de registro se complete previamente algún control con el último valor establecido, asigne el siguiente procedimiento al suceso *Después del cambio de registro del formulario*:

```

Sub SetDefaultNewRec(poEvent As Object)
Dim oForm As Object, oControl As Object
    Set oForm = Application.Events(poEvent).Source 'Obtiene el
formulario en uso
    Set oControl = oForm.Controls("txtCountry")
    oControl.DefaultValue = oControl.Value
End Sub

```

Funciones de base de datos

Se proporciona una colección de funciones para acortar a una sola línea el acceso a los valores de la base de datos: *DLookup*, *DMax*, *DMin*, *Dsum*. Todos aceptan los mismos argumentos: un nombre de campo o una expresión basada en nombres de campo, un nombre de tabla o consulta y una cláusula SQL-where sin la palabra clave WHERE. Por ejemplo:

```

Function Lookup(psField As String, psSearchField As String,
psSearchValue As String) As Variant
    Lookup = Application.DLookup(psField, "myTable", _
        psSearchField & "=" & psSearchValue & "'")
End Function

```

Comandos especiales

El DoCmd (2ª clase raíz) propone un conjunto de funciones que permiten ejecutar en una declaración Basic acciones complejas aunque frecuentes y prácticas. Por nombrar unas pocas:

CopyObject	Copia una tabla o una consulta dentro de la misma base de datos o entre dos bases de datos.
------------	---

OpenSQL	Ejecuta una instrucción SQL SELECT dada y muestre el resultado en una hoja de datos.
OutputTo	Almacena los datos de una tabla o consulta en un archivo HTML. Almacena el contenido real de un formulario en un archivo PDF.
SelectObject	Activa la ventana dada (formulario, informe...)
SendObject	Envía por correo adjunto con el formulario.