LibreOffice

Guía de Base

Capítulo 8,
Tareas de bases de datos

Derechos de autor

Colaboradores

Este libro está adaptado de versiones anteriores del mismo.

De esta edición

Pulkit Krishna	Dan Lewis	Jean Hollis Weber
Juan Peramos	Juan Carlos Sanz Cabrero	

De ediciones previas

Jochen Schiffers	Robert Großkopf	Jost Lange
Hazel Russman	Dan Lewis	

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 borradao.

Fecha de publicación y versión del programa

Versión en español publicada el 4 de agosto 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.

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

Contents

Observaciones generales sobre las tareas de la base de datos

Este capítulo describe algunas soluciones para los problemas que surgen a muchos usuarios de bases de datos.

Filtrado de datos

El filtrado de datos con la interfaz gráfica de usuario (GUI, por sus siglas en inglés) se describe en el “Capítulo 3, Tablas”. Aquí describimos una solución a un problema que muchos usuarios han planteado: cómo usar las *listas desplegables* como un filtro para buscar el contenido de los campos de las tablas, para que luego aparezcan estos campos filtrados en la sección de formulario subyacente y pueden editarse.

Consulta editable

La base para este filtrado es una consulta editable (consulte el “Capítulo 5, Consultas” y la base de datos `Media_with_Macros.odb`[1]) y una tabla adicional en la que se almacenan los datos que se filtrarán. La consulta muestra, a partir de su tabla subyacente, solo los registros que corresponden a los valores del filtro. Si no se proporciona ningún valor de filtro, la consulta muestra todos los registros.

El siguiente ejemplo comienza a partir de la tabla *Media* que incluye, entre otros, los siguientes campos: *ID* (clave principal), *Title*, y *Categoy_ID*. Los tipos de campo son INTEGER, VARCHAR e INTEGER respectivamente.

Primero necesitamos una tabla *Filter*. Esta tabla contiene una clave principal y tres campos de filtro (por supuesto, puede tener más si lo desea): *ID* (clave principal), *Filter*, *Integer y Date*. Los campos *Filter* e *Integer* deben ser del mismo tipo que *Title* y *Category_ID*; en este caso serán VARCHAR e INTEGER, respectivamente. *ID* puede ser el tipo numérico más pequeño, TINYINT, porque la tabla *Filter* nunca contendrá más de un registro.

También puede filtrar los campos que aparecen en la tabla *Media* solo como claves foráneas. En ese caso, debe proporcionar a los campos correspondientes en la tabla *Filter* el tipo apropiado para las claves foráneas, generalmente INTEGER.

La siguiente consulta es ciertamente editable (vaya al panel *Base de datos* y elija el botón *Consultas* y luego haga clic en *Crear consulta en modo SQL…*). Escriba en la ventana que se abre y ejecute pulsando la tecla *F5*:

```
SELECT * FROM "Media"
```

Se muestran todos los registros de la tabla *Media*, incluida la clave primaria.

```
SELECT * FROM "Media"
WHERE "Title" = IFNULL( ( SELECT "Filter" FROM "Filter" ), "Title" )
```

Si el campo *Filter* no es NULL, se muestran aquellos registros para los cuales *Title* coincide con el valor de *Filter*. Si el campo *Filter* es NULL, se utiliza el valor del campo *Title*. Como *Title* es igual al filtro "Title", se muestran todos los registros. Sin embargo, esta suposición no se cumple si el campo *Title* de cualquier registro está vacío (contiene NULL). Eso significa que esos registros nunca se mostrarán sin entrada de título. Por lo tanto, necesitamos mejorar la consulta:

```
SELECT
   Media.* ,
   IFNULL( "Title", '' ) AS "T"
FROM "Media"
WHERE "T" = IFNULL( ( SELECT "Filter" FROM "Filter" ), "T" )
```

1 Vaya a https://documentation.libreoffice.org/assets/Uploads/Documentation/es/Base-62/SampleDatabases62.zip para descargar el archivo que contiene las bases de datos de ejemplo.

Consejo

`IFNULL (expresión, valor)` requiere que la expresión tenga el mismo tipo de campo que el valor.

Si la expresión tiene el tipo de campo VARCHAR, use dos comillas simples '' como valor. Si tiene DATE como su tipo de campo, ingrese una fecha como el valor que no está contenido en el campo de la tabla a filtrar. Utilice este formato: `{D 'AAAA-MM-DD'}`. Si es cualquiera de los tipos de campo numéricos, use el tipo de campo NUMÉRICO para el valor. Ingrese un número que no aparece en el campo de la tabla a filtrar.

Esta variante conducirá a la meta deseada. En lugar de filtrar *Title* directamente, se filtra un campo que lleva el alias *T*. Este campo tampoco tiene contenido pero no es NULL. En las condiciones de la instrucción SQL, solo se considera el campo *T*. Por lo tanto, todos los registros se muestran incluso si *Title* es NULL.

Desafortunadamente no puede hacer esto usando la interfaz gráfica de usuario. Esta instrucción solo está disponible directamente con SQL. Para que sea editable en la interfaz gráfica de usuario, se requieren modificaciones adicionales:

```
SELECT
    "Media".* ,
    IFNULL( "Media"."Title", '' ) AS "T"
FROM "Media"
WHERE "T" = IFNULL( ( SELECT "Filter" FROM "Filter" ), "T" )
```

Si la relación de la tabla con los campos ahora está configurada, la consulta se puede editar en la interfaz gráfica de usuario.

Como prueba, puede escribir un título en `"Filter"."Filter"`. Como `"Filter"."ID"` establece el valor `'0'`, el registro se guarda y se puede comprender el filtrado. Si `"Filter"."Filter"` se queda vacío, la interfaz gráfica de usuario lo trata como NULL.

Una nueva prueba produce una visualización de todos los medios de la tabla. En cualquier caso, antes de crear y probar un formulario, solo se debe ingresar un registro con una clave principal en la tabla *Filter*. Tiene que ser solo un registro, ya que las subconsultas, como se muestra arriba, solo pueden transmitir un valor.

La consulta ahora se puede ampliar para filtrar dos campos:

```
SELECT
    "Media".*,
    IFNULL( "Media"."Title", '' ) AS "T" ,
    IFNULL( "Media"."Category_ID", 100 ) AS "K"
FROM "Media"
WHERE "T" = IFNULL( ( SELECT "Filter" FROM "Filter"), "T")
AND "K" = IFNULL( ( SELECT "Integer" FROM "Filter" ), "K")
```

Esto concluye la creación de la consulta editable. Ahora, para la consulta básica para las dos listas desplegables (consideraremos una lista desplegable por filtro):

Consulta básica de dos listas desplegables

```
SELECT
    DISTINCT "Title",
    "Title"
FROM "Media"
```

```
ORDER BY "Title" ASC
```

Una de las listas desplegables debe mostrar todos los valores de *Título* y luego transmitir el *Title* seleccionado al campo *Filter* en la tabla *Filter* que subyace en el formulario. Tampoco se deben mostrar valores duplicados (por ello se incluye la condición DISTINCT). Y, por supuesto, todo debe mostrarse en el orden correcto.

Luego se crea una consulta correspondiente para el campo *Category_ID*, que es escribir sus datos en el campo *Integer* en la tabla *Filter*.

Si uno de estos campos contiene una clave externa, la consulta se adapta para que la clave externa se pase a la tabla *Filter* subyacente.

El formulario consta de dos partes. El formulario 1 es el formulario basado en la tabla *Filter*. El formulario 2 es el formulario basado en la consulta. El formulario 1 no tiene barra de navegación y el ciclo se establece en *Registro actual*. Además, la propiedad *Permitir adiciones* está establecida en *No*. El primer y único registro para este formulario ya existe.

El formulario 1 contiene dos listas desplegables con las etiquetas apropiadas. La lista desplegable 1 devuelve valores para el campo *Filter* de la tabla *Filter* y está vinculado a la consulta del campo *Title*. La lista desplegable 2 devuelve valores para el campo *Integer* de la tabla *Filter* y se relaciona con la consulta para el campo *Category_ID*.

El formulario 2 contiene un campo de control de tabla en el que se pueden enumerar todos los campos de la consulta, excepto los campos *T* y *K*. El formulario aún funcionaría si estos campos estuvieran presentes; se omiten para evitar una duplicación confusa de los contenidos de los campos. Además, el formulario 2 contiene un botón vinculado a la función *Actualizar formulario*. Se puede incorporar una barra de navegación adicional para evitar el parpadeo de la pantalla cada vez que cambia el formulario, debido a que la barra de navegación está presente en un formulario y no en el otro.

Una vez finalizado el formulario, comienza la fase de prueba. Cuando cambia una lista desplegable, el botón del formulario 2 se usa para almacenar este valor y actualizar el formulario 2. Esto ahora se relaciona con el valor que proporciona la lista desplegable. El filtrado puede hacerse retrospectivamente eligiendo un campo vacío en la lista desplegable.

Búsqueda de datos

La principal diferencia entre buscar datos y filtrar datos está en la técnica de consulta. El objetivo es entregar, en respuesta a los términos de búsqueda (independientes del idioma), una lista resultante de registros que pueden contener solo parcialmente estos términos reales. En primera instancia describiremos los enfoques similares al de tabla y de formulario.

Búsqueda con LIKE

La tabla para el contenido de búsqueda puede ser la misma que ya contiene los valores de los filtros. La tabla *Filter* simplemente se expande para incluir un campo llamado s*earchtext*. Entonces, si es necesario, se puede acceder a la misma tabla y, utilizando los formularios, filtrar y buscar simultáneamente. s*earchtext* tiene el tipo de campo VARCHAR.

El formulario se construye igual que para el del filtrado de la sección anterior. En lugar de una lista desplegable, necesitamos un campo de entrada de texto para el término de búsqueda y también, quizás, un campo de etiqueta con el título *Searchitem*. El campo para el término de búsqueda puede estar solo en el formulario o junto con los campos para el filtrado, si se desean ambas funciones.

Como ya se indicó antes, la diferencia entre filtrar y buscar radica en la técnica de consulta. Mientras que el filtrado utiliza un término que ya aparece en la tabla subyacente, la búsqueda utiliza entradas arbitrarias. (Después de todo, la lista desplegable se construye a partir del contenido de la tabla).

```
SELECT * FROM "Media"
WHERE "Title" = ( SELECT "searchtext" FROM "Filter" )
```

Esta consulta normalmente conduce a una lista de resultados vacía por estos motivos:

- Al ingresar términos de búsqueda, las personas rara vez saben completamente y con precisión cuál es el título que buscan. Por lo tanto, no se mostrará el título correcto con la instrucción SQL anterior. Para encontrar el libro *"Autoestopista a través de la galaxia"* debería ser suficiente poner *"Autoestopista"* en el campo de búsqueda o incluso simplemente *"Autoe"*.
- Si el campo *searchtext* está vacío, solo se muestran los registros en los que no hay título. Esto solo sucede si el artículo no tiene título o si alguien no ingresó su título.

El último motivo que conduce a resultados vacíos se puede eliminar si la condición de filtrado es:

```
SELECT * FROM "Media"
WHERE "Title" = IFNULL( ( SELECT "searchtext" FROM "Filter" ), "Title" )
```

Con este refinamiento del filtrado (¿qué sucede si el título es NULL?) obtenemos un resultado más acorde con las expectativas. Pero el primero motivo de resultados vacíos aún no se subsana. La búsqueda debería funcionar bien cuando solo hay conocimiento fragmentario disponible. Por lo tanto, la técnica de consulta debe usar la condición `LIKE`:

```
SELECT * FROM "Media"
WHERE "Title" LIKE ( SELECT '%' || "searchtext" || '%' FROM "Filter" )
```

o mejor aún:

```
SELECT * FROM "Media"
WHERE "Title" LIKE IFNULL( ( SELECT '%' || "searchtext" || '%' FROM "Filter" ), "Title" )
```

`LIKE`, junto con `%`, significa que se muestran todos los registros que tienen el término de búsqueda en cualquier lugar dentro de ellos. `%` es un comodín para cualquier número de caracteres antes o después del término de búsqueda. Aún quedan varias preguntas después de que se haya creado esta versión de la consulta:

- Es común usar letras minúsculas para los términos de búsqueda. Entonces, cómo se puede obtener un resultado si se escribe *"enganche"* en lugar de *"Enganche"*.
- Qué otras convenciones escritas deben considerarse.
- Qué pasa con los campos que no están formateados como campos de texto. Y si se pueden buscar o bien fechas o o bien números con el mismo campo de búsqueda.
- Y si, como en el caso del filtro, se desea evitar que los valores NULL en el campo hagan que se muestren todos los registros.

La siguiente variante cubre una o dos de estas posibilidades:

```
SELECT * FROM "Media"
WHERE LOWER("Title") LIKE IFNULL( ( SELECT '%' || LOWER("searchtext") || '%' FROM "Filter" ),
LOWER("Title") )
```

La condición cambia el término de búsqueda y el contenido del campo a minúsculas. Esto también permite comparar oraciones completas.

```
SELECT * FROM "Media"
WHERE
    LOWER("Title") LIKE IFNULL( ( SELECT '%' || LOWER("searchtext") || '%' FROM "Filter" ),
    LOWER("Title") ) OR
    LOWER("Category") LIKE ( SELECT '%' || LOWER("searchtext") || '%' FROM "Filter" )
```

La función `IFNULL` debe ocurrir solo una vez, de modo que cuando el término de búsqueda sea NULL, se consulta `LOWER ("Title") LIKE LOWER ("Title")`. Y como *Title* debe ser un campo que no

puede ser NULL, en tales casos se muestran todos los registros. Por supuesto, para múltiples búsquedas de campo, este código se vuelve correspondientemente más largo. En tales casos, es mejor usar una macro, para permitir que el código cubra todos los campos de una vez.

Pero el código todavía funciona con campos que no son texto. Aunque la condición LIKE está realmente adaptada al texto, también funciona para números, fechas y horas sin necesidad de modificaciones. Entonces, de hecho, la conversión de texto no necesita llevarse a cabo. Sin embargo, un campo de tipo tiempo, que es una mezcla de texto y números, no puede interactuar con los resultados de la búsqueda a menos que la consulta se amplíe, de modo que un solo término de búsqueda se subdivida en todos los espacios entre el texto y los números. Esto, sin embargo, aumentará significativamente la complejidad de la consulta.

Consejo

Las consultas que se utilizan para filtrar y buscar registros se pueden incrustar directamente en el formulario.

Toda la condición que se ha reunido anteriormente se puede ingresar en el filtro de fila utilizando las propiedades del formulario:

```
SELECT * FROM "Media"
   WHERE "Title" = IFNULL( ( SELECT "searchtext" FROM "Filter" ), "Title" )
```

…luego se convierte en un formulario que usa el contenido de la tabla *Media*.

Bajo "Filter" tenemos

```
("Media"."Title" = IFNULL( ( SELECT "searchtext" FROM "Filter" ), "Media"."Title" ))
```

En la entrada del filtro asegúrese de que la condición se ponga entre paréntesis y funcione con la convención de nombre de campos "NombreDeTabla"."NombreDeCampo".

La ventaja de esta variante es que el filtro se puede activar y desactivar cuando el formulario está abierto.

Búsqueda con LOCATE

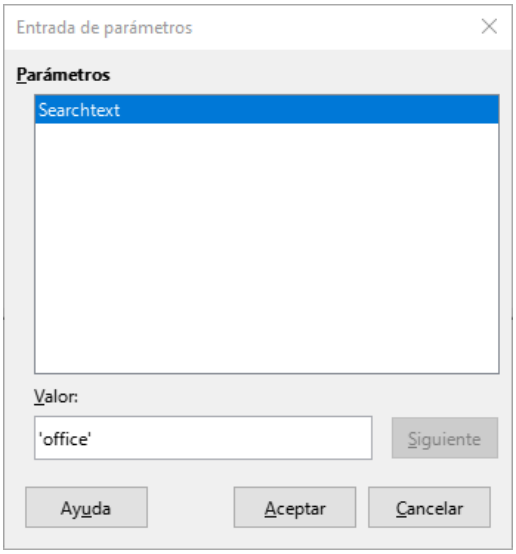
La búsqueda con LIKE suele ser satisfactoria para bases de datos con campos que contienen texto en cantidades que pueden escanearse fácilmente a simple vista. Pero no pasa lo mismo con los campos tipo LONGVARCHAR, que pueden contener varias páginas de texto. Luego, la búsqueda debe determinar dónde se puede encontrar el texto especificado.

Para localizar el texto exactamente, HSQLDB tiene la función LOCATE, la cual toma un término de búsqueda (el texto que desea buscar) como argumento. También puede agregar una posición para buscar.

En resumen: LOCATE (término de búsqueda, campo de texto de la base de datos, posición).

La siguiente explicación utiliza una tabla llamada *table*. La clave primaria se llama *ID* y debe ser única. También hay un campo llamado *memo* que se creó como un campo de tipo LONGVARCHAR. Este campo *memo* contiene algunas oraciones de este manual.[2]

2 Las capturas de pantalla para este capítulo provienen de la base de datos Example_Autotext_Searchmark_Spelling.odb, que se incluye en las bases de datos de ejemplo de este libro y que puede descargar desde https://documentation.libreoffice.org/assets/Uploads/Documentation/es/Base-62/SampleDatabases62.zip



Los ejemplos de consultas con esta función se presentan como **consultas parametrizadas**[3]. El texto de búsqueda a ingresar en estos ejemplos siempre es *"office"*.

	ID	memo
▶	1	What are all these things called? The terms used in LibreOffice for most parts of the user interface (the parts of the program
	2	Introduction In everyday office operation, spreadsheets are regularly used to aggregate sets of data
	5	General notes on the creation of a database The basics of creating a database in LibreOffice are described in Chapter 8 of the Getting
*		

Record 1 of 3

```
SELECT "ID", "memo" FROM "table"
WHERE LOWER ( "memo" ) LIKE '%' || LOWER ( :Searchtext ) || '%'
```

Primero usamos la función LIKE. Esta función solo puede usarse en sentencia de las condiciones de la instrucción SQL. Si el texto de búsqueda se encuentra en alguna parte, se muestra el registro correspondiente. La comparación es entre una versión en minúsculas del contenido del campo, usando LOWER ("memo"), y una versión en minúsculas del texto de búsqueda, usando LOWER (: Searchtext), para que la búsqueda no distinga entre mayúsculas y minúsculas. Cuanto más largo sea el texto en el campo *memo*, más difícil será ver el término en el texto recuperado.

LOCATE le muestra con mayor precisión dónde se encuentra su término de búsqueda. En los registros 1 y 2, el término no aparece. En este caso, LOCATE devuelve la posición como "0". Es fácil confirmar la cifra dada para el registro 5: la cadena "Office" comienza en la posición 6. Naturalmente, también sería posible mostrar los resultados de LOCATE de la misma manera que para LIKE.

En la columna *hit*, que se agrega también a la tabla, los resultados de la búsqueda se muestran con mayor precisión. La consulta anterior se ha utilizado como base para esta. Esto permite que la palabra *"position"* se use en la consulta externa en lugar de tener que repetir LOCATE (LOWER (:Searchtext), LOWER ("memo")) cada vez. En principio, esto no es diferente de guardar la consulta anterior y usarla como fuente para esta.

3 Las **consultas parametrizadas** son aquellas que tienen sus argumentos (o parámetros) como una variable cuyo valor puede cambiar en el transcurso de la sesión de trabajo. También conocidas como declaraciones preparadas, son un medio de precompilar una declaración SQL para que todo lo que se necesite proporcionar sean los parámetros (piense en variables) que deben insertarse en la instrucción para que se ejecute.

	ID	memo	position
▶	1	What are all these things called? The terms used in LibreOffice for most parts of the user interface (the	58
	2	Introduction In everyday office operation, spreadsheets are regularly used to aggregate	26
	3	The Base environment contains four work areas: Tables, Queries, Forms, and Reports. Depending on the work area selected, various tasks -	0
	4	Reports – presentation of data Before an actual report in the form of a recall notice can be printed, the	0
	5	General notes on the creation of a database The basics of creating a database in LibreOffice are described in Chapter	87
	6	Accessing external databases An external database must exist before it can be accessed. Assuming that	0
✱			

Record 1 of 6

```
SELECT "ID", "memo",
    LOCATE( LOWER ( :Searchtext ), LOWER ( "memo" ) ) AS "position"
FROM "table"
```

`"position" = 0` significa que no hay resultado. En este caso, se muestra `'** not found **'`.

`"position" <10` significa que el término de búsqueda aparece justo al comienzo del texto. Se pueden escanear fácilmente 10 caracteres a simple vista. Por lo tanto, se muestra todo el texto. Aquí en lugar de `SUBSTRING ("memo", 1)`, podríamos haber usado solo `"memo"`.

Para todos los demás resultados, la búsqueda busca un espacio `' '` de hasta 10 caracteres antes del término de búsqueda. El texto que se muestra no comienza en el medio de una palabra sino después de un espacio. `SUBSTRING ("memo", LOCATE (' ', "memo", "position" -10) +1)` asegura que el texto comience al comienzo de una palabra que contenga 10 caracteres como máximo antes del término de búsqueda.

En la práctica, querríamos usar más caracteres, ya que hay muchas palabras más largas que eso, y el término de búsqueda podría estar dentro de otra palabra con más de 10 caracteres en frente. *LibreOffice* contiene el término de búsqueda *"Office"* con la *"O"* como sexto carácter. Si el término de búsqueda hubiera sido *"hand"*, el registro 4 habría sido fatal para la visualización, pues contiene la palabra *"LibreOffice-Handbooks"* que tiene 12 caracteres a la izquierda de *"hand"*. Si se buscaran espacios para un máximo de 10 caracteres a la izquierda, el primero encontrado habría sido el carácter que sigue a la coma. Esto se mostraría en la columna *hit* como comenzando con *"the built-in help system…"*

	ID	memo	position	hit
▶	1	What are all these things called?	58	in LibreOffice for most p
	2	Introduction	26	everyday office operation
	3	The Base environment contains four work	0	**not found**
	4	Reports – presentation of data	0	**not found**
	5	General notes on the creation of a database	87	in LibreOffice are descri
	6	Accessing external databases	0	**not found**

Record 1 of 6

```
SELECT "ID", "memo", "position",
    CASE
        WHEN "position" = 0 THEN '**not found**'
        WHEN "position" < 10 THEN SUBSTRING ( "memo", 1, 25 )
        ELSE SUBSTRING ( "memo", LOCATE( ' ', "memo", "position" - 10 ) + 1, 25 )
    END AS "hit"
FROM
    (SELECT "ID", "memo", LOCATE(LOWER ( :Searchtext ), LOWER("memo")) AS "position"
    FROM "table" )
```

La técnica de consulta es la misma que para la consulta anterior. Solo la longitud de la coincidencia mostrada (*hit*) se ha reducido a 25 caracteres. La función `SUBSTRING` requiere como argumentos: primero, uno que busque el texto, luego la posición inicial para el resultado y, como tercer argumento opcional, la longitud de la cadena de texto que se mostrará. Aquí se ha establecido bastante corto para fines de demostración.

Una ventaja de acortarlo es que se reducen los requisitos de almacenamiento para un gran

número de registros y se puede ver fácilmente la ubicación. Una desventaja visible de este tipo de acortamiento de cadena es que el corte se realiza estrictamente de acuerdo con el límite de 25 caracteres, sin tener en cuenta dónde comienzan las palabras.

ID	memo	position	hit
1	What are all these things called?	58	in LibreOffice for most parts
2	Introduction	26	everyday office operation, spreadsheets
3	The Base environment contains four work areas:	0	**not found**
4	Reports – presentation of data	0	**not found**
5	General notes on the creation of a database	87	in LibreOffice are described
6	Accessing external databases	0	**not found**

Record 1 of 6

```
SELECT "ID", "memo", "position",
    CASE
        WHEN "position" = 0 THEN '**not found**'
        WHEN "position" < 10 THEN SUBSTRING ( "memo", 1, LOCATE(' ',"memo", 25 ) )
        ELSE SUBSTRING ( "memo", LOCATE( ' ', "memo", "position" - 10 ) + 1,
            ( LOCATE(' ',"memo", "position" + 20 ) -
                ( LOCATE(' ',"memo", "position" - 10 ) + 1 ) )
            )
    END AS "hit"
FROM
    (SELECT "ID", "memo", LOCATE(LOWER ( :Searchtext ), LOWER("memo")) AS "position"
    FROM "table" )
```

Aquí buscamos desde el carácter en la posición 25 en los *hits* hasta el siguiente lugar con este carácter. El contenido que se mostrará se encuentra entre estas dos posiciones.

Es mucho más simple si la coincidencia aparece al comienzo del campo. Aquí `LOCATE (' ', "memo", 25)` proporciona la posición exacta donde comienza el texto. Dado que queremos que el texto se muestre desde el principio, esto cumple exactamente con la duración del término que se visualizará en el formulario.

La búsqueda del espacio que sigue al término de búsqueda no es más complicada si el término se encuentra más adelante en el campo. La búsqueda simplemente comienza donde está la coincidencia. Luego se cuentan otros 20 caracteres, que se deben seguir en todos los casos. El siguiente espacio después de eso se ubica utilizando `LOCATE (' ', "memo", "position" +20)`. Esto proporciona solo la ubicación concreta dentro del campo, no la longitud de la cadena que se mostrará. Para eso, necesitamos restar la posición en la que debe comenzar la visualización del texto coincidente. Esto ya ha sido establecido dentro de la consulta con `LOCATE (' ', "memo", "position" -10) +1`. De esta manera, se puede encontrar la longitud correcta del texto.

La misma técnica se puede utilizar para encadenar consultas juntas. La consulta anterior ahora se convierte en la fuente de datos para la nueva. Se ha insertado, encerrado entre paréntesis, bajo el término `FROM`. Solo se cambia el nombre de los campos hasta cierto punto, ya que ahora hay múltiples posiciones y coincidencias. Además, la siguiente posición recibe una referencia usando `LOCATE (LOWER (: Searchtext), LOWER ("memo"), "position01" +1)`. Significa que la búsqueda comienza nuevamente en la posición después del texto coincidente anterior.

	ID	memo	position01	hit01	position02	hit02	position03
	2	Introduction	26	everyday office operation,	0	**not found**	0
▶	3	The Base	0	**not found**	0	**not found**	0
	4	Reports –	0	**not found**	0	**not found**	0
	5	General notes on the	87	in LibreOffice are described	209	of LibreOffice, called	307
	6	Accessing external	0	**not found**	0	**not found**	0

Record 3 of 6

```
SELECT "ID", "memo", "position01", "hit01", "position02",
    CASE
        WHEN "position02" = 0 THEN '**not found**'
        WHEN "position02" < 10 THEN SUBSTRING ( "memo", 1, LOCATE( ' ', "memo", 25 ) )
        ELSE SUBSTRING ( "memo", LOCATE( ' ', "memo", "position02" - 10 ) + 1,
            ( LOCATE( ' ', "memo", "position02" + 20 ) -
                ( LOCATE( ' ', "memo", "position02" - 10 ) + 1 ) )
            )
    END AS "hit02",
    CASE
        WHEN "position02" = 0 THEN 0
        ELSE LOCATE( LOWER ( :Searchtext ), LOWER ( "memo" ), "position02" + 1 )
    END AS "position03"
FROM
    (SELECT "ID", "memo", "position01",
        CASE
            WHEN "position01" = 0 THEN '**not found**'
            WHEN "position01" < 10 THEN SUBSTRING ("memo", 1, LOCATE(' ', "memo",25 ) )
            ELSE SUBSTRING ( "memo", LOCATE( ' ', "memo", "position01" - 10 ) + 1,
                ( LOCATE( ' ', "memo", "position01" + 20 ) -
                    ( LOCATE( ' ', "memo", "position01" - 10 ) + 1 ) )
                )
        END AS "hit01",
        CASE
            WHEN "position01" = 0 THEN 0
            ELSE LOCATE( LOWER ( :Searchtext ), LOWER ( "memo" ), "position01" + 1 )
        END AS "position02"
    FROM
        (SELECT "ID", "memo", LOCATE(LOWER( :Searchtext ), LOWER("memo")) As "position01"
        FROM "table")
    )
```

La consulta más externa establece los campos correspondientes para las otras dos consultas y también proporciona *hit02,* utilizando el mismo método que se utilizó anteriormente para *hit01*. Además, esta consulta más externa determina si hay más coincidencias. La posición correspondiente se da como `"position03"`. Solo el registro 5 tiene más coincidencias, y estas se pueden encontrar en otra subconsulta.

El apilamiento de consultas que se muestran aquí podría llevarse más lejos si se desea. Sin embargo, la adición de cada nueva consulta externa pone una carga adicional en el sistema. Sería necesario realizar algunas pruebas para determinar qué tan lejos fue útil y realista llegar. El “Capítulo 9, Macros” muestra cómo se pueden usar las macros para encontrar todas las cadenas de texto coincidentes en un campo mediante el uso de un formulario.

Manejo de imágenes y documentos en Base

Los formularios en Base usan controles gráficos para manejar las imágenes. Si está utilizando una base de datos interna HSQLDB, los controles gráficos son la única forma de leer imágenes de la base de datos sin usar macros. También se pueden usar como enlaces a imágenes fuera del archivo de base de datos.

En los ejemplos de esta sección se usará la base de datos de ejemplo `Example_Documents_Import_Export.odb`[4].

Leer imágenes en la base de datos

La base de datos requiere una tabla que cumpla al menos las siguientes condiciones:

Nombre del campo	***Tipo de campo***	***Descripción***
ID	INTEGER	ID es la clave principal de esta tabla.
Image	IMAGE	Contiene la imagen como dato binario.

4 Puede descargar esta base de datos de ejemplo desde https://documentation.libreoffice.org/assets/Uploads/Documentation/es/Base-62/SampleDatabases62.zip

Una clave principal (primaria) **tiene** que estar presente, pero no tiene que ser un número entero. Se deben agregar otros campos que agreguen información sobre la imagen.

Los datos que se leerán en el campo de imagen no son visibles en una tabla. En su lugar, verá la palabra <OBJECT>. Del mismo modo, las imágenes no se pueden ingresar directamente en una tabla. Debe usar un formulario que contenga un *control de imagen*, el cual se abre cuando se hace clic para mostrar un diálogo de selección de archivos. Posteriormente muestra la imagen que se leyó desde el archivo seleccionado.

Las imágenes que se deben insertar directamente en la base de datos deben ser lo más pequeñas posible. Como Base no proporciona ninguna manera (excepto mediante el uso de macros) para exportar imágenes en su tamaño original, tiene sentido usar solo el tamaño necesario, por ejemplo, el necesario para imprimir en un informe. Las imágenes cuyos tamaños oscilan los megapíxeles son completamente innecesarias e hinchan la base de datos. Después de agregar solo unas pocas imágenes, el HSQLDB interno proporciona una excepción `Java.NullPointerException` y ya no puede almacenar el registro actual. Incluso si las imágenes no son tan grandes, puede suceder que la base de datos quede inutilizable.

Además, las imágenes no deben integrarse en tablas que se diseñaron para usarse en búsquedas. Si, por ejemplo, tiene una base de datos de personal y se van a incluir imágenes para usar en pases, es mejor almacenarlos en una tabla separada con una clave foránea en la tabla principal. Esto significa que en la tabla principal se puede buscar significativamente más rápido, ya que la tabla en sí no requiere tanta memoria.

Vinculación de imágenes y documentos

Con una estructura de carpetas cuidadosamente diseñada, es más conveniente acceder a archivos externos directamente. Los archivos fuera de la base de datos pueden ser tan grandes como sea necesario, sin tener ningún efecto en el funcionamiento de la base de datos. Desafortunadamente, esto también significa que cambiar el nombre de una carpeta en su propio ordenador o en Internet puede hacer que se pierda el acceso al archivo.

Si no desea leer imágenes directamente en la base de datos, sino solo vincularlas, debe hacer un pequeño cambio en la tabla anterior:

Nombre del campo	***Tipo de campo***	***Descripción***
ID	INTEGER	ID es la clave principal de esta tabla.
Image	TEXT	Contiene la ruta a la imagen.

Si el tipo de campo se establece como TEXT, el *control de imagen* en el formulario transmitirá la ruta al archivo. El *control de imagen* aún puede acceder a la imagen exactamente como una imagen interna.

Lamentablemente, no puede hacer lo mismo con un documento. Ni siquiera es posible leer la ruta, ya que los controles gráficos están diseñados para imágenes gráficas y el diálogo de selección de archivos muestra solo los archivos con un formato gráfico.

Con una imagen, el contenido se puede ver al menos en el control gráfico si se utiliza la ruta al archivo. Un documento no se puede mostrar, incluso si la ruta está almacenada en una tabla. Primero, debemos ampliar un poco la tabla para que al menos una pequeña cantidad de información sobre el documento pueda hacerse visible.

Nombre del campo	*Tipo de campo*	*Descripción*
ID	INTEGER	ID es la clave principal de esta tabla.
Description	TEXT	Descripción del documento, términos de búsqueda.
File	TEXT	Contiene la ruta al documento.

Para hacer visible la ruta al documento, necesitamos incluir un control *selección de archivo* en el formulario.

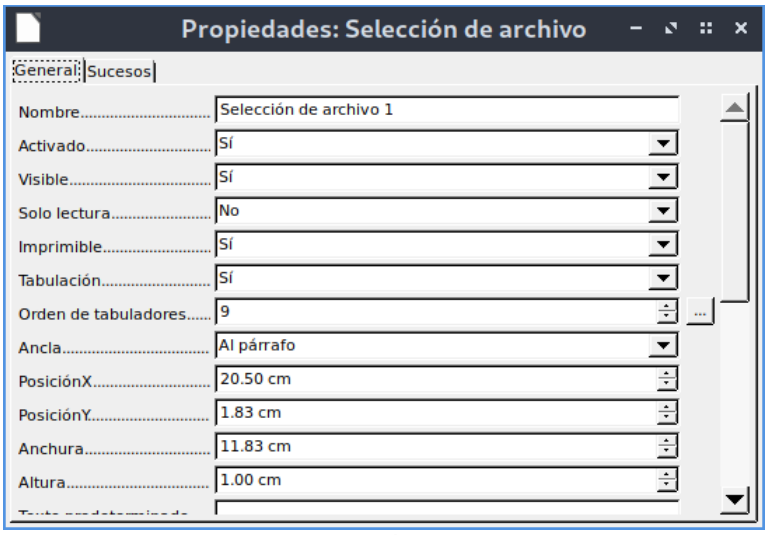


Figura 1: Control para selección de archivos. Puede insertarse en un formulario desde la barra de herramientas Más controles

Un control de *selección de archivo* no tiene pestaña *Datos* en su diálogo *Propiedades*. Por lo tanto, no está vinculado a ningún campo en la tabla subyacente.

Vinculación de documentos con una ruta absoluta

Si se usa el control de *selección de archivo*, la ruta puede mostrarse pero no almacenarse. Para esto es necesario un procedimiento especial que está vinculado a **Sucesos > Texto modificado** en el diálogo *Propiedades* del control:

```
SUB PathRead(oEvent AS OBJECT)
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM oField2 AS OBJECT
   DIM stUrl AS STRING
   oField = oEvent.Source.Model
   oForm = oField.Parent
   oField2 = oForm.getByName("graphical_control")
   IF oField.Text <> "" THEN
      stUrl = ConvertToUrl(oField.Text)
      oField2.BoundField.updateString(stUrl)
   END IF
END SUB
```

Se pasa al control el evento que desencadena el procedimiento y ayuda a encontrar el formulario y el control en el que se almacenará la ruta. El uso de `oEvent AS OBJECT` simplifica el acceso

cuando otro usuario quiere usar una macro con el mismo nombre en un subformulario. Hace que el control de *selección de archivos* sea accesible a través de `oEvent.Source.Model`. Se accede al formulario como el padre del control de *selección de archivos*. El nombre del formulario es, por lo tanto, irrelevante.

Desde el formulario, ahora se puede acceder al control llamado `"graphical_control"`. Este control se usa normalmente para almacenar las rutas a los archivos de imagen. En este caso, la URL del archivo seleccionado se escribe en él. Para asegurarse de que la URL funciona con las convenciones del sistema operativo, el texto en el control de *selección de archivos* se convierte en una forma generalmente válida mediante `ConvertToUrl`.

La tabla de la base de datos ahora contiene una ruta con el formato absoluto: *file:///…*

Si el ingreso de la ruta a los archivos se leen usando un control gráfico, esto generará una ruta relativa. Para que esto sea utilizable, debe mejorarse. El procedimiento para hacerlo es mucho más largo, ya que implica una comparación entre la ruta de entrada y la real.

Vinculación de documentos con una ruta relacionada

La siguiente macro está vinculada a la propiedad *Texto modificado* del control de *selección de archivos*. (El código se intercala con la explicación de su funcionamiento).

```
SUB PathRead
   DIM oDoc AS OBJECT
   DIM oDrawpage AS OBJECT
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM oField2 AS OBJECT
   DIM arUrl_Start()
   DIM ar()
   DIM ar1()
   DIM ar2()
   DIM stText AS STRING
   DIM stUrl_complete AS STRING
   DIM stUrl_Text AS STRING
   DIM stUrl AS STRING
   DIM stUrl_cut AS STRING
   DIM ink AS INTEGER
   DIM i AS INTEGER
   oDoc = thisComponent
   oDrawpage = oDoc.Drawpage
   oForm = oDrawpage.Forms.getByName("Form")
   oField = oForm.getByName("graphical_control")
   oField2 = oForm.getByName("filecontrol")
```

Primero, como en todos los procedimientos, se declaran las variables. Luego se buscan los campos que son importantes para la entrada de rutas.

Todo el siguiente código se lleva a cabo solo si realmente hay algo en el campo de selección de archivos, es decir, no ha sido vaciado por un cambio de registro.

```
   IF oField2.Text <> "" THEN
   arUrl_Start = split( oDoc.Parent.Url, oDoc.Parent.Title )
```

```
ar = split( ConvertToUrl(oFeld2.Text), "/" )
stText = ""
```

Se lee la ruta al archivo de la base de datos. Esto se lleva a cabo, como se muestra arriba, primero leyendo toda la URL, luego dividiéndolo en una matriz para que el primer elemento de la matriz contenga la ruta directa.

Luego, todos los elementos de la ruta que se encuentran en el control de *selección de archivos* se leen en la matriz `ar`. El separador es `/`. Esto se puede hacer directamente en Linux. En Windows, el contenido de `oField2` debe convertirse en una URL que utilizará una barra diagonal (`\`) y no una barra diagonal inversa como delimitador de ruta.

El propósito de la división es obtener la ruta al archivo simplemente cortando el nombre de archivo al final. Por lo tanto, en el siguiente paso, la ruta al archivo se vuelve a unir y se coloca en la variable `stText`. El bucle termina no con el último elemento en la matriz `ar`, sino con el elemento anterior.

```
FOR i = LBound(ar()) TO UBound(ar()) - 1
   stText = stText & ar(i) & "/"
NEXT
stText = Left(stText,Len(stText)-1)
arUrl_Start(0) = Left(arUrl_Start(0),Len(arUrl_Start(0))-1)
```

El separador final `/` se elimina nuevamente, ya que de lo contrario aparecería un valor de matriz vacío en la siguiente matriz, lo que interferiría con la comparación de la ruta. Para una comparación correcta, el texto debe convertirse en una URL adecuada que comience con `file:///`.

Finalmente, la ruta al archivo de la base de datos se compara con la ruta que se ha creado.

```
stUrl_Text = ConvertToUrl(stText)
ar1 = split(stUrl_Text,"/")
ar2 = split(arUrl_Start(0),"/")
stUrl = ""
ink = 0
stUrl_cut = ""
```

La matriz `ar1ar2` se compara paso a paso en un bucle.

```
FOR i = LBound(ar2()) TO UBound(ar2())
   IF i <= UBound(ar1()) THEN
```

El siguiente código se ejecuta solo si el número `i` no es mayor que el número de elementos en `ar1`. Si el valor en `ar2` es el mismo que el valor correspondiente en `ar1`, y hasta el momento no se ha encontrado ningún valor incompatible, el contenido común se almacena en una variable que finalmente se puede cortar del valor de la ruta.

```
      IF ar2(i) = ar1(i) AND ink = 0
         THEN  tUrl_cut = stUrl_cut & ar1(i) & "/"
      ELSE
```

Si hay una diferencia en cualquier punto entre las dos matrices, entonces, para cada valor diferente, el signo para subir un directorio se agregará a la variable `stUrl`.

```
         stUrl = stUrl & "../"
         ink = 1
      END IF
```

Tan pronto como el índice almacenado en `i` sea mayor que el número de elementos en `ar1`, cada valor adicional en `ar2` hará que se almacenen más `../` en la variable `stUrl`.

```
      ELSE
         stUrl = stUrl & "../"
      END IF
      NEXT
      stUrl_complete = ConvertToUrl(oFeld2.Text)
      oFeld.boundField.UpdateString(stUrl & Right(stUrl_complete,Len(stUrl_complete)-
Len(stUrl_cut)))
   END IF
END SUB
```

Cuando se completa el ciclo a través de `ar2`, hemos establecido si y en qué medida el archivo al que se va a acceder es más alto en el árbol que el archivo de base de datos. Ahora se puede crear `stUrl_complete` a partir del texto en el control de *selección de archivos*. Esto también contiene el nombre del archivo. Finalmente, el valor se transfiere al control gráfico. El valor de la URL comienza con `stUrl`, que contiene el número necesario de puntos (`../`). Luego, se corta el comienzo de `stUrl_complete`, la parte que resultó ser la misma para la base de datos y el archivo externo. La forma de cortar la cadena se almacena en `stUrl_cut`.

Mostrar imágenes y documentos vinculados

Las imágenes vinculadas se pueden mostrar directamente en un control gráfico. Pero una pantalla más grande sería mejor para mostrar detalles.

Los documentos normalmente no son visibles en Base.

Para hacer posible este tipo de visualización, nuevamente necesitamos usar macros. Esta macro se inicia utilizando un botón en el formulario que contiene el control gráfico. (El código se intercala con la explicación de su funcionamiento).

```
SUB View(oEvent AS OBJECT)
   DIM oDoc AS OBJECT
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM oShell AS OBJECT
   DIM stUrl AS STRING
   DIM stField AS STRING
   DIM arUrl_Start()
   oDoc = thisComponent
   oForm = oEvent.Source.Model.Parent
   oField = oForm.getByName("graphical_control")
   stUrl = oField.BoundField.getString
```

Se localiza el control gráfico en el formulario. Como la tabla no contiene la imagen en sí misma, sino solo una ruta almacenada como una cadena de texto, este texto se recupera mediante `getString`.

Luego se determina la ruta al archivo de la base de datos. Se accede al archivo *.odb*, el contenedor de los formularios, utilizando `oDoc.Parent`. La URL completa, incluido el nombre del archivo, se lee con `oDoc.Parent.Url`. El nombre de archivo también se almacena en `oDoc.Parent.Title`. El texto se separa utilizando la función de división con el nombre del

archivo como separador. Esto proporciona la ruta al archivo de la base de datos como el primer y único elemento de la matriz.

```
   arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
   oShell = createUnoService("com.sun.star.system.SystemShellExecute")
   stField = convertToUrl(arUrl_Start(0) + stUrl)
   oShell.execute(stField,,0)
END SUB
```

Los programas externos se pueden iniciar utilizando la estructura `com.sun.star.system.SystemShellExecute`. La ruta absoluta al archivo, unida desde la ruta al archivo de la base de datos y la ruta relativa almacenada internamente desde el archivo de la base de datos, se pasa al programa externo. La interfaz gráfica del sistema operativo determina qué programa se llama para abrir el archivo.

El mandato `oShell.execute` toma tres argumentos. El primero es un archivo ejecutable o la ruta a un archivo de datos que está vinculado a un programa por el sistema. El segundo es una lista de argumentos para el programa. El tercero es un número que determina cómo se deben informar los errores. Las posibilidades son `0` (mensaje de error predeterminado), `1` (sin mensaje) y `2` (solo permiten la apertura de URL absolutas).

Leer documentos en la base de datos

Al leer los documentos, siempre se deben observar las siguientes condiciones:

- Cuanto más grandes sean los documentos, más difícil de manejar será la base de datos. Por lo tanto, para documentos grandes, una base de datos externa es mejor que la interna.
- Al igual que las imágenes, los documentos no se pueden buscar. Se almacenan como datos binarios y, por lo tanto, se pueden colocar en un *contro de imagen*.
- Los documentos que deban leerse desde la base de datos interna de HSQLDB solo se pueden leer con macros. No puede hacerse solo con SQL.

  Las siguientes macros para leer dentro y fuera dependen de una tabla que incluye una descripción de los datos y el nombre de archivo original, así como una versión binaria del archivo. El nombre de archivo no se almacena automáticamente junto con el archivo, pero puede proporcionar información útil sobre el tipo de datos almacenados en un archivo que se va a leer. Solo entonces el archivo puede ser leído con seguridad por otros programas.

  La tabla contiene los siguientes campos:

Nombre del campo	***Tipo de campo***	***Descripción***
ID	INTEGER	ID es la clave principal de esta tabla.
Description	TEXT	Descripción del documento, términos de búsqueda
File	IMAGE	La imagen o archivo en forma binaria.
Filename	TEXT	El nombre del archivo, incluido el sufijo del archivo. Importante para la lectura posterior.

El formulario formulario *fileimport_name* (de la base de datos de ejemplo `Example_Documents_Import_Export.odb`[5]) para leer archivos dentro y fuera se ve así:

5 Puede descargar esta base de datos de ejemplo desde https://documentation.libreoffice.org/assets/Uploads/Documentation/es/Base-62/SampleDatabases62.zip

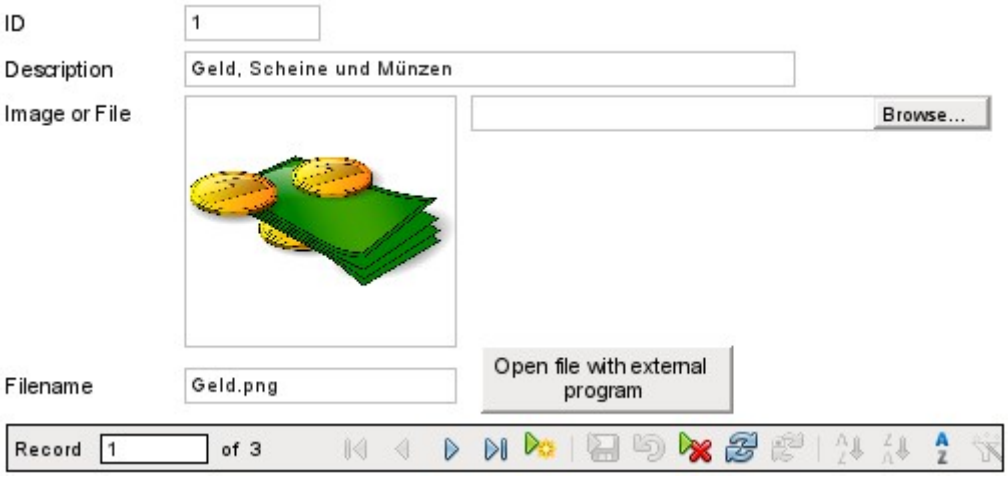


Figura 2: Formulario fileimport_name de la base de datos de ejemplo Example_Documents_Import_Export.odb

Si los archivos de imagen están presentes en la base de datos, se pueden ver en el control gráfico del formulario. Todos los demás tipos de archivo son invisibles.

La siguiente macro para leer un archivo se activa mediante el diálogo *Propiedades* del control *selección de archivos*: **Sucesos > Texto modificado**.

```
SUB FileInput_withName(oEvent AS OBJECT)
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM oField2 AS OBJECT
   DIM oField3 AS OBJECT
   DIM oStream AS OBJECT
   DIM oSimpleFileAccess AS OBJECT
   DIM stUrl AS STRING
   DIM stName AS STRING
   oField = oEvent.Source.Model
   oForm = oField.Parent
   oField2 = oForm.getByName("txt_filename")
   oField3 = oForm.getByName("graphical_control")
   IF oField.Text <> "" THEN
      stUrl = ConvertToUrl(oField.Text)
      ar = split(stUrl,"/")
      stName = ar(UBound(ar))
      oField2.BoundField.updateString(stName)
      oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
      oStream = oSimpleFileAccess.openFileRead(stUrl)
      oField3.BoundField.updateBinaryStream(oStream, oStream.getLength())
   END IF
END SUB
```

Como el evento desencadenante de la macro proporciona el nombre de otro campo de formulario, no es necesario verificar si los campos son parte del formulario principal o de un subformulario. Todo lo que es necesario saber es que todos los controles deben estar en el mismo formulario.

El control `"txt_filename"` almacena el nombre del archivo a buscar. En el caso de las imágenes, este nombre debe ingresarse a mano sin usar una macro. Aquí, en cambio, el nombre de archivo se determina a través de una URL y se ingresa automáticamente cuando se leen los datos.

El campo `"graphical_control"` almacena los datos reales, tanto para imágenes como para otros archivos.

La ruta completa, incluido el nombre del archivo, se lee desde el control de *selección de archivos* utilizando `oFeld.Text`. Para garantizar que la URL no se vea afectada por condiciones específicas del sistema operativo, el texto que se ha leído se convierte al formato de URL estándar mediante `ConvertToUrl`. Esta URL universalmente válida se divide en una matriz. El separador es `/`. El último elemento de la ruta es el nombre del archivo. `Ubound(ar)` da el índice para este último elemento. El nombre de archivo real puede leerse usando `ar(Ubound(ar))` y transferirse al campo como una cadena.

Para leer en el archivo en sí requiere `UnoService com.sun.star.ucb.SimpleFileAccess`. Este servicio puede leer el contenido del archivo como una secuencia de datos. Esto se almacena temporalmente en el objeto `oStream` y luego se inserta como una secuencia de datos en el control vinculado al campo *File* en la tabla. Esto requiere que se proporcione la longitud del flujo de datos, así como el objeto `oStream`.

Los datos ahora están dentro del control del formulario como con una entrada normal. Sin embargo, si el formulario simplemente se cierra en este punto, los datos no se almacenan. El almacenamiento requiere que se presione el botón *Guardar registro* en la barra de navegación. También ocurre automáticamente al pasar al siguiente registro.

Determinar los nombres de los archivos de imagen.

En el método anterior se mencionó brevemente que el nombre del archivo utilizado para la entrada en un control gráfico no se puede determinar directamente. Aquí hay una macro para determinar este nombre de archivo, la cual se ajusta al formulario anterior. El nombre de archivo no puede determinarse con certeza por un suceso directamente vinculado al control gráfico. Por lo tanto, la macro se inicia utilizando el diálogo **Propiedades del formulario > Sucesos > Antes de la acción en el registro de datos**.

```
SUB ImagenameRead(oEvent AS OBJECT)
   oForm = oEvent.Source
   IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
      oField = oForm.getByName("graphical_control")
      oField2 = oForm.getByName("txt_filename")
      IF oField.ImageUrl <> "" THEN
         stUrl = ConvertToUrl(oFeld.ImageUrl)
         ar = split(stUrl,"/")
         stName = ar(UBound(ar))
         oField2.BoundField.updateString(stName)
      END IF
   END IF
END SUB
```

Antes de la acción en el registro de datos se llevan a cabo dos implementaciones con sus respectivos diferentes nombres. Se puede acceder fácilmente al formulario utilizando la implementación `"ODatabaseForm"`.

En el control gráfico se puede acceder a la URL de la fuente de datos utilizando `ImageUrl`. Esta URL se lee y el nombre del archivo se determina utilizando el procedimiento anterior `FileInput_withName` para luego transferirse al campo `"txt_filename"`.

Eliminar nombres de archivos de imagen de la memoria

Si después de ejecutar la macro anterior pasa al siguiente registro, la ruta a la imagen original aún está disponible. Si ahora se lee un archivo que no es de imagen usando el control *selección de archivos*, el nombre de archivo de la imagen sobrescribirá el nombre de ese archivo, a menos que use la siguiente macro.

Lamentablemente, la macro anterior no puede eliminar la ruta, ya que el archivo de imagen solo se lee cuando se guarda el registro. Eliminar la ruta en ese punto eliminaría la imagen.

La macro se inicia utilizando el diálogo **Propiedades del formulario> Sucesos> Después de la acción en el registro de datos**.

```
SUB ImagenameReset(oEvent AS OBJECT)
   oForm = oEvent.Source
   IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
      oField = oForm.getByName("graphical_control")
      IF oField.ImageUrl <> "" THEN
         oField.ImageUrl = ""
      END IF
   END IF
END SUB
```

Como en el procedimiento `"ImagenameRead"`, en esta macro se accede al control gráfico. Si hay una entrada en `ImageUrl`, se elimina.

Lectura y visualización de imágenes y documentos

Tanto para archivos no gráficos como para imágenes de tamaño original, se debe presionar el botón *Open file with external program* del formulario *fileimport_name* de la base de datos de ejemplo `Example_Documents_Import_Export.odb`[6]. Luego, los archivos en la carpeta temporal se pueden leer y mostrar usando el programa vinculado al sufijo de archivo en el sistema operativo.

La macro se inicia usando el diálogo *Propiedades:Botón* y yendo a Sucesos > Ejecutar una acción.

```
SUB FileDisplay_withName(oEvent AS OBJECT)
   DIM oDoc AS OBJECT
   DIM oDrawpage AS OBJECT
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM oField2 AS OBJECT
   DIM oStream AS OBJECT
   DIM oShell AS OBJECT
   DIM oPath AS OBJECT
   DIM oSimpleFileAccess AS OBJECT
```

6 Puede descargar esta base de datos de ejemplo desde `https://wiki.documentfoundation.org/images/6/62/SampleDatabases62.zip`

```
   DIM stName AS STRING
   DIM stPath AS STRING
   DIM stField AS STRING
   oForm = oEvent.Source.Model.Parent
   oField = oForm.getByName("graphical_control")
   oField2 = oForm.getByName("txt_filename")
   stName = oField2.Text
   IF stName = "" THEN
      stName = "file"
   END IF
   oStream = oField.BoundField.getBinaryStream
   oPath = createUnoService("com.sun.star.util.PathSettings")
   stPath = oPath.Temp & "/" & stName
   oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
   oSimpleFileAccess.writeFile(stPath, oStream)
   oShell = createUnoService("com.sun.star.system.SystemShellExecute")
   stField = convertToUrl(stPath)
   oShell.execute(stField,,0)
END SUB
```

La posición de los otros controles afectados en el formulario viene dada por el botón. Si falta un nombre de archivo, el archivo simplemente recibe el nombre `"file"`.

El contenido del control de formulario `"graphical_control"` corresponde al del campo *File* en la tabla. Se lee como una secuencia de datos. La ruta a la carpeta temporal se usa como ruta para estos datos; se puede configurar usando **Herramientas > Opciones > LibreOffice > Rutas**.

Si los datos se utilizarán posteriormente para otros fines, y no solo se van a mostrar, se pueden copiar desde esta ruta. Dentro de la macro, el archivo se abre directamente después de una lectura exitosa, utilizando el programa que ha sido vinculado al sufijo del archivo por la interfaz gráfica de usuario del sistema operativo.

Fragmentos de código

Estos fragmentos de código proceden de consultas en las listas de correo. Surgen de problemas particulares que tal vez podrían ser útiles como soluciones para sus propios experimentos de bases de datos.

Obtener la edad actual de alguien

Una consulta debe calcular la edad real de una persona a partir de una fecha de nacimiento. Consulte también las funciones en el Apéndice de esta guía.

```
SELECT DATEDIFF( 'yy', "Birthdate", CURDATE()) AS "Age" FROM "Person"
```

Esta consulta da la edad como una diferencia en años. Pero, la edad de un niño nacido el 31 de diciembre de 2011 se daría como 1 año si la fecha actual fuera el 1 de enero de 2012. Por lo tanto, también debemos considerar la posición del día dentro del año. Esto es posible si se usa la función `DAYOFYEAR()`. Otra función llevará a cabo la comparación.

```
SELECT CASEWHEN (
   DAYOFYEAR( "Birthdate" ) > DAYOFYEAR( CURDATE() ),
   DATEDIFF ( 'yy', "Birthdate", CURDATE() ) - 1,
```

```
    DATEDIFF( 'yy', "Birthdate", CURDATE() )
)
AS "Age" FROM "Person"
```

Ahora tenemos la edad actual correcta en años.

`CASEWHEN()` también se puede usar para hacer que el texto `"Birthdate today"` aparezca en otro campo, si `DAYOFYEAR( "Birthdate" ) = DAYOFYEAR( CURDATE() )`.

Ahora podría surgir una objeción sutil: ¿Qué pasa con los años bisiestos?. Para las personas nacidas después del 28 de febrero, habrá un error de un día. No es un problema grave en el uso diario, pero deberíamos esforzarnos por la precisión.

```
CASEWHEN (
    ( MONTH( "Birthdate" ) > MONTH( CURDATE() )
    ) OR
    (
    ( MONTH( "Birthdate" ) = MONTH ( CURDATE() ) )
    AND ( DAY( "Birthdate" ) > DAY (CURDATE() ) )
    ),
DATEDIFF( 'yy', "Birthdate", CURDATE() ) - 1,
DATEDIFF( 'yy', "Birthdate", CURDATE() )
)
```

El código anterior logra este objetivo. Mientras el mes de la fecha de nacimiento sea mayor que el mes actual, la función de diferencia de año restará un año. Igualmente, se restará un año cuando los dos meses sean iguales, pero el día del mes para la fecha de nacimiento es mayor que el día en la fecha actual. Lamentablemente, esta fórmula no es comprensible para la GUI.

Solo con el diálogo *Ejecutar instrucción SQL* (que se accede desde **Herramientas > SQL…**) manejará esta consulta con éxito y eso evitaría que nuestra consulta sea editada. Pero la consulta debe ser editable, así que aquí está cómo engañar a la interfaz gráfica de usuario (GUI, por sus siglas en inglés):

```
CASE
    WHEN MONTH("Birthdate") > MONTH(CURDATE())
    THEN DATEDIFF('yy',"Birthdate",CURDATE())-1
WHEN (MONTH("Birthdate") = MONTH(CURDATE()) AND DAY("Birthdate") > DAY(CURDATE()))
    THEN DATEDIFF('yy',"Birthdate",CURDATE())-1
    ELSE DATEDIFF('yy',"Birthdate",CURDATE())
END
```

Con esta instrucción, la interfaz gráfica de usuario ya no reacciona con un mensaje de error. La edad ahora se da con precisión incluso en años bisiestos y la consulta sigue siendo editable.

Mostrar los cumpleaños que ocurrirán en los próximos días

Con un pequeño fragmento de cálculo, podemos determinar a partir de la tabla quién celebrará sus cumpleaños dentro de los próximos ocho días.

```
SELECT * FROM "Table"
WHERE
    DAYOFYEAR( "Date" ) BETWEEN DAYOFYEAR( CURDATE() )
    AND DAYOFYEAR( CURDATE() ) + 7
    OR DAYOFYEAR( "Date" ) < 7 -
        DAYOFYEAR( CAST( YEAR( CURDATE() ) || '-12-31' AS DATE) ) +
        DAYOFYEAR( CURDATE() )
```

La consulta muestra todos los registros cuya entrada de fecha se encuentra entre el día actual del año y los siguientes 7 días.

Para mostrar 8 días, incluso al final de un año, el día en que comenzó el año debe verificarse a fondo. Esta comprobación se produce solo para los números de día que son como máximo 7 días posteriores al número del último día para el año actual (generalmente 365) más el número de día para la fecha actual. Si la fecha actual es más de 7 días desde el final del año, el total es < 1. Ningún registro en la tabla tiene una fecha como esa, por lo que en tales casos esta condición parcial no se cumple.

En la fórmula anterior, los años bisiestos darán un resultado incorrecto, ya que sus fechas se desplazan por la ocurrencia del 29 de febrero. El código debe ser más extenso para evitar este error:

```
SELECT * FROM "Table"
WHERE
   CASE
      WHEN
         DAYOFYEAR(CAST(YEAR("Date")||'-12-31' AS DATE)) = 366
         AND DAYOFYEAR("Date") > 60 THEN DAYOFYEAR("Date") - 1
      ELSE
         DAYOFYEAR("Date")
   END
BETWEEN
   CASE
      WHEN
         DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) = 366
         AND DAYOFYEAR(CURDATE()) > 60 THEN DAYOFYEAR(CURDATE()) - 1
      ELSE
         DAYOFYEAR(CURDATE())
   END
AND
   CASE
      WHEN
         DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) = 366
         AND DAYOFYEAR(CURDATE()) > 60 THEN DAYOFYEAR(CURDATE()) + 6
      ELSE
         DAYOFYEAR(CURDATE()) + 7
   END
OR DAYOFYEAR("Datum") < 7 -
   DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) +
   DAYOFYEAR(CURDATE())
```

Los años bisiestos se pueden reconocer teniendo 366 como el número total de días en lugar de 365. Esto se usa para la determinación correspondiente.

Por un lado, cada valor de fecha debe probarse para ver si se encuentra en un año bisiesto, y también para el recuento correcto para el día 60 (31 días en enero y 29 en febrero). En este caso, todos los siguientes valores de DAYOFYEAR para la fecha deben incrementarse en 1. Luego, el 1 de marzo en un año bisiesto corresponderá exactamente al 1 de marzo en un año normal.

Por otro lado, el año actual (CURDATE ()) debe probarse para ver si de hecho es un año bisiesto. Aquí también el número de días debe incrementarse en 1.

Mostrar el valor final para los próximos 8 días tampoco es tan simple, ya que el año todavía no está incluido en la consulta. Sin embargo, esta sería una condición fácil de agregar para el siguiente:

YEAR("Date") = YEAR(CURDATE()) para el actual o YEAR("Date") = YEAR(CURDATE()) + 1

Agregar días al valor fecha

Al prestar medios, la biblioteca puede querer saber el día exacto en que debe devolverse el medio. Desafortunadamente, el HSQLDB interno no proporciona la función DATEADD() que está disponible en muchas bases de datos externas y también en el Firebird interno de LibreOffice Base. Aquí sigue una forma indirecta de lograr esto por un período de tiempo limitado.

Primero se crea una tabla que contiene una secuencia de fechas que cubren el lapso de tiempo deseado. Para este propósito, se abre Calc, se coloca la cadena *ID* en la celda A1 y *Date* en la celda B1. En la celda A2 ingresamos 1 y en la celda B2 la fecha de inicio, por ejemplo, 15/01/2015. Seleccione A2 y B2 y arrástrelos hacia abajo. Esto creará una secuencia de números en la columna A y una secuencia de fechas en la columna B.

Luego, toda esta tabla, incluidos los encabezados de columna, se selecciona, copia e importa a Base: haga clic con el botón derecho en el panel inferior *Tablas* y vaya al menú contextual *Pegar…* En el diálogo *Copiar tabla* que se abre, escriba *Date* en el campo *Nombre de tabla*. En Opciones, seleccione Definición y Datos y marque la casilla Usar la primera fila para los nombres de las columnas. Pulse el botón *Siguiente*. Transfiera todas las columnas, que en este caso solo son *ID* y *Date*, al panel de la derecha en el paso *Aplicar columnas*. Pulse *Siguiente* otra vez. Después de eso, en el paso *Formatos de tipos*, asegúrese de que el campo *ID* tenga el tipo INTEGER y el campo *Date* el tipo DATE. No es necesaria una clave principal, ya que los registros no se modificarán más adelante. Como no se ha definido una clave primaria, la tabla está protegida contra escritura. Pulse el botón *Crear*. Verá la nueva tabla en el panel *Tablas* de Base.

Consejo

También puede usar una técnica de consulta para crear dicha vista. Si usa una tabla de filtro, incluso puede controlar la fecha de inicio y el rango de valores de fecha.

```
SELECT DISTINCT CAST
    ( "Y"."Nr" + (SELECT "Year" FROM "Filter" WHERE "ID" = True) - 1
|| '-' ||
    CASEWHEN( "M"."Nr" < 10, '0' || "M"."Nr", '' || "M"."Nr" ) ||
'-' ||
    CASEWHEN( "D"."Nr" < 10, '0' || "D"."Nr", '' || "D"."Nr" )
    AS DATE ) AS "Date"
FROM "Nrto31" AS "D", "Nrto31" AS "M", "Nrto31" AS "Y"
WHERE "Y"."Nr" <= (SELECT "Year" FROM "Filter" WHERE "ID" = True)
AND "M"."Nr" <= 12 AND "D"."Nr" <= 31
```

Esta vista accede a una tabla que contiene solo los números del 1 al 31 y está protegida contra escritura. Otra tabla de filtros contiene el año de inicio y el intervalo de años que la vista debería cubrir. La fecha se junta a partir de estos valores, creando una expresión de fecha (año, mes, día) en texto, que luego se puede convertir en una fecha. HSQLDB acepta todos los días hasta 31 por mes y cadenas con formato como 31/02/2015. Sin embargo, el 31/02/2015 se transmite como el 3/03/2015. Por lo tanto, al preparar la vista, debe usar DISTINCT para excluir valores de fecha duplicados.

Aquí, la siguiente vista es efectiva:

```
SELECT "a"."Date",
    (SELECT COUNT(*) FROM "View_date" WHERE "Date" <= "a"."Date")
```

```
    AS "lfdNr"
FROM "View_Date" AS "a"
```

Usando la numeración de línea, el valor de la fecha se convierte en un número. Como no puede eliminar datos en una vista, no se necesita protección adicional contra escritura.

Mediante una consulta, ahora podemos determinar una fecha específica, por ejemplo, la fecha dentro de 14 días:

```
SELECT "a"."Loan_Date",
   (SELECT "Date" FROM "Date" WHERE "ID" =
      (SELECT "ID" FROM "Date" WHERE "Date" = "a"."Loan_Date")+14)
      AS "Returndate"
FROM "Loans" AS "a"
```

La primera columna muestra la fecha del préstamo. Se accede a esta columna mediante una subconsulta correlacionada que nuevamente se divide en dos consultas. `SELECT "ID" FROM "Date"` proporciona el valor del campo ID, correspondiente a la fecha de emisión, y 14 días se agrega al valor. El resultado es asignado al campo ID por la subconsulta externa. Esta nueva identificación determina qué fecha entra en el campo de fecha.

Desafortunadamente, en la visualización de esta consulta, el tipo de fecha no se reconoce automáticamente, por lo que se hace necesario utilizar el formato. En un formulario, la visualización correspondiente se puede almacenar, de modo que cada consulta arroje un valor de fecha.

Es posible una variante directa para determinar el valor de la fecha utilizando una forma más corta:

```
SELECT "Loan_Date",
   DATEDIFF( 'dd', '1899-12-30', "Loan_Date" ) + 14
      AS "Returndate"
FROM "Table"
```

El valor numérico devuelto puede formatearse dentro de un formulario como una fecha, utilizando un campo formateado. Sin embargo, se necesita mucho trabajo para que esté disponible para el procesamiento posterior de SQL en una consulta.

Agregar una hora a un intervalo de tiempo

MySQL tiene una función llamada `TIMESTAMPADD()`. Una función similar no existe en HSQLDB. Pero el valor numérico interno de la marca de tiempo se puede usar para sumar o restar, usando un campo formateado en un formulario.

A diferencia de la adición de días a una fecha, los tiempos causan un problema que puede no ser obvio al principio.

```
SELECT "DateTime"
   DATEDIFF('ss', '1899-12-30', "DateTime" ) / 86400.0000000000 +
      36/24 AS "DateTime+36hours"
FROM "Table"
```

El nuevo tiempo calculado se basa en la diferencia con el tiempo cero del sistema. Como en los cálculos de fechas, esta es la fecha del 30/12/1899.

Nota

Se supone que la fecha cero del 30/12/1899 fue elegida porque el año 1900, a diferencia de la mayoría de los años divisible por 4, no fue un año bisiesto. Entonces, la etiqueta "1" del cálculo interno se movió de nuevo al 31/12/1899 y no al 01/01/1900.

La diferencia se expresa en segundos, pero el número interno cuenta los días como números antes del punto decimal, y las horas, minutos y segundos como lugares decimales. Como un día contiene 60 * 60 * 24 segundos, el segundo recuento debe dividirse por 86400 para poder calcular los días y fracciones de días correctamente. Si el HSQLDB interno debe proporcionar decimales, debe incluirse en el cálculo, por lo que en lugar de 86400, debemos dividir por 86400.0000000000.

Los lugares decimales en una consulta deben usar un punto decimal como separador, independientemente de las convenciones locales. El resultado tendrá 10 decimales después del punto.

A este resultado se deben agregar las horas totales como una parte fraccionaria de un día. La figura calculada, con el formato adecuado, se puede crear en la consulta. Lamentablemente, el formato no se guarda, pero se puede transferir con el formato correcto utilizando un campo formateado en un formulario o informe.

Si se van a agregar minutos o segundos, tenga cuidado de que se suministren como fracciones de un día.

Si la fecha cae dentro de noviembre, diciembre, enero, etc., no hay problemas con el cálculo. Parecen bastante precisos: agregar 36 horas a una marca de tiempo del 20/01/2015 13:00:00 da 22/01/2015 00:00:00. Pero las cosas son diferentes para el 20/04/2015 13:00:00. El resultado es 22/04/2015 00:00:00. El cálculo sale mal debido al horario de verano. La hora "perdida" o "ganada" por el cambio de hora no se tiene en cuenta. Dentro de una sola zona horaria, hay varias formas de obtener un resultado "correcto". Aquí hay una variación simple:

```
SELECT "DateTime"
DATEDIFF( 'dd', 1899-12-30', "DateTime" ) +    HOUR  ( "DateTime" ) / 24.0000000000 +
   MINUTE( "DateTime" ) / 1440.0000000000 + SECOND( "DateTime" ) / 86400.0000000000 +
   36/24
AS "DateTime+36hours"  FROM "Table"
```

En lugar de contar horas, minutos y segundos desde el origen de la fecha, se cuentan a partir de la fecha actual. El 20/05/2015 la hora es 13:00 pero sin horario de verano, se mostrará como 12:00. La función HOUR tiene en cuenta el horario de verano y proporciona 13 horas como parte del tiempo por hora. Esto se puede agregar correctamente a la parte diaria. Los minutos y segundos se tratan exactamente de la misma manera. Finalmente, las horas adicionales se agregan como una parte fraccionaria de un día y todo se muestra como una marca de tiempo calculada usando el formato de celda.

Deben tenerse en cuenta dos cosas en este cálculo:

- Al pasar del horario de invierno al horario de verano, los valores por hora no salen correctamente. Esto puede corregirse usando una tabla auxiliar, que toma las fechas para el comienzo y el final del horario de verano y corrige el conteo por hora. Un negocio algo complicado.
- La visualización de tiempos solo es posible con campos formateados. El resultado es un número decimal, no una marca de tiempo que podría almacenarse directamente como tal en la base de datos. Debe copiarse dentro del formulario o convertirse de un número decimal a una marca de tiempo mediante una consulta complicada. El punto de ruptura en la conversión es el valor de la fecha, ya que pueden estar involucrados años bisiestos o meses con diferentes números de días.

Obtener un saldo de cuenta corriente por conceptos

En lugar de usar una agenda doméstica, una base de datos en un PC puede simplificar el trabajo agotador de sumar gastos para alimentos, ropa, transporte, etc. Queremos que la mayoría de estos detalles sean visibles de inmediato en la base de datos, por lo que nuestro ejemplo supone que los ingresos y los gastos se almacenarán como valores señalados en un campo llamado *Amount*. En principio, todo se puede ampliar para cubrir campos separados y una suma relevante para cada uno.

```
SELECT "ID", "Amount", (SELECT SUM("Amount")
FROM "Cash"
WHERE "ID" <= "a"."ID") AS "Balance" FROM "Cash" AS "a"
ORDER BY "ID" ASC
```

Esta consulta provoca para cada nuevo registro un cálculo directo del saldo de la cuenta corriente. Al mismo tiempo, la consulta sigue siendo editable porque el campo *Balance* se crea a través de una subconsulta de correlación. La consulta depende del *ID* de clave principal creada automáticamente para calcular el estado de la cuenta. Sin embargo, los saldos generalmente se calculan diariamente. Entonces necesitamos una consulta de fecha.

```
SELECT "ID", "Date", "Amount", ( SELECT SUM( "Amount" )
FROM "Cash"
WHERE "Date" <= "a"."Date" ) AS "Balance" FROM "Cash" AS "a"
ORDER BY "Date", "ID" ASC
```

El gasto ahora aparece ordenado y sumado por fecha. Todavía queda la cuestión de la categoría, ya que queremos los saldos correspondientes para las categorías individuales de gasto.

```
SELECT "ID", "Date", "Amount", "Acct_ID",
( SELECT "Acct" FROM "Acct" WHERE "ID" = "a"."Acct_ID" ) AS "Acct_name",
( SELECT SUM( "Amount" ) FROM "Cash" WHERE "Date" <= "a"."Date" AND "Acct_ID" =
"a"."Acct_ID" ) AS "Balance",
( SELECT SUM( "Amount" ) FROM "Cash" WHERE "Date" <= "a"."Date" ) AS "Total_balance"
FROM "Cash" AS "a"
ORDER BY "Date", "ID" ASC
```

Esto crea una consulta editable en la que, además de los campos de entrada (*Date, Amount, Acct_ID*), el nombre de la cuenta, el saldo relevante y el saldo total aparecen juntos. Como las subconsultas de correlación se basan parcialmente en entradas anteriores (`"Date" <= "a"."Date"`), solo las nuevas entradas pasarán sin problemas. Las alteraciones a un registro anterior son inicialmente detectables solo en ese registro. La consulta debe actualizarse si se van a realizar cálculos posteriores que dependen de ella.

Numerar líneas

Los campos que se incrementan automáticamente están bien. Sin embargo, no le dicen definitivamente cuántos registros están presentes en la base de datos o están realmente disponibles para ser consultados. Los registros a menudo se eliminan y muchos usuarios intentan en vano determinar qué números ya no están presentes para que el número corrido coincida.

```
SELECT "ID", ( SELECT COUNT( "ID" ) FROM "Table" WHERE "ID" <= "a"."ID" ) AS "Nr."
FROM "Table" AS "a"
```

El campo *ID* se lee, y el segundo campo se determina mediante una subconsulta de correlación, que busca determinar cuántos valores de campo en *ID* son menores o iguales al valor de campo actual. A partir de esto se crea un número de línea en ejecución.

Cada registro al que desea aplicar esta consulta contiene campos. Para aplicar esta consulta a los registros, primero debe agregar estos campos a la consulta. Puede colocarlos en el orden que

desee en la cláusula SELECT. Si tiene los registros en un formulario, debe modificar el formulario para que los datos del formulario provengan de esta consulta.

Por ejemplo, el registro contiene *field1, field2* y *field3*. La consulta completa sería:

```
SELECT "ID", "field1", "field2", "field3", ( SELECT COUNT( "ID" ) FROM "Table" WHERE "ID" <=
"a"."ID" ) AS "Nr." FROM "Table" AS "a"
```

También es posible una numeración para una agrupación correspondiente:

```
SELECT "ID", "Calculation", ( SELECT COUNT( "ID" ) FROM "Table" WHERE
"ID" <= "a"."ID" AND "Calculation" = "a"."Calculation" ) AS "Nr." FROM
"Table" AS "a" ORDER BY "ID" ASC, "Nr." ASC
```

Aquí una tabla contiene diferentes números calculados. ("Calculation"). Para cada número calculado, "Nr." se expresa por separado en orden ascendente después de ordenar en el campo *ID*. Esto produce una numeración de 1 en adelante.

Si el orden de clasificación real dentro de la consulta está de acuerdo con los números de línea, se debe asignar un tipo apropiado de clasificación. Para este propósito, el campo de clasificación debe tener un valor único en todos los registros. De lo contrario, dos números de lugar tendrán el mismo valor. Esto realmente puede ser útil si, por ejemplo, se representa el orden de lugar en una competición, ya que resultados idénticos conducirán a una posición conjunta. Para que el orden de lugar se exprese de tal manera que, en caso de posiciones conjuntas, se omita el valor siguiente, la consulta debe construirse de manera algo diferente:

```
SELECT "ID",
   ( SELECT COUNT( "ID" ) + 1 FROM "Table" WHERE "Time" < "a"."Time" ) AS "Place"
FROM "Table" AS "a"
```

Se evalúan todas las entradas para las cuales el campo *Time* tiene un valor menor. Eso cubre a todos los atletas que llegaron al puesto ganador antes que el atleta actual. A este valor se agrega el número 1. Esto determina el lugar del atleta actual. Si el tiempo es idéntico al de otro atleta, se colocan conjuntamente. Esto hace posible situar los puestos como 1er lugar, 2º lugar, 2º lugar, 4º lugar.

Sería más problemático si se necesitaran los números de línea y el lugar al mismo tiempo. Podría ser útil si se necesita combinar varios registros en una línea.

```
SELECT "ID", ( SELECT COUNT( "ID" ) + 1 FROM "Table"
   WHERE "Time" < "a"."Time" ) AS "Place",
CASE WHEN(SELECT COUNT("ID")+1 FROM "Table" WHERE "Time"="a"."Time" ) = 1
   THEN (SELECT COUNT("ID")+1 FROM "Table" WHERE "Time"<"a"."Time")
   ELSE (SELECT(SELECT COUNT("ID")+1 FROM "Table" WHERE "Time"<"a"."Time")
   + COUNT("ID") FROM "Table" WHERE "Time"="a"."Time" "ID"<"a"."ID"
END
AS "LineNumber" FROM "Table" AS "a"
```

La segunda columna todavía da el orden del puesto. La tercera columna verifica primero si solo una persona cruzó la línea esta vez. Si es así, el orden de puestos se convierte directamente en un número de línea. De lo contrario, se agrega un valor adicional al puesto. Por el mismo tiempo ("Time" = "a"."Time") se agrega al menos 1, si hay otra persona con la *ID* de clave primaria, cuya clave primaria es más pequeña que la clave primaria en el registro actual ("ID"<"a"."ID"). Por lo tanto, esta consulta produce valores idénticos para el puesto siempre que no exista una segunda persona con el mismo tiempo. Si existe una segunda persona con el mismo tiempo, la identificación determina qué persona tiene el número de línea menor.

Por cierto, esta clasificación por número de línea puede servir para cualquier propósito que deseen los usuarios de la base de datos. Por ejemplo, si una serie de registros se ordena por nombre, los registros con el mismo nombre no se ordenan al azar, sino de acuerdo con su clave

principal, que por supuesto es única. También de esta manera, la numeración puede conducir a una clasificación de registros.

La numeración de líneas también es un buen preludio para la combinación de registros individuales en un solo registro. Si se crea una consulta de numeración de líneas como una vista, se puede aplicar una consulta adicional sin crear ningún problema. Como ejemplo simple, aquí una vez más es la primera consulta de numeración con un campo adicional:

```
SELECT "ID", "Name",
   ( SELECT COUNT( "ID" ) FROM "Table" WHERE "ID" <= "a"."ID" ) AS "Nr."
FROM "Table" AS "a"
```

Esta consulta se convierte en la vista *View1*. La consulta se puede usar, por ejemplo, para poner los primeros tres nombres juntos en una línea:

```
SELECT "Name" AS "Name_1",
   (SELECT "Name" FROM "View1" WHERE "Nr."=2) AS "Name_2",
   (SELECT "Name" FROM "View1" WHERE "Nr."=3) AS "Name_3"
FROM "View1" WHERE "Nr."=1
```

De esta forma, se pueden convertir varios registros en campos adyacentes. Esta numeración se ejecuta desde el primero hasta el último registro. Si a todas estas personas se les debe asignar el mismo apellido, esto se puede llevar a cabo de la siguiente manera:

```
SELECT "ID", "Name", "Surname",
   (SELECT COUNT("ID") FROM "Table" WHERE "ID"<="a"."ID" AND "Surname"="a"."Surname")
AS "Nr." FROM "Table" AS "a"
```

Ahora que se ha creado la vista, la familia se puede reunir.

```
SELECT "Surname", "Name" AS "Name_1",
( SELECT "Name" FROM "View1" WHERE "Nr." = 2 AND "Surname" = "a"."Surname") AS
"Name_2",
( SELECT "Name" FROM "View1" WHERE "Nr." = 3 AND "Surname" = "a"."Surname") AS
"Name_3"
FROM "View1" AS "a" WHERE "Nr." = 1
```

Así, en una libreta de direcciones, todos los miembros de una familia (`"Surname"`) se pueden reunir para que cada dirección se tenga en cuenta solo una vez al enviar una carta, pero todos los que deberían recibirla se enumeran.

Hay que tener cuidado aquí, pues no se quiere una función de bucle sin fin. La consulta en el ejemplo anterior limita los registros paralelos que se convertirán en campos a 3. Este límite se eligió deliberadamente. No aparecerán más nombres incluso si el valor de `"Nr."` es mayor que 3. En algunos casos, ese límite es claramente comprensible. Por ejemplo, si estamos creando un calendario, las líneas pueden representar las semanas del año y las columnas los días de la semana. Como en el calendario original solamente la fecha determina el contenido del campo, la numeración de línea se usa para numerar los días de cada semana de forma continua y luego las semanas del año se convierten en registros. Esto requiere que la tabla contenga un campo de fecha con fechas continuas y un campo para los eventos. Además, la fecha más temprana siempre creará un `"Nr." = 1`. Entonces, si desea que el calendario comience el lunes, la fecha más temprana debe ser el lunes. La columna 1 es el lunes, la 2 el martes y así sucesivamente. La subconsulta luego termina en `"Nr." = 7`. De esta forma, los siete días de la semana se pueden mostrar uno al lado del otro y se puede crear una vista de calendario correspondiente.

Obtener un salto de línea a través de una consulta

A veces es útil agrupar varios campos mediante una consulta y separarlos por saltos de línea, por ejemplo, al leer una dirección completa en un informe.

El salto de línea dentro de la consulta está representado por `Char (13)`. Ejemplo:

```
SELECT "Firstname" || ' ' || "Surname" || Char(13) || "Road" || Char(13) || "Town" FROM "Table"
```

Esto produce:

Firstname Surname
Road
Town

Dicha consulta, con una línea que numera hasta 3, le permite imprimir etiquetas de dirección en tres columnas mediante la creación de un informe. La numeración es necesaria a este respecto para que se puedan colocar tres direcciones una al lado de la otra en un registro. Esa es la única forma en que permanecerán uno al lado del otro cuando se lean en el informe.

En algunos sistemas operativos es necesario incluir `Char(10)` junto con `Char (13)` en el código.

```
SELECT "Firstname" || ' ' || "Surname" || Char(13) || Char(10) || "Street" || Char(13) ||Char(10) ||
"Town" FROM "Table"
```

Agrupar y resumir

Para otras bases de datos y para versiones nuevas de HSQLDB, el mandato `Group_Concat()` está disponible. Se puede usar para agrupar campos individuales en un registro en un campo. Entonces, por ejemplo, es posible almacenar nombres y apellidos en una tabla, luego presentar los datos de tal manera que un campo muestre los apellidos como apellidos, mientras que un segundo campo contiene todos los nombres relevantes secuencialmente, separados por comas.

Este ejemplo es similar en muchos aspectos a la numeración de líneas. La agrupación en un campo común es un tipo de complemento a esto.

Surname	***First name***
Müller	Karin
Schneider	Gerd
Müller	Egon
Schneider	Volker
Müller	Monika
Müller	Rita

es convertido por la consulta a:

Surname	***First name***
Müller	Karin, Egon, Monika, Rita
Schneider	Gerd, Volker

Este procedimiento puede, dentro de ciertos límites, expresarse en HSQLDB.

El siguiente ejemplo se refiere a una tabla llamada *Name* con los campos *ID, Firstname* y *Surname*. La siguiente consulta se ejecuta primero en la tabla y se guarda como una vista llamada *View_Group*.

```
SELECT "Surname", "Firstname",
   ( SELECT COUNT( "ID" ) FROM "Name" WHERE "ID" <= "a"."ID"
   AND "Surname" = "a"."Surname" ) AS "GroupNr"
FROM "Name" AS "a"
```

Puede leer en el “Capítulo 5, Consultas” cómo esta consulta accede al contenido del campo en la misma línea de consulta. Produce una secuencia numerada ascendente, agrupada por apellido. Esta numeración es necesaria para la siguiente consulta, de modo que en el ejemplo se enumere un máximo de 5 nombres.

```
SELECT "Surname",
( SELECT "Firstname" FROM "View_Group" WHERE "Surname" = "a"."Surname" AND "GroupNr"
= 1 ) ||
IFNULL( ( SELECT ', ' || "Firstname" FROM "View_Group" WHERE "Surname" = "a"."Surname"
AND "GroupNr" = 2 ), '' ) ||
IFNULL( ( SELECT ', ' || "Firstname" FROM "View_Group" WHERE "Surname" = "a"."Surname"
AND "GroupNr" = 3 ), '' ) ||
IFNULL( ( SELECT ', ' || "Firstname" FROM "View_Group" WHERE "Surname" = "a"."Surname"
AND "GroupNr" = 4 ), '' ) ||
IFNULL( ( SELECT ', ' || "Firstname" FROM "View_Group" WHERE "Surname" = "a"."Surname"
AND "GroupNr" = 5 ), '' )
AS "Firstnames"
FROM "View_Group" AS "a"
```

Usando subconsultas, los nombres de los miembros del grupo se buscan uno tras otro y se combinan.

A partir de la segunda subconsulta, debe asegurarse de que los valores `'NULL'` no establezcan la combinación completa en `'NULL'`. Es por eso que se muestra un resultado de `''` en lugar de `'NULL'`.