



Guía de Base

Capítulo 5
Consultas

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
Juan Peramos

Jean Hollis Weber
Juan Carlos Sanz Cabrero

B. Antonio Fernández

De ediciones previas

Jochen Schiffers
Hazel Russman

Robert Großkopf
Steve Schwettman

Jost Lange
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 14 de junio 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
<i>Clic con el botón derecho</i>	<i>Control+clic o clic derecho</i> depende de la configuración del equipo	Abre menú contextual
<i>Ctrl (Control)</i>	⌘ (Comando)	Utilizado con otras teclas
<i>F5</i>	Mayúscula+⌘+F5	Abre el navegador
<i>F11</i>	⌘+T	Abre la ventana de estilos y formato

Contenido

Derechos de autor.....	2
Colaboradores.....	2
De ediciones previas.....	2
Fecha de publicación y versión del programa.....	2
Uso de LibreOffice en macOS.....	2
Información general sobre consultas.....	5
Crear consultas.....	5
Crear consultas utilizando el diálogo Diseño de consulta.....	5
Usar funciones en una consulta.....	12
Definir relaciones en la consulta.....	15
Definir propiedades de consulta.....	18
Mejora de consultas usando el modo SQL.....	19
Usar un alias en una consulta.....	29
Consultas para la creación de campos en Listado.....	30
Consultas como base para información adicional en formularios.....	32
Posibilidades de introducción de datos en consultas.....	33
Uso de parámetros en consultas.....	37
Subconsultas.....	38
Subconsultas relacionadas.....	39
Consultas como tablas fuente para consultas.....	39
Resumiendo datos con consultas.....	43
Acceso más rápido a consultas mediante vistas de tabla.....	44
Errores de cálculo en consultas.....	45

Información general sobre consultas

Las consultas a una base de datos son la herramienta más poderosa que tenemos para usar bases de datos de manera práctica. Pueden reunir datos de diferentes tablas, calcular resultados cuando sea necesario y filtrar rápidamente un registro específico de una masa de datos. Las grandes bases de datos de Internet que las personas usan todos los días existen principalmente para ofrecer un resultado rápido y práctico para el usuario a partir de una gran cantidad de información mediante una selección cuidadosa de palabras clave, incluidos los anuncios relacionados con la búsqueda que alientan a las personas a realizar compras.

Crear consultas

Las consultas se pueden crear tanto en la Interfaz como directamente como código SQL. En ambos casos, se abre una ventana, donde puede crear una consulta y también corregirla si es necesario.

Crear consultas utilizando el diálogo Diseño de consulta

La creación de consultas con el asistente se describe brevemente en el «Capítulo 8, Primeros pasos con Base» de la *Guía de primeros pasos*. Aquí explicaremos la creación directa de consultas en la Vista de diseño.

En la ventana principal de la base de datos, haga clic en el icono *Consultas* en la sección *Bases de datos*, luego en la sección *Tareas*, haga clic en *Crear consulta en modo de diseño*. Aparecerá la ventana de diseño con dos áreas para la creación en vista de diseño de la consulta y sobre ella un diálogo para agregar tablas o consultas.

La ventana permite combinar varias tablas (y también vistas y consultas). Para agregar una tabla seleccione la opción *Tablas* en el diálogo superpuesto, seleccione una tabla, y luego haga clic en el botón *Añadir*. O haga doble clic en el nombre de la tabla. Cualquiera de los métodos agrega la tabla al área gráfica del diálogo *Diseño de consulta* (figura 1).

Cuando se hayan seleccionado todas las tablas o consultas necesarias, haga clic en el botón *Cerrar*. Se pueden agregar tablas y consultas adicionales más adelante si es necesario. Sin embargo, no se puede crear una consulta sin al menos una tabla, por lo que se tiene que hacer una selección al principio.

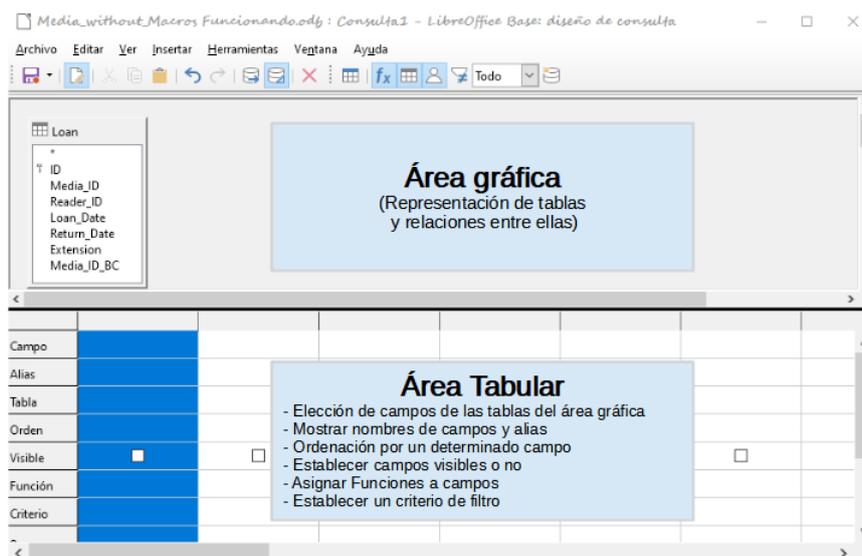


Figura 1: Áreas de Diseño de consulta

La figura 1 muestra las divisiones básicas del diálogo *Diseño de consulta*: El *área gráfica* muestra las tablas que se vincularán a la consulta. También se pueden mostrar las relaciones entre sí en

relación con la consulta. El *área tabular* es para la selección de campos para mostrar o para establecer condiciones relacionadas con estos campos.

Como nuestro formulario sencillo se refiere a la tabla *Loan*, primero explicaremos la creación de una consulta usando esta tabla.



En el cuadro de diálogo seleccione *Tablas* y, de las disponibles, seleccione la tabla *Loan* y haga clic en el botón *Añadir*, luego en el botón *Cerrar*.

En el *área tabular* haga clic en *Campo*, en la primera columna, para visualizar una flecha hacia abajo. Haga clic en esta flecha para abrir la lista desplegable de campos disponibles. El formato es `[Nombre_de_tabla.Nombre_de_campo]`, razón por la cual todos los nombres de campo comienzan aquí con la palabra *Loan* y un punto.

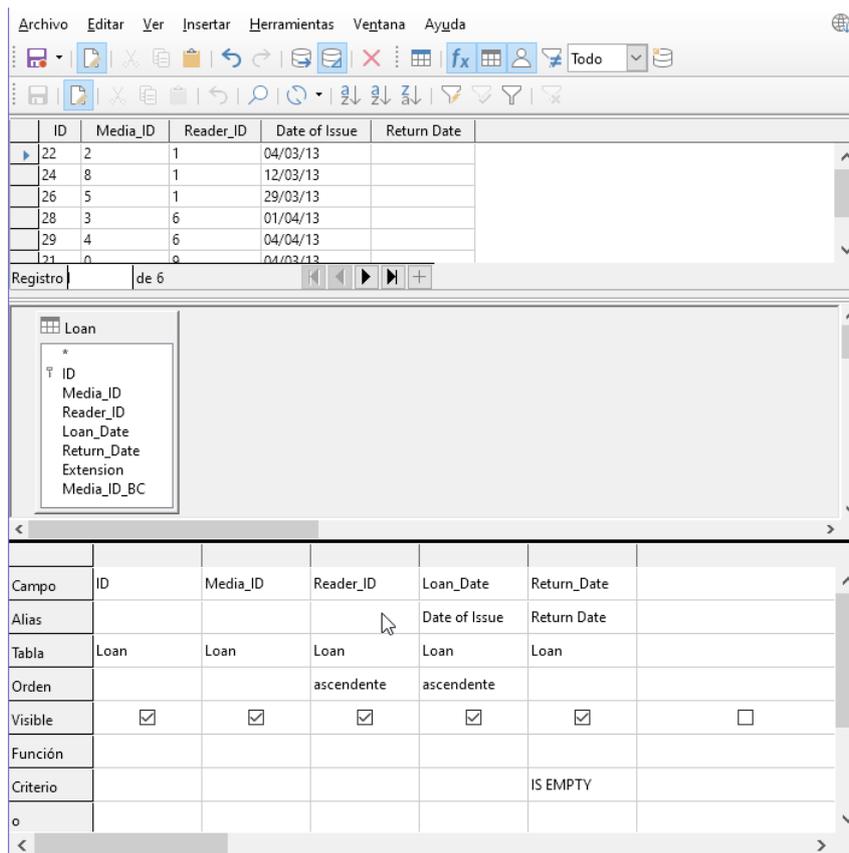
Campo	
Alias	Loan.* Loan.ID
Tabla	Loan.Media_ID Loan.Reader_ID
Orden	Loan.Loan_Date
Visible	Loan.Return_Date Loan.Extension
Función	Loan.Media_ID_BC
Criterio	
o	

La selección del campo, designada como *Loan.**, tiene un significado especial. Un clic en esta opción agregará todos los campos de la tabla subyacente a la consulta. Cuando utiliza esta designación de campo con el comodín «*», la apariencia de la consulta se vuelve indistinguible de la tabla que se consulta.

Consejo

Para una transferencia rápida de los campos de una tabla al *área tabular* sitúese en la vista de tabla en el *área gráfica* y haga doble clic en cada campo, esta operación inserta ese campo en el *área tabular* en la siguiente posición libre.

Para la consulta que estamos creando, seleccione los primeros cinco campos de la tabla *Loan*, uno a continuación del otro. Vaya a la columna *Loan_Date* y en la fila criterio escriba `IS EMPTY`. De esta manera habrá creado su primera consulta consistente en que se muestren las filas de la tabla *Loan* exceptuando las que tengan algo escrito en su correspondiente columna.



Las consultas en modo de diseño siempre se pueden probar. Para comprobar el resultado de una consulta debe hacer clic en el botón *Ejecutar consulta* de la barra de herramientas. Esto hace que aparezca una nueva área con el resultado de la consulta en vista tabular de los datos resultantes sobre el área gráfica.

Una ejecución de prueba de una consulta siempre es útil antes de guardarla, para aclarar si la consulta realmente logra su objetivo. A menudo, un error lógico impide que una consulta obtenga datos. En otros casos, puede suceder que se muestren precisamente esos registros que desea excluir.

En principio, una consulta que produce un mensaje de error en la base de datos subyacente no se puede guardar hasta que se corrija el error.

	ID	Media_ID
▶	22	2
	24	8
	26	5
	28	3
	29	4
	21	0
+	<Campo automático>	

Figura 2: Consulta editable

	Media_ID	Reader_ID
▶	2	1
	8	1
	5	1
	3	6
	4	6
	0	9

Figura 3: Consulta no editable

En el resultado de una prueba, preste especial atención a la primera columna de la tabla resultante de la consulta. El marcador de registro activo (flecha) siempre aparece en el lado izquierdo de la tabla, en este caso apuntando al primer registro como el registro activo.

Mientras que el texto del primer campo del primer registro en la figura 2 está resaltado, el campo correspondiente en la figura 3 muestra solo un borde discontinuo. El resaltado indica que este campo se puede modificar, es decir, que los registros se pueden editar. El borde discontinuo indica que este campo no se puede modificar.

La figura 2 también contiene una línea adicional para la entrada de un nuevo registro, con el campo *ID* ya marcado como *<Campo automático>*. Esto también muestra que se pueden agregar nuevas entradas.



Consejo

Una regla básica es que si la clave principal (de la tabla consultada) no está incluida en la consulta no son posibles nuevas entradas.

Campo	ID	Media_ID	Reader_ID	Loan_Date	Return_Date
Alias				Date of Issue	Return Date

Los campos *Loan_Date* y *Return_Date* tienen alias. Esto no hace que se les cambie el nombre a los campos de la tabla, sino que aparezcan estos otros nombres en la consulta del usuario.

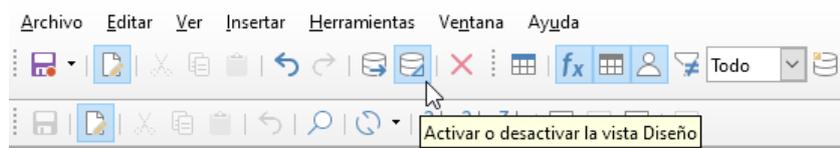
	ID	Media_ID	Reader_ID	Date of Issue	Return Date
▶	22	2	1	04/03/13	

La vista de tabla anterior muestra cómo los alias reemplazan los nombres de campo reales.

Return_Date
Return Date
Loan
<input checked="" type="checkbox"/>
IS EMPTY

El campo *Return_Date* no solo tiene un alias, sino también un criterio de búsqueda, lo que hará que solo se muestren aquellos registros para los que este campo esté vacío (Al poner **IS EMPTY** en la fila *Criterio* del campo *Return_Date*).

Este criterio de exclusión hará que solo se muestren aquellos registros relacionados con artículos del préstamo que aún no han sido devueltos.



Para aprender mejor el lenguaje SQL, es aconsejable cambiar de vez en cuando al *modo SQL*. Esto se hace mediante el botón *Activar o desactivar la vista Diseño*. Lo cual muestra la ventana del editor de texto de SQL y oculta el área gráfica y el área tabular.

Para volver al modo de vista *Diseño* vuelva a pulsar este mismo botón.

ID	Media_ID	Reader_ID	Date of Issue	Return Date
22	2	1	04/03/13	
24	8	1	12/03/13	
26	5	1	29/03/13	
28	3	6	01/04/13	
29	4	6	04/04/13	
21	0	9	04/03/13	

```

SELECT "ID", "Media_ID", "Reader_ID", "Loan_Date" AS "Date of Issue",
"Return_Date" AS "Return Date"
FROM "Loan" WHERE "Return_Date" IS NULL
ORDER BY "Reader_ID" ASC, "Date of Issue" ASC;

```

Aquí se revela la fórmula SQL creada por nuestras elecciones anteriores. Para facilitar la lectura, se han incluido algunos saltos de línea. Desafortunadamente, el editor no almacena estos saltos de línea, por lo que cuando se vuelve a llamar la consulta, aparecerá como un salto de línea continuo en el borde de la ventana.

- **SELECT** es el principio de los criterios de selección (los campos que se mostrarán).
- **AS** especifica los alias de campo que se utilizarán. (en algunos casos esta instrucción no aparece, simplemente se muestra el nombre del campo y su alias a continuación).
- **FROM** muestra la tabla que se utilizará como fuente de la consulta.
- **WHERE** proporciona las condiciones para la consulta, en este caso, que el campo *Return_Date* debe estar vacío (*IS NULL*).
- **ORDER BY** define los criterios de clasificación también en este caso orden ascendente (*ASC*) para los dos campos *Reader_ID* y *Loan_Date*. Esta especificación de clasificación ilustra cómo se puede usar el alias para el campo *Loan_Date* dentro de la consulta misma.



Consejo

Cuando trabaje en el modo Vista de diseño, use **IS EMPTY** para un campo vacío. Cuando trabaje en modo SQL, use **IS NULL**, que es lo que reconoce SQL.

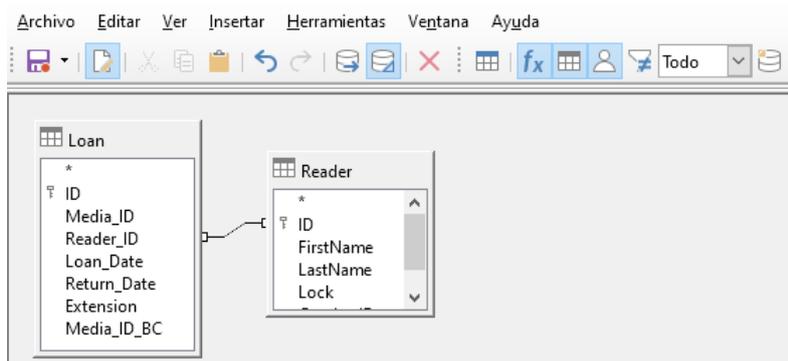
Cuando desee usar el orden descendente en modo SQL, use **DESC** en lugar de **ASC**.

Hasta ahora, los campos *Media_ID* y *Reader_ID* solo son visibles como campos numéricos. Los nombres de los lectores no están claros. Para mostrarlos en una consulta, se debe incluir la tabla *Reader*. Para este propósito volvemos al modo de diseño, y agregamos una nueva tabla a la vista *Diseño*.



Mediante el botón *Añadir tabla o consulta* se pueden agregar más tablas o consultas y hacerlas visibles en la interfaz gráfica de usuario. Si los enlaces entre las tablas ya se declararon en el

momento de su creación (consulte el «Capítulo 3, Tablas»), estas tablas se muestran con los enlaces directos correspondientes.



Si no hay un enlace, se puede crear en este momento arrastrando el ratón desde un campo de una tabla hasta otro de otra tabla. Crearemos un enlace entre el campo *Reader_ID* de la tabla *Loan* y el campo *ID* de la tabla *Reader*.



Nota

La vinculación de tablas de momento solo funciona en la base de datos interna o en bases de datos relacionales externas. Por ejemplo, las tablas de una hoja de cálculo no se pueden vincular entre sí. Primero tienen que importarse a una base de datos interna.

Para crear un enlace entre las tablas, basta con una simple importación, sin necesidad de crear una clave primaria.

Ahora los campos de la tabla *Reader* se pueden incorporar en el área tabular. Los campos se agregan al final de la consulta.

La posición de los campos se puede corregir en el área tabular del editor con el ratón. Así, por ejemplo, el campo *FirstName* se ha arrastrado a su posición directamente antes del campo *Loan_Date*.

	←			
Reader_ID	Loan_Date	Return_Date	FirstName	LastName
	Date of Issue	Return Date		
Loan	Loan	Loan	Reader	Reader

	ID	Media_ID	Reader_ID	FirstName	LastName	Date of Issue	Return Date
22	2	1	Heinrich	Müller	04/03/13		
24	8	1	Heinrich	Müller	12/03/13		
26	5	1	Heinrich	Müller	29/03/13		
28	3	6	Greta	Garbo	01/04/13		
29	4	6	Greta	Garbo	04/04/13		
21	0	9	Terence	Nobody	04/03/13		

Registro | de 6

Ahora los nombres son visibles. El *Reader_ID* se ha vuelto superfluo, con lo que se eliminará. Tampoco tiene sentido ordenar por *Reader_ID*. Se ordenará por Apellido y luego Nombre.

Esta consulta ya no es adecuada para su uso como una consulta que permita nuevas entradas en la tabla resultante, ya que carece de la clave principal para la tabla *Reader* agregada.

Solo si esta clave principal está integrada, la consulta se puede editar de nuevo. De hecho, se puede editar totalmente, los nombres de los lectores también se pueden modificar. Por esta razón,

hacer que los resultados de la consulta se puedan editar es una facilidad que debe usarse con extrema precaución, si es necesario bajo el control de un formulario.



Precaución

Una consulta editable puede crear problemas. La edición de datos en la consulta también edita los datos de la tabla subyacente y los registros de la tabla. Al editar los datos se modifica su significado. Por ejemplo, al cambiar el ID de un lector los artículos que el lector había tomado prestados y devuelto no quedan a su nombre.

Si tiene que facilitar la edición de datos, hágalo de forma que pueda ver los efectos de la edición de los datos editados.

Incluso aunque una consulta puede editarse, no es tan fácil de usar como un formulario con controles de listado, (estos controles pueden mostrar los nombres de los lectores, pero realmente contienen el *Reader_ID* de la tabla). Los controles de listado no se pueden agregar a una consulta; solo se pueden usar en formularios.

```
SELECT "Loan"."ID", "Loan"."Media_ID", "Loan"."Reader_ID",  
"Reader"."FirstName", "Reader"."LastName",  
"Loan"."Loan_Date" AS "Date or Issue", "Loan"."Return_Date" AS "Return Date"  
FROM "Loan", "Reader"  
WHERE "Loan"."Reader_ID" = "Reader"."ID" AND "Loan"."Return_Date" IS NULL  
ORDER BY "Loan"."Reader_ID" ASC, "Date or Issue" ASC
```

Si ahora volvemos a la Vista SQL, vemos que todos los campos ahora se muestran entre comillas dobles y los nombres de los campos van precedidos del nombre de la tabla y un punto: ["Nombre_de_tabla"."Nombre_de_campo"]. Esto es necesario para que la base de datos sepa de qué tabla provienen los campos previamente seleccionados. Después de todo, los campos en diferentes tablas pueden tener fácilmente los mismos nombres de campo. En la estructura de la tabla anterior, esto se nota claramente en el campo ID.



Nota

La siguiente consulta funciona sin poner nombres de tabla delante de los nombres de campo:

```
SELECT "ID", "Number", "Price" FROM "Stock", "Dispatch" WHERE "Dispatch"."stockID" =  
"Stock"."ID"
```

ID se toma de la tabla que aparece primero en la definición FROM. La definición de la tabla en la Fórmula WHERE también es superflua, porque *stockID* solo aparece una vez (en la tabla *Dispatch*) y la *ID* se tomó claramente de la tabla *Stock* (de la posición de la tabla en la consulta).

Si un campo en la consulta tiene un alias, puede referirse a él, por ejemplo, en la ordenación, sin este nombre de tabla.

La ordenación se lleva a cabo en la interfaz gráfica de acuerdo con la secuencia de campos en la vista tabular. Si, desea ordenar primero por *Loan_Date* y luego por *Reader_ID* (de la tabla *Loan*), puede hacerlo usando una de las siguientes maneras:

- Cambiando la secuencia de campos en el área de la tabla de la interfaz gráfica de usuario (arrastre y suelte *Loan_Date* a la izquierda de *Reader_ID*)
- Agregando un campo adicional, solo para la ordenación, configurado para que sea invisible (el editor lo registrará solo temporalmente si no se definió ningún alias).

Ej.: agregue otro campo *Loan_Date* justo antes de *Reader_ID* o bien agregue otro campo *Reader_ID* justo después de *Loan_Date*

- O modificando el texto para la orden ORDER BY en el editor de SQL de manera adecuada: [ORDER BY "Loan_Date", "Loan"."Reader_ID"].



Consejo

Una consulta puede requerir un campo que no sea parte del resultado de la consulta. En el gráfico de la siguiente sección, *Return_Date* es un ejemplo. Esta consulta está buscando registros que no contengan una fecha de retorno. Este campo proporciona un criterio para la consulta, pero no proporciona datos visibles útiles.

Usar funciones en una consulta

El uso de funciones permite que una consulta proporcione más que una vista filtrada de los datos en una o más tablas. La siguiente consulta calcula cuántos artículos se han prestado, según el *Reader_ID*.

Campo	ID	Reader_ID	Return_Date
Alias	Count		
Tabla	Loan	Loan	Loan
Orden			
Visible	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Función	Recuento	Agrupar	
Criterio			IS EMPTY
o			

Figura 4: Funciones en consulta

Para la *ID* de la tabla *Loan*, se selecciona la función *Recuento*. En principio, no importa qué campo de una tabla se elija para ello. La única condición es que el campo no debe estar vacío en ninguno de los registros. Por esta razón, el campo de clave principal, que nunca está vacío, es la opción más adecuada. (Se cuentan todos los campos con un contenido que no sea *NULL*).

Para el *Reader_ID*, que da acceso a la información del lector, se elige la función *Agrupar*. De esta forma, los registros con el mismo *Reader_ID* se agrupan. El resultado muestra el número de registros para cada *Reader_ID*. Como criterio de búsqueda, *Return_Date* se establece como *IS EMPTY*, igual que en el ejemplo anterior.

	Count	Reader_ID
1	9	
3	1	
2	6	

Registro | de 3

```
SELECT COUNT( "ID" ) "Count", "Reader_ID"
FROM "Loan" WHERE "Return_Date" IS NULL
GROUP BY "Reader_ID"
```

- Agrupar
- (sin función)
- Promedio
- Recuento
- Máximo
- Mínimo
- Suma
- Cada
- Cualquiera
- Alguno
- STDDEV_POP
- STDDEV_SAMP
- VAR_SAMP
- VAR_POP
- Collect
- Fusión
- Intersección
- Agrupar

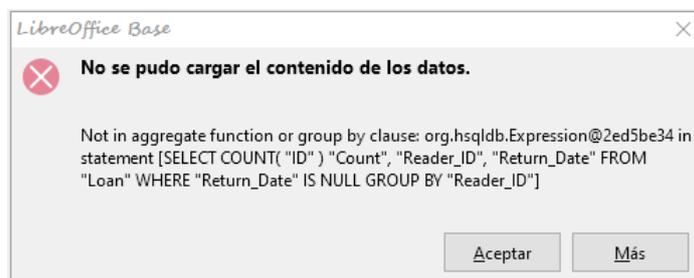
El resultado de la consulta muestra que *Reader_ID* 1 tiene un total de 3 artículos prestados.

Si la función *Recuento* (COUNT) se hubiera asignado a *Return_Date* en lugar de *ID*, cada *Reader_ID* tendría un artículo 0 en préstamo, ya que *Return_date* se ha definido como *NULL*.

La fórmula correspondiente en el código SQL se muestra arriba. La interfaz gráfica de usuario proporciona las funciones que se muestran a la derecha, que corresponden a funciones HSQLDB.

Consulte «Mejora de consultas usando el modo SQL» en la página 19, para obtener una explicación de las funciones.

Si un campo en una consulta está asociado con una función, el resto de los campos mencionados en la consulta también deben estar asociados con funciones. Si esto no es así, recibirá el siguiente mensaje de error:



En la figura 4 el campo *Return_Date* no tiene función asociada y se produce dicho error. Una traducción algo libre sería: La siguiente expresión no contiene ninguna función agregada o agrupación.

Consejo

Cuando se utiliza el *Modo de vista de diseño*, un campo solo es visible si la fila *Visible* contiene una marca de verificación para el campo. Cuando se utiliza el modo SQL, un campo solo es visible cuando sigue a la palabra clave SELECT.

Nota

Cuando un campo no está asociado con una función, el número de filas en la salida de la consulta está determinado por las condiciones de búsqueda.

Cuando un campo está asociado con una función, el número de filas en la salida de la consulta se determina si hay alguna agrupación o no. Si no hay agrupación, solo se verá una fila en la salida de la consulta. Si hay agrupación, el número de filas coincide con el número de valores distintos que tiene el campo de agrupación. Por lo tanto, todos los campos visibles deben estar asociados con una función o no estar asociados con ninguna para evitar el conflicto en la salida de la consulta.

Después de esto, el código de la consulta aparece en el mensaje de error, pero sin referencia específica al campo ofensivo. En este caso, el campo *Return_Date* se ha agregado como un campo visible. Este campo no tiene ninguna función asociada y tampoco está incluido en la declaración de agrupación.

La información proporcionada al pulsar el botón *Más* del mensaje de error no es muy esclarecedora para el usuario final de la base de datos. Simplemente muestra el código de error de SQL.

Para corregir el error, elimine la marca de verificación en la fila «Visible» del campo *Return_Date*. Aunque su condición de búsqueda (Criterio) se aplica cuando se ejecuta la consulta, no es visible en el resultado.

Usando la Interfaz se pueden aplicar cálculos básicos y funciones adicionales.

Suponga que una biblioteca no emite avisos cuando un artículo debe devolverse, sino que emite avisos de retraso cuando el período del préstamo haya expirado y el artículo no haya sido devuelto. Esta es una práctica común en las bibliotecas escolares y públicas que hacen préstamos

solo por períodos cortos y fijos. En este caso, la emisión de un aviso de retraso significa que se debe pagar una multa. ¿Cómo calculamos estas multas?

Campo	ID	Media_ID	Reader_ID	Date	Recuento("Recall"."Date") * 2	Return_Date
Alias				RecallCount	RecallAmmount	
Tabla	Loan	Loan	Loan	Recall		Loan
Orden						
Visible	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
Función	Agrupar	Agrupar	Agrupar	Recuento		
Criterio						IS EMPTY

En la consulta que se muestra, las tablas *Loan* y *Recall* se consultan conjuntamente. A partir del recuento de las entradas de datos en la tabla *Recall*, se determina el número total de avisos de retraso. La multa por los artículos vencidos se establece en la misma consulta en 2,00 € y se aplica creando un nuevo campo. En lugar de un nombre de campo, se utiliza la fórmula correspondiente: «Recuento("Recall"."Date")*2».

La interfaz gráfica de usuario agrega las comillas y convierte el término «Recuento» en la orden SQL apropiada.



Precaución

Solo para usuarios que utilizan la coma para su separador decimal:

Si desea ingresar números con lugares decimales utilizando la Interfaz, debe asegurarse de que usa un punto decimal en lugar de una coma como separador decimal dentro de la instrucción SQL. Las comas se utilizan como separadores de campo, por lo que si utiliza comas, se crearán nuevos campos de consulta para la parte decimal.

Una entrada con una coma en la vista SQL siempre conduce a un campo adicional que contiene el valor numérico de la parte decimal.

ID	Media_ID	Reader_ID	RecallCount	RecallAmount
24	8	1	1	2
22	2	1	1	2

Registro 1 de 2

```

SELECT "Loan"."ID", "Loan"."Media_ID", "Loan"."Reader_ID",
COUNT( "Recall"."Date" ) "RecallCount",
COUNT( "Recall"."Date" ) * 2 "RecallAmount"
FROM "Recall", "Loan"
WHERE "Recall"."Loan_ID" = "Loan"."ID"
AND "Loan"."Return_Date" IS NULL
GROUP BY "Loan"."ID", "Loan"."Media_ID", "Loan"."Reader_ID", "Loan"."Return_Date"
    
```

La consulta ahora arroja para cada artículo aún en préstamo las multas que se han acumulado, según los avisos de retraso acumulados y el campo de multiplicación adicional. La siguiente estructura de consulta también será útil para calcular las multas adeudadas por usuarios individuales.

Campo	Reader_ID	Date	Recuento("Recall"."Date") * 2	Return_Date
Alias		RecallCount	RecallAmount	
Tabla	Loan	Recall		Loan
Orden				
Visible	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Función	Agrupar	Recuento		
Criterio				IS EMPTY

Se han eliminado los campos *Loan.ID* y *Loan.ID_Media*. Se usaron en la consulta anterior para crear un registro separado para cada artículo mediante una agrupación. Ahora se agrupará solo por el lector. El resultado de la consulta se ve de la siguiente manera:

	Reader_ID	RecallCount	RecallAmount
▶	1	2	4

Registro | de 1

En lugar de enumerar los artículos para *Reader_ID* = 1 por separado, se contaron todos los campos "Recall". "Date" y se calculó un total de 4.00 € como la multa debida.

Definir relaciones en la consulta

Cuando se buscan datos en tablas o formularios, la búsqueda generalmente se limita a una tabla o un formulario. La función de búsqueda incorporada no puede recorrer los enlaces de un formulario principal con un subformulario. Para tal fin, los datos que se quieren buscar se recopilan mejor mediante una consulta.

	Title
▶	Der kleine Hobbit
	Das sogenannte Böse
	Eine kurze Geschichte der Zeit
	Traditionelle und kritische Theorie
	Die neue deutsche Rechtschreibung
	I hear you knocking
	Datenbanken mit OpenOffice.org 3
	Das Postfix-Buch
	Im Augenblick

Registro | 1 | de 9

Campo	Title
Alias	
Tabla	Media
Orden	
Visible	<input checked="" type="checkbox"/>
Función	
Criterio	

	Title	Subtitle
▶	I hear you knocking	Youn can't catch me
	I hear you knocking	The stumble
	I hear you knocking	Sabre dance (Single version)
	Im Augenblick	Amsterdam
	Im Augenblick	Hier unten am Deich
	Im Augenblick	Köln-Ehrenfeld
	Im Augenblick	Bei Mir
	Im Augenblick	Gott sei Dank

Registro | de 8

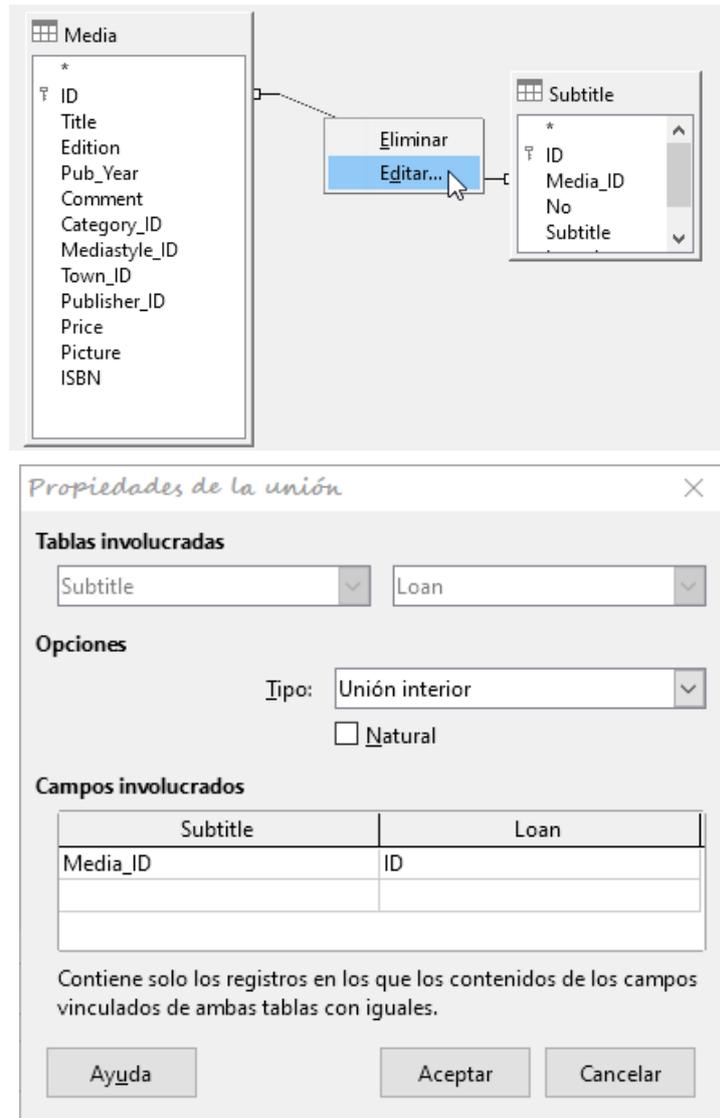
Campo	Title	Subtitle
Alias		
Tabla	Media	Subtitle
Orden		
Visible	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Función		
Criterio		

Una consulta sencilla para el campo *Título* de la tabla *Media* arroja un resultado de: 9 registros en total (los que tienen un título). Pero si se añade la tabla *Subtitle* y el campo *Subtitle* de esta tabla

en la consulta, el resultado de la consulta se reduce a solo 2 Títulos. Solamente estos dos artículos tienen también subtítulos. El resto de los registros, aunque tienen título, no tienen subtítulo.

Este resultado se debe a la relación de unión entre tablas. En la consulta, la unión está establecida para que solo se muestren aquellos registros en los que el campo *Media_ID* de la tabla *Subtitle* sea igual al campo *ID* de la tabla *Media*. Los demás registros quedan excluidos.

Las condiciones de unión se deben editar para poder mostrar los registros deseados. Nos referimos aquí no a uniones en el diseño de relación entre tablas, sino a uniones dentro de consultas.



De manera predeterminada, las relaciones se establecen como uniones interiores. El diálogo proporciona información sobre cómo funciona este tipo de combinación.

Las dos tablas previamente seleccionadas se enumeran como tablas involucradas que no se pueden seleccionar aquí.

Los campos relevantes de las dos tablas se leen de las definiciones de tabla. Si no hay una relación específica en las definiciones de las tablas, se puede crear una en este momento para la consulta. Sin embargo, si ha planificado su base de datos de manera ordenada utilizando HSQLDB, no debería ser necesario modificar estos campos.

La configuración más importante es el tipo de unión. Las relaciones se pueden establecer de modo que se seleccionen todos los registros de la tabla *Subtitle*, pero solo aquellos registros de la tabla *Media* que tengan un subtítulo.

O por el contrario, que se muestren todos los registros de la tabla *Media*, independientemente de si tienen o no un subtítulo.

La opción de la casilla de verificación *Natural* especifica que los campos vinculados en las tablas se tratan como iguales. También puede evitar el uso de esta configuración definiendo sus relaciones correctamente al comienzo de la planificación de su base de datos.

Para el tipo *Unión a la derecha*, la descripción apunta que se mostrarán todos los registros de la tabla *Media* [Subtitle RIGHT JOIN Media]. Como no hay subtítulos que carezcan de un título en los artículos, pero sí hay títulos en los artículos que carecen de subtítulos, esta es la opción correcta.



Después de confirmar la unión correcta, los resultados de la consulta se ven como queríamos. Título y subtítulo se muestran juntos en la consulta. Naturalmente, los títulos aparecen más de una vez como en la relación anterior. Sin embargo, siempre que no se cuenten los resultados, esta consulta se puede utilizar como base para funciones de búsqueda. Consulte los fragmentos de código que se utilizan en este capítulo, en el «Capítulo 8, Tareas de base de datos» y en el «Capítulo 9, Macros».

	Title	Subtitle
▶	Der kleine Hobbit	
	Das sogenannte Böse	
	Eine kurze Geschichte der Zeit	
	Traditionelle und kritische Theorie	
	Die neue deutsche Rechtschreibung	
	I hear you knocking	Youn can't catch me
	I hear you knocking	The stumble
	I hear you knocking	Sabre dance (Single version)
	Datenbanken mit OpenOffice.org 3	
	Das Postfix-Buch	
	Im Augenblick	Amsterdam
	Im Augenblick	Hier unten am Deich
	Im Augenblick	Köln-Ehrenfeld
	Im Augenblick	Bei Mir
	Im Augenblick	Gott sei Dank

Registro 1 de 15

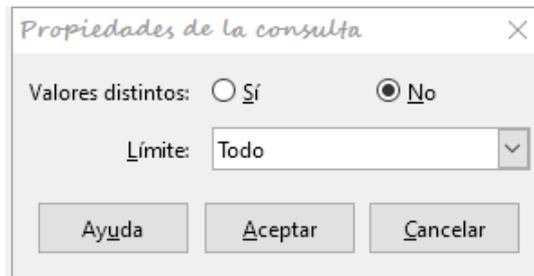
Definir propiedades de consulta

A partir de la versión 4.1 de LibreOffice, es posible definir propiedades adicionales en el editor de consultas.



Figura 5: Abrir propiedades de la consulta en el editor de consultas

Al la izquierda del botón para abrir las *Propiedades de la consulta* hay un Control de lista: *Límite* para regular la cantidad de registros que se muestran, así como un botón para *Valores distintos*. Estas opciones también están presentes en el diálogo *Propiedades de la consulta*:



La configuración de *Valores distintos* determina si la consulta debe suprimir registros duplicados.

	FirstName	LastName	Return_Date
▶	Greta	Garbo	
	Greta	Garbo	
	Lisa	Gerd	
	Heinrich	Müller	
	Heinrich	Müller	
	Heinrich	Müller	
	Terence	Nobody	

Supongamos que se realiza una consulta para determinar qué lectores aún tienen artículos prestados. Sus nombres se muestran si el campo de fecha de retorno está vacío. Los nombres se muestran más de una vez para los lectores que tienen varios artículos prestados.

Si elige *Valores distintos*, los registros con el mismo contenido desaparecerán. La consulta se verá así:

	FirstName	LastName	Return_Date
▶	Greta	Garbo	
	Lisa	Gerd	
	Heinrich	Müller	
	Terence	Nobody	

```
SELECT DISTINCT
"Reader"."FirstName", "Reader"."LastName", "Loan"."Return_Date"
FROM "Loan", "Reader"
WHERE "Loan"."Reader_ID" = "Reader"."ID" AND "Loan"."Return_Date" IS NULL
ORDER BY "Reader"."LastName" ASC
```

La consulta original:

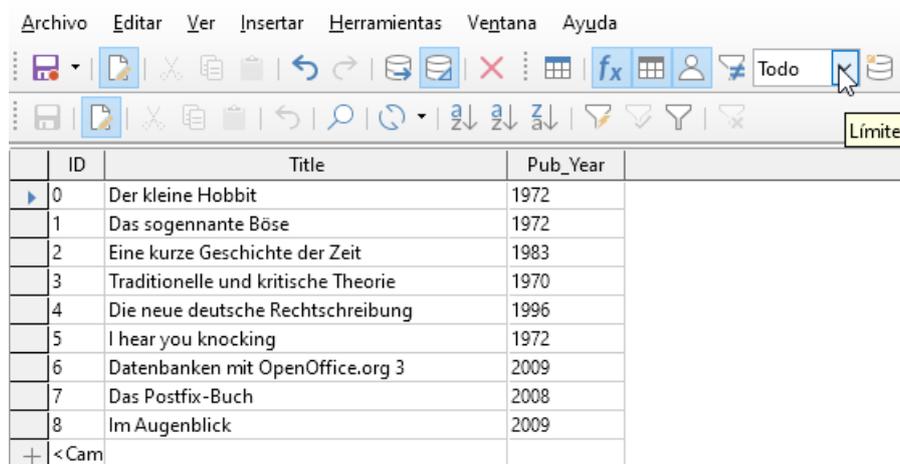
```
SELECT "Reader"."FirstName", "Reader"."LastName" ...
```

Agregar `DISTINCT` a la consulta suprime los registros duplicados.

```
SELECT DISTINCT "Reader"."FirstName", "Reader"."LastName" ...
```

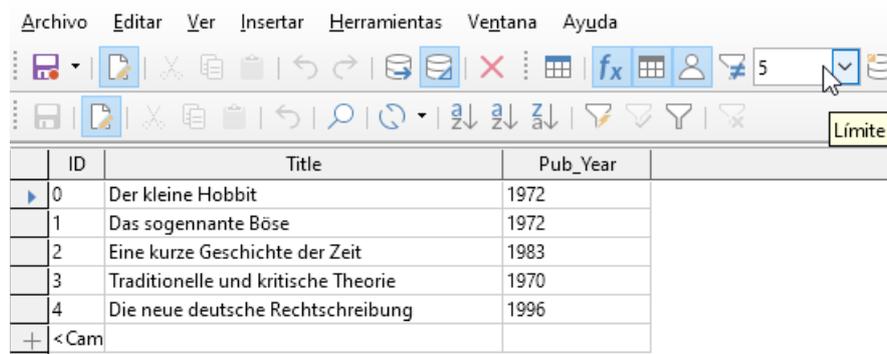
La capacidad de especificar registros únicos también estuvo presente en versiones anteriores. Sin embargo, era necesario cambiar de la vista de diseño a la vista SQL para insertar el calificador DISTINCT. Esta propiedad es compatible con versiones anteriores de LibreOffice.

La configuración del *Límite* determina cuántos registros se mostrarán en la consulta.



ID	Title	Pub_Year
0	Der kleine Hobbit	1972
1	Das sogenannte Böse	1972
2	Eine kurze Geschichte der Zeit	1983
3	Traditionelle und kritische Theorie	1970
4	Die neue deutsche Rechtschreibung	1996
5	I hear you knocking	1972
6	Datenbanken mit OpenOffice.org 3	2009
7	Das Postfix-Buch	2008
8	Im Augenblick	2009
+ <Cam		

Figura 6: Se muestran todos los registros en la tabla Media. La consulta se puede editar, ya que incluye la clave primaria.



ID	Title	Pub_Year
0	Der kleine Hobbit	1972
1	Das sogenannte Böse	1972
2	Eine kurze Geschichte der Zeit	1983
3	Traditionelle und kritische Theorie	1970
4	Die neue deutsche Rechtschreibung	1996
+ <Cam		

Solo se muestran los primeros cinco registros (*ID* 0-4). No se solicitó una ordenación, por lo que se utiliza el orden predeterminado (por clave principal). A pesar de la limitación en la salida, la consulta se puede editar. Esto distingue la entrada en la interfaz gráfica de lo que, en versiones anteriores, solo era accesible usando SQL.

```
SELECT "ID", "Title", "Pub_Year" FROM "Loan" LIMIT 5
```

Simplemente se ha agregado `LIMIT 5` a la consulta original. El tamaño del límite puede ser el que se necesite.



Precaución

Establecer límites en la interfaz gráfica no es compatible con versiones anteriores. En las versiones de LibreOffice anteriores a la 4.1, un límite solo se podía establecer en modo SQL directo. El límite entonces requería una clasificación (`ORDER BY...`) o una condición (`WHERE...`).

Mejora de consultas usando el modo SQL

Si durante la entrada gráfica usa **Ver > Activar o desactivar la vista Diseño** para desactivar la vista Diseño, verá el editor de SQL en lugar de la vista Diseño. Esta es la mejor manera para que los principiantes aprendan el lenguaje de consulta estándar para bases de datos. A veces, también

es la única forma de realizar una consulta en la base de datos cuando la interfaz no puede traducir sus requisitos a las órdenes SQL necesarias.

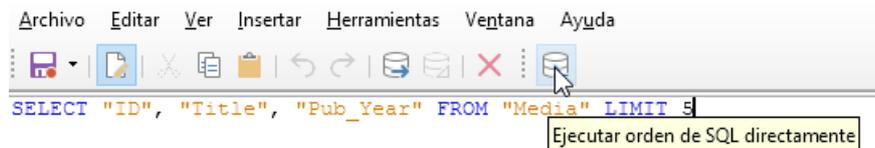
```
SELECT * FROM "Nombre_Tabla"
```

Esto mostrará todo lo que está en la tabla indicada. El asterisco "*" presenta todos los campos de la tabla.

```
SELECT * FROM "Nombre_Tabla" WHERE "Nombre_Campo" = 'Karl'
```

Aquí hay una restricción significativa. Solo se muestran aquellos registros para los que el campo "Nombre_Campo" contiene el término exacto «Karl», no mostrará, por ejemplo, «Karl Egon».

A veces, las consultas en Base no se pueden llevar a cabo utilizando la interfaz, ya que puede que no sean reconocidas algunas órdenes SQL concretas. En estos casos, es necesario desactivar la vista de *Diseño* y usar **Editar > Ejecutar orden de SQL directamente** para acceder a la base de datos. Este método tiene la desventaja de que solo se puede trabajar con la consulta en modo *editor de SQL*.



El uso directo de las órdenes SQL también es accesible mediante la interfaz gráfica de usuario, como muestra la figura anterior. Haga clic en el icono resaltado *Ejecutar orden de SQL directamente* una vez desactivada la vista *Diseño*. Ahora, cuando hace clic en el icono *Ejecutar*, la consulta ejecuta las órdenes SQL directamente.

A continuación se muestran las amplias posibilidades disponibles para plantear consultas a la base de datos y especificar el tipo de resultado requerido:

```
SELECT [{LIMIT <offset> <límite> | TOP <límite>}][ALL | DISTINCT]
{ <Formulación_para_select> | "Nombre_Tabla".* | * } [, ...]
[INTO [CACHED | TEMP | TEXT] "nueva_Tabla"]
FROM "Lista_de_Tablas"
[WHERE Expresión_SQL]
[GROUP BY Expresión_SQL [, ...]]
[HAVING Expresión_SQL]
[ { UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT] } |
INTERSECT [DISTINCT] } Declaración_consulta]
[ORDER BY Criterio_de_Ordenación [, ...]]
[LIMIT <límite> [OFFSET <offset>]];
```

[{LIMIT <offset> <límite> | TOP <límite>}]:

Esto limita el número de registros que se mostrarán:

LIMIT 10 20 comienza en el undécimo registro y muestra los siguientes 20 registros.

TOP 10 siempre muestra los primeros 10 registros. Es lo mismo que LIMIT 0 10.

LIMIT 10 0 omite los primeros 10 registros y muestra todos los registros a partir del 11.

Puede hacer lo mismo usando la condición SELECT En la fórmula [LIMIT <límite> [OFFSET <offset>]]. LIMIT 10 muestra solo 10 registros. Agregar OFFSET 20 hace que comience en el registro n.º 21. Esta forma final de limitación de visualización requiere una instrucción de clasificación (ORDER BY...) o una condición (WHERE...).

Todos los valores empleados en el límite deben ser de tipo *integer*. No es posible reemplazar una entrada por una subconsulta para que, por ejemplo, se muestren los cinco últimos registros de una serie.

[ALL | DISTINCT]

SELECT ALL es el valor predeterminado. Se muestran todos los registros que cumplen las condiciones de búsqueda. Ejemplo: SELECT ALL "Nombre" FROM "Tabla_nombres" produce todos los nombres; si "Peter" aparece tres veces y "Egon" cuatro veces en la tabla, estos nombres se muestran tres y cuatro veces respectivamente. SELECT DISTINCT "Nombre" FROM "Tabla_nombres" suprime los resultados de la consulta que tienen el mismo contenido. En este caso, "Peter" y "Egon" aparecen solo una vez. DISTINCT se refiere a la totalidad del campo al que accede la consulta. Si, por ejemplo, también se solicita el apellido, los registros de "Peter Müller" y "Peter Maier" contarán como distintos. Incluso si especifica la condición DISTINCT, se mostrarán ambas.

<Formulación para select>

```
{ Expresión | COUNT(*) |  
{ COUNT | MIN | MAX | SUM | AVG | SOME | EVERY | VAR_POP | VAR_SAMP | STDDEV_POP |  
STDDEV_SAMP }  
([ALL | DISTINCT]) Expresión } [[AS] "alias_a_mostrar"]
```

Los nombres de campo, los cálculos, los totales de registros son todas las entradas posibles.



Nota

Los cálculos dentro de una base de datos a veces conducen a resultados inesperados. Suponga que una base de datos contiene las calificaciones otorgadas por algún trabajo de clase y desea calcular la calificación promedio.

Primero deberá sumar todas las calificaciones. Supongamos que la suma es 80. Ahora debe dividirse por el número de alumnos (digamos 30). La consulta produce 2.

Esto ocurre porque está trabajando con campos de tipo INTEGER. Por lo tanto, el resultado del cálculo produce un valor entero. Debe tener al menos un valor del tipo DECIMAL en su cálculo. Puede lograr esto utilizando una función de conversión HSQLDB o (simplemente) puede dividir entre 30.0 en lugar de 30. Esto dará un resultado con un decimal. Dividir por 30.00 da dos lugares decimales.

Tenga el cuidado de usar puntos para la separación decimal (estilo inglés). El uso de comas en las consultas está reservado para los separadores de campo.

Además, hay diferentes funciones disponibles para el campo que se muestra. Excepto COUNT(*) (que cuenta todos los registros) ninguna de las siguientes funciones accede a los campos NULL.

```
COUNT | MIN | MAX | SUM | AVG | SOME | EVERY | VAR_POP | VAR_SAMP | STDDEV_POP |  
STDDEV_SAMP
```

COUNT("Nombre") cuenta todas las entradas para el campo Nombre.

MIN("Nombre") muestra el primer nombre alfabéticamente. El resultado de esta función siempre se formatea tal como esté en el campo. El texto se muestra como texto, los enteros como enteros, los decimales como decimales, etc.

MAX ("Nombre") muestra el último nombre alfabéticamente.

SUM ("Número") puede sumar solo valores en campos numéricos. La función falla en campos de fecha u hora.



Consejo

Aunque la función SUM falla en los campos de hora, se puede hacer efectiva empleando la siguiente fórmula:

```
SELECT (SUM( HOUR("Time") ) * 3600 + SUM( MINUTE("Time") )) * 60 +  
SUM( SECOND("Time") ) AS "seconds" FROM "Table"
```

La suma mostrada se compone de horas, minutos y segundos separados. Las horas y los minutos se convierten a segundos. Con lo que se obtiene el total en segundos.

Añadiendo una división por 3600 obtendrá un resultado horario en horas con los minutos y segundos como lugares decimales.

```
SELECT ((SUM( HOUR("Time") ) * 3600 + SUM( MINUTE("Time") )) * 60 +  
SUM( SECOND("Time") )) / 3600.0000 AS "hours" FROM "Table"
```

Con un formato adecuado, se puede convertir a un valor horario correcto en una consulta o formulario.

AVG ("Número") muestra el promedio del contenido de una columna. Esta función también está limitada a campos numéricos.

SOME("Nombre_Campo"), EVERY ("Nombre_Campo"): Los campos utilizados con estas funciones deben tener el tipo de campo Sí / No [BOOLEAN] (contiene solo 0 o 1). Además, producen un resumen del contenido del campo al que se aplican.

SOME devuelve TRUE (o 1) si al menos una entrada para el campo es 1, y devuelve FALSE (o 0) solo si todas las entradas son 0.

EVERY devuelve TRUE solo si todos los valores en el campo son 1, y devuelve FALSE si al menos un valor es 0.



Consejo

El tipo de campo booleano es Sí / No [BOOLEAN]. Este campo solo almacena 0 o 1. (equivalentes a FALSE y TRUE). No se almacena SI o NO.

Si utiliza la búsqueda en consultas, utilice TRUE o 1 (para SI), o bien FALSE o 0 (para NO). Si intenta utilizar «Sí» o «No» no encontrará nada.

Ejemplo:

```
SELECT "Clase", EVERY("Nadador")  
FROM "Tabla1"  
GROUP BY "Clase"
```

En una tabla *Tabla1* tenemos los alumnos de la escuela, esta tabla tiene varios campos:

Clase que contiene las clases a que pertenecen los alumnos, *Alumno* contiene los nombres de los estudiantes, *Nadador* es un campo booleano que describe si un estudiante sabe nadar o no (1 o 0). y una clave principal. *ID*

Solo *Clase* y *Nadador* están incluidos en esta consulta.

Debido a que la consulta está agrupada por las entradas del campo *Clase*, **EVERY** devolverá el valor del campo, *Nadador*, para cada clase. Si todos los alumnos de una clase saben nadar, **EVERY** devolverá **TRUE**. Si al menos un alumno de la clase no sabe nadar, **EVERY** devuelve **FALSE**.

En el resultado de la consulta, el campo *Nadador* es una casilla de verificación, una marca en esta casilla indica **TRUE** y ninguna marca **FALSE**.

VAR_POP | VAR_SAMP | STDDEV_POP | STDDEV_SAMP

Son funciones estadísticas y solo afectan a los campos enteros y decimales.

Todas estas funciones devuelven 0, si los valores dentro del grupo son todos iguales. Las funciones estadísticas son incompatibles con **DISTINCT**. Básicamente, calculan sobre todos los valores de los registros en la consulta, y **DISTINCT** excluye registros con los mismos valores con lo que el resultado sería erróneo.

[AS] "alias_a mostrar": Devuelve el alias del campo indicado dentro de la consulta.

"Nombre_Tabla".* | * [, ...]

Cada campo a mostrar se proporciona con sus nombres de campo, separados por comas. Si se utilizan campos de varias tablas en la consulta, es necesaria una combinación del nombre de la tabla con el nombre del campo: "Nombre_Tabla"."Nombre_campo".

En lugar de una lista detallada de todos los campos de una tabla, Puede mostrar su contenido total utilizando el símbolo asterisco «*», de este modo no es necesario usar el nombre de la tabla si los resultados solo se aplican a una tabla. Sin embargo, si la consulta incluye todos los campos de una tabla y al menos un campo de una segunda tabla, use:

"Nombre_Tabla1".*, "Nombre_Tabla2"."Nombre_Campo"

[INTO [CACHED | TEMP | TEXT] "nueva_tabla"]

El resultado de esta consulta se debe escribir directamente en una tabla nueva con el nombre indicado. Las propiedades de campo para la nueva tabla se definen a partir de las definiciones de campo contenidas en la consulta. Escribir en una tabla nueva no funciona desde el modo **SQL**, ya que solo maneja los resultados que se pueden mostrar. En su lugar, debe usar **Herramientas > SQL**. La tabla resultante inicialmente no es editable, ya que carece de una clave primaria.

FROM <Lista_de_Tablas>

"Nombre_Tabla1" [{**CROSS** | **INNER** | **LEFT OUTER** | **RIGHT OUTER**} **JOIN** "Nombre_Tabla2"
ON Expresión] [, ...]

Las tablas que se deben buscar conjuntamente generalmente están en una lista separada por comas. La relación de las tablas entre sí se define con la palabra clave **WHERE**.

Si las tablas están vinculadas a través de **JOIN** en lugar de una coma, su relación se define mediante el término que precede inmediatamente a **ON**, después de la segunda tabla.

Un simple **JOIN** tiene el efecto de mostrar solo aquellos registros para los que se aplican las condiciones de ambas tablas.

Ejemplo:

```
SELECT "Tabla1"."Nombre", "Tabla2"."Clase"  
FROM "Tabla1", "Tabla2"  
WHERE "Tabla1"."ClassID" = "Tabla2"."ID"
```

Equivale a:

```
SELECT "Tabla1"."Nombre", "Tabla2"."Clase" FROM "Tabla1"  
JOIN "Tabla2"
```

```
ON "Tabla1"."ClassID" = "Tabla2"."ID"
```

Aquí se muestran los nombres y las clases correspondientes. Si un nombre no tiene una clase en la lista, ese nombre no se incluye en el resultado. Si una clase no tiene nombres, tampoco se muestra. La incorporación de INNER no altera esto.

```
SELECT "Tabla1"."Nombre", "Tabla2"."Clase"  
FROM "Tabla1"  
LEFT JOIN "Tabla2"  
ON "Tabla1"."ClassID" = "Tabla2"."ID"
```

Si se agrega LEFT, todos los nombres de Tabla1 se muestran incluso si no tienen clase. Si, por el contrario, se agrega RIGHT, todas las clases se muestran incluso si no tienen nombres en ellas. La adición de OUTER no necesita usarse aquí. (RIGHT OUTER JOIN es lo mismo que RIGHT JOIN, lo mismo ocurre con LEFT OUTER JOIN).

```
SELECT "Tabla1"."Jugador1", "Tabla2"."Jugador2"  
FROM "Tabla1" AS "Tabla1"  
CROSS JOIN "Tabla2" AS "Tabla1"  
WHERE "Tabla1"."Jugador1" <> "Tabla2"."Jugador2"
```

Un CROSS JOIN requiere que la tabla se suministre con un alias, pero no siempre es necesario agregar el término ON. Todos los registros de la primera tabla se emparejan con todos los registros de la segunda tabla. La consulta anterior produce todos los emparejamientos posibles de registros de la primera tabla con los de la segunda tabla, excepto los emparejamientos entre registros para el mismo jugador. En el caso de CROSS JOIN, la condición no debe incluir un enlace entre las tablas especificadas en el término ON. En cambio, las condiciones WHERE se pueden usar. Si las condiciones se formulan exactamente como en el caso de un simple JOIN, obtendrá el mismo resultado:

```
SELECT "Tabla1"."Nombre", "Tabla2"."Clase"  
FROM "Tabla1"  
JOIN "Tabla2"  
ON "Tabla1"."ClaseID" = "Tabla2"."ID"
```

da el mismo resultado que:

```
SELECT "Tabla1"."Nombre", "Tabla2"."Class"  
FROM "Tabla1" AS "Tabla1"  
CROSS JOIN "Tabla2" AS "Tabla2"  
WHERE "Tabla1"."ClaseID" = "Tabla2"."ID"
```

[WHERE Expresión SQL]

La introducción estándar de las condiciones para solicitar un filtrado más preciso de los datos. Aquí generalmente se definen también las relaciones entre las tablas si no están ya enlazadas con JOIN.

[GROUP BY Expresión SQL [, ...]]

Se usa cuando se desea dividir los datos de la consulta en grupos antes de aplicar las funciones para cada uno de los grupos por separado. La división se basa en los valores del campo o campos contenidos en el término GROUP BY.

Ejemplo:

```
SELECT "Nombre", SUM("Entrada"-"Salida") AS "Saldo"  
FROM "Tabla1"  
GROUP BY "Nombre";
```

Se suman los registros que contienen el mismo nombre. El resultado de la consulta, proporciona la suma de Entrada - Salida para cada persona. Este campo se llamará *Saldo*.

Cada fila del resultado de la consulta contiene el valor del campo *Nombre* y el saldo calculado para ese valor específico.



Consejo

Cuando los campos se procesan usando una función particular (por ejemplo, COUNT, SUM ...), todos los campos que no se procesan con una función pero que se deben mostrar se agrupan usando GROUP BY.

[HAVING Expresión SQL]

La fórmula HAVING se parece mucho a la fórmula WHERE. La diferencia es que la fórmula WHERE se aplica a los valores de los campos seleccionados en la consulta, mientras que la fórmula HAVING se aplica a los valores calculados seleccionados. Específicamente, la fórmula WHERE no puede usar una función agregada como parte de una condición de búsqueda; la fórmula HAVING en cambio sí puede.

La fórmula HAVING tiene dos propósitos, como se muestra en los ejemplos que siguen. En el primer ejemplo, la condición de búsqueda requiere que el tiempo de ejecución mínimo sea inferior a 40 minutos. En el segundo ejemplo, la condición de búsqueda requiere que el saldo de un individuo sea positivo.

Los resultados de la consulta para el primer ejemplo enumeran los nombres de las personas cuyo tiempo de ejecución ha sido inferior a 40 minutos al menos una vez y el tiempo de ejecución mínimo. Las personas que han tenido todos los tiempos de ejecución mayores a 40 minutos no figuran en la lista.

Los resultados de la consulta para el segundo ejemplo enumeran los nombres de las personas que tienen un saldo mayor de entrada que de salida, los nombres cuyo saldo sea 0 o el de entrada inferior al de salida no se mostrarán.

Ejemplo 1:

```
SELECT "Nombre", "Runtime"  
FROM "Tabla1"  
GROUP BY "Nombre", "Runtime"  
HAVING MIN("Runtime") < '00:40:00';
```

Ejemplo 2:

```
SELECT "Nombre", SUM("Entrada"-"Salida") AS "Saldo"  
FROM "Tabla1"  
GROUP BY "Nombre"  
HAVING SUM("Entrada"-"Salida") > 0;
```

[Expresión SQL]

Las expresiones SQL se combinan de acuerdo con el siguiente esquema:

```
[NOT] condición [{ OR | AND } condición]
```

Ejemplo:

```
SELECT *  
FROM "Nombre_Tabla"  
WHERE NOT "Return_date" IS NULL AND "ReaderID" = 2;
```

Los registros leídos de la tabla son aquellos que contienen un "Return_date" y el ReaderID es 2. En la práctica, esto significa que todos los artículos prestados a un lector específico y devueltos se pueden utilizar. Las condiciones están vinculadas con AND. La expresión NOT se refiere solo a la primera condición.

```
SELECT *
```

```
FROM "Nombre_Tabla"  
WHERE NOT ("Return_date" IS NULL AND "ReaderID" = 2);
```

Los paréntesis alrededor de la condición, con NOT fuera de ellos, muestran solo aquellos registros que no cumplen por completo la condición entre paréntesis. Esto cubriría todos los registros, excepto los del *ReaderID* número 2, que aún no hayan sido devueltos.

[Expresión SQL]: condiciones

```
{ valor [| valor]
```

Un valor puede ser un valor sencillo o varios valores unidos por dos líneas verticales ||. Naturalmente, esto también se aplica al contenido del campo.

```
SELECT "Apellido" || ', ' || "Nombre" AS "Nombre Completo"  
FROM "Tabla_nombres"
```

El contenido de los campos *Apellido* y *Nombre* se muestran juntos en un campo llamado *Nombre Completo*. Además se inserta una coma y un espacio entre *Apellido* y *Nombre*.

```
| valor { = | < | <= | > | >= | <> | != } valor
```

Estos signos corresponden a los operadores matemáticos conocidos:

```
{ Igual que | Menor que | Menor o igual que | Mayor que | Mayor o igual que | Distinto de | No igual a}
```

```
| valor IS [NOT] NULL
```

El campo correspondiente no tiene contenido, porque no se ha escrito nada en él. Esto no se puede determinar de manera inequívoca en la interfaz, ya que un campo de texto visualmente vacío no significa que el campo esté completamente sin contenido. De manera predeterminada en Base los campos vacíos están configurados como NULL.

```
| EXISTS(Resultado_Consulta)
```

Ejemplo:

```
SELECT "Nombre"  
FROM "Tabla1"  
WHERE EXISTS  
  (SELECT "Nombre"  
   FROM "Tabla2"  
   WHERE "Tabla2"."Nombre" = "Tabla1"."Nombre")
```

Se muestran los nombres de la *Tabla1* que coinciden con los nombres de la *Tabla2*.

```
| valor BETWEEN valor AND valor
```

BETWEEN valor1 AND valor2 devuelve todos los valores desde el valor1 hasta el valor2 (incluido). Si los valores son letras, se utiliza un orden alfabético en el que las letras minúsculas tienen el mismo valor que las mayúsculas correspondientes.

```
SELECT "Nombre"  
FROM "Tabla_nombres"  
WHERE "Nombre" BETWEEN 'A' AND 'E';
```

Esta consulta devuelve todos los nombres que comienzan con A, B, C o D (incluyendo también los que tengan las letras minúsculas). Como E se establece como el límite superior, los nombres que comienzan con E no se incluyen. La letra E aparece justo antes de los nombres que comienzan con E y solo si el nombre es exactamente «E» se incluye.

```
| valor [NOT] IN ( {valor [, ...] | Resultado_Consulta } )
```

Esto requiere una lista de valores o una consulta. La condición se cumple si el valor se incluye en la lista de valores o en el resultado de la consulta.

```
| valor [NOT] LIKE valor [ESCAPE] valor }
```

El operador LIKE es uno que se necesita en muchas funciones de búsqueda simples. El valor se ingresa utilizando el siguiente patrón:

'%' representa cualquier número de caracteres (incluido 0),

'_' reemplaza exactamente un carácter.

Para buscar un carácter especial como '%' o '_', los caracteres deben seguir inmediatamente a otro carácter definido como ESCAPE.

```
SELECT "Nombre"  
FROM "Tabla_nombres"  
WHERE "Nombre" LIKE '\_%' ESCAPE '\'
```

Esta consulta muestra todos los nombres que comienzan con un guión bajo. '\' se utiliza como el carácter ESCAPE. (para indicar que este carácter es parte de la consulta y no un carácter específico de sustitución).

[Expresión SQL]: valores

```
[+ | -] { Expresión [{ + | - | * | / | || } Expresión]
```

Los valores pueden tener uno de los signos anteriores. Se permiten la suma, resta, multiplicación, división y concatenación de expresiones. Un ejemplo de concatenación:

```
SELECT "Apellido"||', '||"Nombre"  
FROM "Tabla_nombres"
```

De esta forma, la consulta muestra los registros como un campo que contiene el campo "Apellido" seguido de coma y el campo "Nombre". El operador de concatenación puede calificarse mediante las siguientes expresiones.

```
| ( Condición )
```

Se aplica igual que en la sección anterior.

```
| Función ( [Parámetro] [...] )
```

Vea la sección sobre funciones en el apéndice.

Las siguientes consultas también se denominan sub-consultas (sub-selecciones).

```
| Resultado de Consulta que devuelve exactamente una respuesta
```

Como un registro solo puede tener un valor en cada campo, solo una consulta que produce exactamente un valor puede mostrarse en su totalidad.

```
| {ANY|ALL} (Consulta que devuelve exactamente una respuesta de una columna completa)
```

A menudo hay una condición que compara una expresión con un grupo completo de valores. Combinado con ANY, esto significa que la expresión debe aparecer al menos una vez en el grupo. Esto también se puede especificar utilizando la condición IN. = ANY produce el mismo resultado que IN.

Combinado con ALL significa que todos los valores del grupo deben corresponder a una expresión única.

[Expresión SQL]: Expresión

```
{ 'Texto' | Entero | Numero de coma flotante | ["Tabla"."]Campo" | TRUE | FALSE | NULL }
```

Básicamente, los valores sirven como argumentos para diversas expresiones, dependiendo del formato de origen. Para buscar el contenido de los campos de texto, coloque el contenido entre comillas. Los enteros (Integer) se escriben sin comillas, al igual que los números de coma flotante.

Los campos representan los valores que hay en esos campos en la tabla. Por lo general, los campos se comparan entre sí o con valores específicos. En SQL, los nombres de campo deben colocarse entre comillas dobles, ya que de lo contrario no se reconocerán correctamente. Por lo general, SQL supone que el texto sin comillas dobles no tiene caracteres especiales, es decir, una sola palabra sin espacios y en mayúsculas.

Si hay varias tablas en la consulta, un campo se debe identificar con el nombre de la tabla seguido de un punto y el nombre del campo.

TRUE y FALSE generalmente derivan de los campos Sí/No.

NULL significa que no hay contenido. No es lo mismo que 0, sino que está «vacío».

UNION [ALL | DISTINCT] Resultado_Consulta

Esto vincula las consultas para que el contenido de la segunda consulta se escriba debajo de la primera. Para que esto funcione, todos los campos en ambas consultas deben coincidir en tipo. Este enlace de varias consultas funciona solo en el modo de órdenes SQL directas.

```
SELECT "Nombre"  
FROM "Tabla1"  
UNION DISTINCT  
    SELECT "Nombre"  
    FROM "Tabla2";
```

Esta consulta devuelve todos los nombres de *Tabla1* y *Tabla2*; el término adicional **DISTINCT** significa que no se mostrarán nombres duplicados. **DISTINCT** es el valor predeterminado en este contexto. Por defecto, los nombres se ordenan alfabéticamente en orden ascendente. **ALL** hace que se muestren todos los campos *Nombre* de *Tabla1*, seguidos de los campos *Nombre* de *Tabla2*. En este caso, se ordena por la clave primaria de manera predeterminada.

El uso de esta técnica de consulta permite enumerar consecutivamente valores de un registro tras otro en una columna. Supongamos que tiene una tabla llamada *Stock* en la que hay campos para *Precio_venta*, *Precio_Rebaja_1* y *Precio_Rebaja_2*. A partir de esto, se desea calcular un campo de combinación que enumerará estos precios directamente uno debajo del otro:

```
SELECT  
    "Precio_venta"  
FROM "Stock" WHERE "Stock_ID" = 1  
UNION  
SELECT  
    "Precio_Rebaja_1"  
FROM "Stock" WHERE "Stock_ID" = 1  
UNION  
SELECT  
    "Precio_Rebaja_2"  
FROM "Stock" WHERE "Stock_ID" = 1;
```

La clave principal para la tabla *Stock* naturalmente tiene que establecerse mediante una macro, ya que el campo de combinación tendrá una entrada coincidente.

MINUS [DISTINCT] | EXCEPT [DISTINCT] Resultado_consulta

```
SELECT "Nombre"
```

```
FROM "Tabla1"
EXCEPT
  SELECT "Nombre"
  FROM "Tabla2";
```

Muestra todos los nombres de *Tabla1*, excepto los nombres que figuran en *Tabla2*. **MINUS** y **EXCEPT** conducen al mismo resultado. La clasificación es alfabética.

INTERSECT [DISTINCT] Resultado_consulta

```
SELECT "Nombre"
FROM "Tabla1"
INTERSECT
  SELECT "Nombre"
  FROM "Tabla2";
```

Esto mostrará los primeros nombres que aparecen en ambas tablas. La ordenación es nuevamente alfabética. Actualmente esto solo funciona en modo de orden SQL directa.

[ORDER BY Criterio_de_Ordenación [, ...]]

Expresión puede ser un nombre de campo, un número de columna (empezando a contar por 1 desde la izquierda), un alias (formulado con **AS**, por ejemplo) o una expresión de valor compuesto (consulte [*Expresión SQL*]: valores). El orden de clasificación es ascendente (**ASC**). Si desea una ordenación descendente, debe especificar **DESC**.

```
SELECT "Nombre", "Apellido" AS "Nombre Completo"
FROM "Tabla1"
ORDER BY "Apellido";
```

devuelve el mismo resultado que:

```
SELECT "Nombre", "Apellido" AS "Nombre Completo"
FROM "Tabla1"
ORDER BY 2;
```

al igual que:

```
SELECT "Nombre", "Apellido" AS "Nombre Completo"
FROM "Tabla1"
ORDER BY "Nombre Completo";
```

Usar un alias en una consulta

Las consultas pueden devolver un resultado con nombres de campos cambiados.

```
SELECT "First_name", "Surname" AS "Name"
FROM "Table1"
```

El campo *Surname* pasa a llamarse *Name* en el resultado.

Si una consulta involucra dos tablas, cada nombre de campo debe estar precedido por el nombre de la tabla:

```
SELECT "Table1"."First_name", "Table1"."Surname" AS "Name", "Table2"."Class"
FROM "Table1", "Table2"
WHERE "Table1"."Class_ID" = "Table2"."ID"
```

El nombre de la tabla también puede tener un alias, pero no se reproducirá en la vista de tabla. Si se establece dicho alias, todos los nombres de tabla en la consulta deben modificarse en consecuencia:

```
SELECT "a"."Nombre", "a"."Apellido" AS "Nombre Completo", "b"."Clase"
```

```
FROM "Tabla1" AS "a", "Tabla2" AS "b"  
WHERE "a"."Class_ID" = "b"."ID"
```

La asignación de un alias para una tabla se puede llevar a cabo más brevemente sin usar el término AS:

```
SELECT "a"."Nombre", "a"."Apellido" "Nombre Completo", "b"."Clase"  
FROM "Tabla1" "a", "Tabla2" "b"  
WHERE "a"."Class_ID" = "b"."ID"
```

Sin embargo, esto hace que el código sea menos legible. Por lo que se recomienda usar la forma abreviada solo en circunstancias excepcionales.



Nota

Desafortunadamente, el código para el editor de consultas en la interfaz gráfica de usuario se ha cambiado recientemente para que las designaciones de alias se creen sin usar el prefijo AS. Este cambio se debe a que al utilizar bases de datos externas, cuyo método de especificar alias no es predecible, la inclusión de AS ha dado lugar a mensajes de error.

Si se edita el código de la consulta utilizando la interfaz (no directamente en SQL) los alias existentes pierden su prefijo AS. Si se quiere evitar esto, las consultas siempre se tienen que editar en la *vista SQL*.

Un nombre de alias también permite usar una tabla con el filtrado correspondiente más de una vez dentro de una consulta:

```
SELECT "CuentasCaja"."Saldo", "Cuenta"."Fecha",  
       "a"."Saldo" AS "Actual",  
       "b"."Saldo" AS "Deseado"  
FROM "Cuenta"  
     LEFT JOIN "Cuenta" AS "a"  
     ON "Cuenta"."ID" = "a"."ID" AND "a"."Saldo" >= 0  
     LEFT JOIN "Cuenta" AS "b"  
     ON "Cuenta"."ID" = "b"."ID" AND "b"."Saldo" < 0
```

Consultas para la creación de campos en Listado

Los controles de *Listado*, generalmente, muestran un valor que no se corresponde con el contenido de la tabla subyacente. Se utilizan para mostrar el valor asignado a una clave externa en lugar de la clave en sí. El valor que finalmente se guarda en el formulario no debe aparecer en la primera posición de los campos del *Listado*.

```
SELECT "Nombre", "ID"  
FROM "Tabla1";
```

Esta consulta mostrará todos los valores de *Nombre* y los valores de *ID* de la clave principal que proporciona la tabla subyacente del formulario. Por supuesto, aún no es óptimo. Los nombres aparecen sin clasificar y, en el caso de nombres idénticos, es imposible determinar a qué persona hace referencia.

```
SELECT "Nombre"||' '||"Apellido", "ID"  
FROM "Tabla1"  
ORDER BY "Nombre"||' '||"Apellido";
```

Ahora aparecen *Nombre* y *Apellido*, separados por un espacio. Los nombres se vuelven distinguibles y también se ordenan. Pero el orden sigue la lógica habitual de comenzar con la

primera letra de la cadena, por lo que se ordena por *Nombre*, y después por *Apellido*. Un orden diferente al de los campos mostrados sería confuso.

```
SELECT "Apellido"||', '||"Nombre", "ID"
FROM "Tabla1"
ORDER BY "Apellido"||', '||"Nombre";
```

Esto, ahora, muestra a una ordenación que se corresponde mejor con la manera habitual de ordenar. Los miembros de una familia aparecen juntos, uno debajo del otro; sin embargo, estarían intercalados miembros de otras familias con el mismo apellido. Para distinguirlos, necesitaríamos agruparlos de manera diferente dentro de la consulta.

Hay un problema final: si dos personas tienen el mismo apellido y nombre, no podrán distinguirse. Una solución podría ser usar un sufijo en el nombre. ¡Pero, imagine cómo se vería un saludo dirigido al Sr. «Müller II»! Una opción más acertada sería usar su *ID*.

```
SELECT "Apellido"||', '||"Nombre"||' - Id:'||"ID", "ID"
FROM "Tabla1"
ORDER BY "Apellido"||', '||"Nombre"||"ID";
```

Aquí todos los registros son distinguibles. Lo que realmente se muestra es: «Muller, Henry - Id: 2» (poniendo como ejemplo al Sr. Henry Muller con un ID de valor 2).

En el formulario *Loan* de la base de ejemplo *Media*, hay un control de listado que debe mostrar solo los artículos que aún no se han prestado. Se crea usando la siguiente fórmula SQL:

```
SELECT "Title" || ' - Nr. ' || "ID", "ID"
FROM "Media"
WHERE "ID" NOT IN
  (SELECT "Media_ID"
   FROM "Loan"
   WHERE "Return_Date" IS NULL)
ORDER BY "Title" || ' - Nr. ' || "ID" ASC
```

Es importante que este control de listado se actualice siempre que un artículo dentro de él sale en préstamo.

En el siguiente ejemplo el control de tabla utiliza controles de listado que muestran el contenido de varios campos para una selección de artículos y su precio.

	Quantity	Goods		
	2	Schnellhefter, Pappe	-	0,46 €
	5	Papier, 500 Blatt	-	5,65 €
	10	Bleistift HB	-	0,25 €
	1	Hefter, Tischgerät	-	11,25 €
	1	Locher, Registratur	-	15,48 €
		Bleistift HB	-	0,25 €
		Desktop-PC Prozessor i7 A	-	599,00 €
		Hefter, Tischgerät	-	11,25 €
		Locher, Registratur	-	15,48 €
		MiniPC Nano Prozessor: i5	-	398,00 €
		Papier, 500 Blatt	-	5,65 €
		Schnellhefter, Pappe	-	0,46 €
Record	5	Ultrabook Bildschirm: 15"	-	889,00 €

Para hacer que su representación funcione adecuadamente y estén alineados los campos que aparecen en el listado primero se debe elegir una fuente de ancho fijo adecuada. Se puede utilizar «Courier» o cualquier fuente «monotype» como «Liberation Mono».

Se crea el control de tabla y en sus datos se usa el código SQL:

```
SELECT LEFT("Stock"||SPACE(25),25) || ' - ' || RIGHT(SPAC(8)||"Price",8) || ' €',"ID"
FROM "Stock"
```

```
ORDER BY ("Stock" || ' - ' || "Price" || ' €') ASC
```

El contenido del campo Stock se ha rellenado con espacios para que toda la cadena tenga una longitud mínima de 25 caracteres para asegurar que no se trunque una cantidad alta de stock. Luego se colocan las primeras 25 letras de la descripción del artículo y se corta el excedente.

Cuando el contenido del campo de lista contiene caracteres que no se imprimen (como líneas nuevas) se vuelve más complejo. Se debe adaptar el código para que el texto se ajuste al control.

```
SELECT LEFT(REPLACE("Stock",CHAR(10),' ')||SPACE(25), 25) || ' - ' || ...
```

Esto reemplaza una línea (CHAR10 en Linux) con un espacio. Para Windows, se debe usar CHAR (13) para eliminar el retorno de carro.

La cantidad de espacios necesarios también se determina en función de la consulta.

```
SELECT LEFT("Stock"||SPACE((SELECT MAX(LENGTH("Stock"))
FROM "Stock")), (SELECT MAX(LENGTH("Stock")) FROM "Stock")) || ' - ' || RIGHT('
' || "Price", 8) || ' €', "ID"
FROM "Stock"
ORDER BY ("Stock" || ' - ' || "Price" || ' €') ASC
```

Como el precio se debe mostrar justificado a la derecha, para separarlo del artículo se rellena con espacios a la izquierda del campo y además se coloca un máximo de ocho caracteres desde la derecha. La representación seleccionada funcionará para cualquier precio hasta 99999,99€.

Si desea reemplazar el punto decimal con una coma en SQL, necesitará un código adicional:

```
REPLACE(RIGHT(' ' || "Price", 8), '.', ',')
```

Consultas como base para información adicional en formularios

Si desea que un formulario muestre una información adicional que de otro modo no sería visible, existen varias posibilidades de consulta. Lo más sencillo es recuperar esta información con consultas independientes e insertar los resultados en el formulario.

La desventaja de este método es que los cambios en los registros pueden afectar al resultado de la consulta, y desafortunadamente estos cambios no se muestran de manera automática.

Aquí hay un ejemplo del control de existencias para un pago.

La tabla de la factura contiene las claves foráneas del total de artículos y un número de identificación de la factura. El comprador tiene muy poca información si no se imprime la factura una información más detallada (resultado de una consulta).

Los artículos se identifican solo mediante la lectura de un código de barras. Sin una consulta aplicada, el formulario solo muestra la información de la siguiente tabla:

Total	C_Barras
3	17
2	24

Lo que está oculto detrás de estos números no se puede hacer visible usando un *Listado*, ya que la clave externa se ingresa directamente usando el código de barras. Tampoco es posible usar un *Listado* junto al artículo para mostrar al menos el precio unitario.

Una consulta puede ayudar:

```
SELECT "Checkout"."Receipt_ID", "Checkout"."Total", "Stock"."Item", "Stock"."Unit_Price",
"Checkout"."Total"*"Stock"."Unit_price" AS "Total_Price"
FROM "Checkout", "Item"
```

```
WHERE "Stock"."ID" = "Checkout"."Item_ID";
```

Ahora, al menos después de haber ingresado la información, sabemos cuánto se debe pagar por los 3 artículos de la referencia 17. Solo la información relevante para el número de factura (*Receipt_ID*) debe filtrarse a través del formulario. Lo que ahora falta es el total del precio de la factura.

```
SELECT "Checkout"."Receipt_ID", SUM("Checkout"."Total"*"Stock"."Unit_price") AS "Sum"  
FROM "Checkout", "Stock"  
WHERE "Stock"."ID" = "Checkout"."Item_ID"  
GROUP BY "Checkout"."Receipt_ID";
```

Diseñe el formulario para mostrar un registro de la consulta cada vez. Dado que la consulta se agrupa por *Receipt_ID*, el formulario muestra información sobre un cliente cada vez.



Consejo

Si un formulario debe mostrar valores de fecha que dependan de otra fecha (por ejemplo, en una biblioteca, un período de préstamo podría ser de 21 días, ¿cuál es la fecha de devolución?), No puede utilizar las funciones integradas de HSQLDB porque no hay una función «DATEADD».

La consulta:

```
SELECT "Date", DATEDIFF('dd','1899-12-30',"Date")+21 AS  
"ReturnDate" FROM "Table"
```

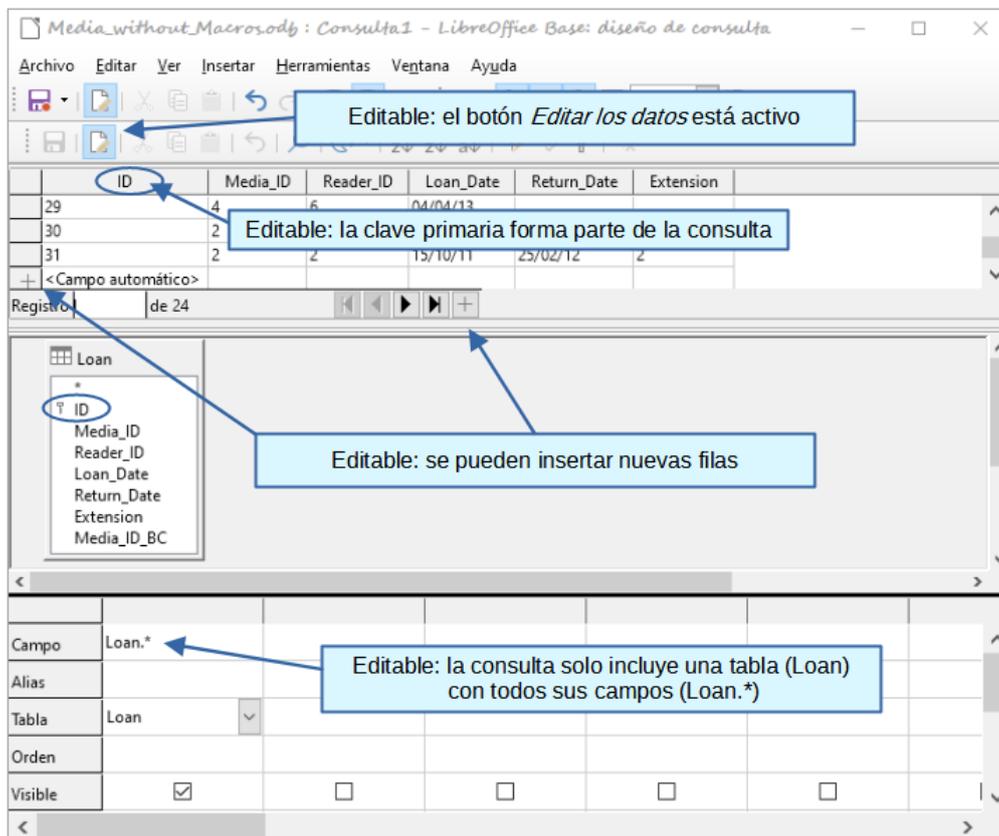
Devolverá la fecha correcta para la devolución en un formulario. Esta consulta cuenta los días del 30.12.1899. (Fecha predeterminada, que Calc también usa como valor 0)

El valor devuelto solo es un número y no una fecha que se pueda utilizar en otra consulta.

El número devuelto no es adecuado para su uso en consultas porque el formato de las consultas no se guarda. Necesita crear una vista en su lugar.

Posibilidades de introducción de datos en consultas

Para introducir datos en una consulta, la clave principal de la tabla utilizada en la consulta debe estar presente. Esto también se aplica a una consulta que relaciona varias tablas: todas las claves primarias de las tablas implicadas deben estar presentes.



En el préstamo de artículos, no tiene sentido mostrar a un lector elementos que ya han sido devueltos hace algún tiempo.

```
SELECT "ID", "Reader_ID", "Media_ID", "Loan_Date"
FROM "Loan"
WHERE "Return_Date" IS NULL;
```

De esta manera, un formulario puede mostrar dentro de un campo de control de tabla todos los artículos que un lector en particular ha tomado prestados durante su pertenencia a la biblioteca. La consulta también debe filtrarse utilizando una estructura de formulario adecuada (lector en el formulario principal, consulta en el subformulario), de modo que solo se muestren los artículos que realmente tiene prestados.

La consulta es adecuada para introducir datos, ya que la clave principal se incluye en la consulta.

Una consulta no se puede editar si consta de más de una tabla y se accede a las tablas mediante un alias. En ese caso, no influye el que la clave principal esté o no presente en la consulta.

ID	Title	Category_ID	KatID	Category	
0	Der kleine Hobbit	0	0	Fantasy	
1	Das sogenannte Böse	2	2	Liedermacher	
5	I hear you knocking	1	1	Rock	
8	Im Augenblick	2	2	Liedermacher	
+ <Campo automático>		<Campo automático>			

Campo	ID	Title	Category_ID	ID	Category
Alias				KatID	
Tabla	Media	Media	Media	Category	Category
Orden					
Visible	<input checked="" type="checkbox"/>				

```
SELECT "Media"."ID", "Media"."Title", "Media"."Category_ID", "Category"."ID" AS "katID",
"Category"."Category"
FROM "Media", "Category"
WHERE "Media"."Category_ID" = "Category"."ID";
```

Esta consulta se puede editar, ya que ambas claves principales están incluidas y se puede acceder a ellas en las tablas (que no tienen un alias de tabla). Una consulta que vincula varias tablas juntas también requiere que todas las claves primarias estén presentes.

En una consulta que involucra varias tablas, no es posible alterar un campo de clave externa de una tabla que se refiere a un registro de otra tabla.

En el registro que se muestra a continuación, se intentó cambiar la categoría del título «Der kleine Hobbit». El campo *Category_ID* se cambió de 0 a 2. El cambio pareció realizarse y la nueva categoría apareció en el registro. Sin embargo, resultó imposible guardarlo.

ID	Title	Category_ID	KatID	Category	
0	Der kleine Hobbit	2	2	Liedermacher	
1	Das sogenannte Böse	2	2	Liedermacher	
5	I hear you knocking	1	1	Rock	
8	Im Augenblick	2	2	Liedermacher	
+ <Campo automático>		<Campo automático>			

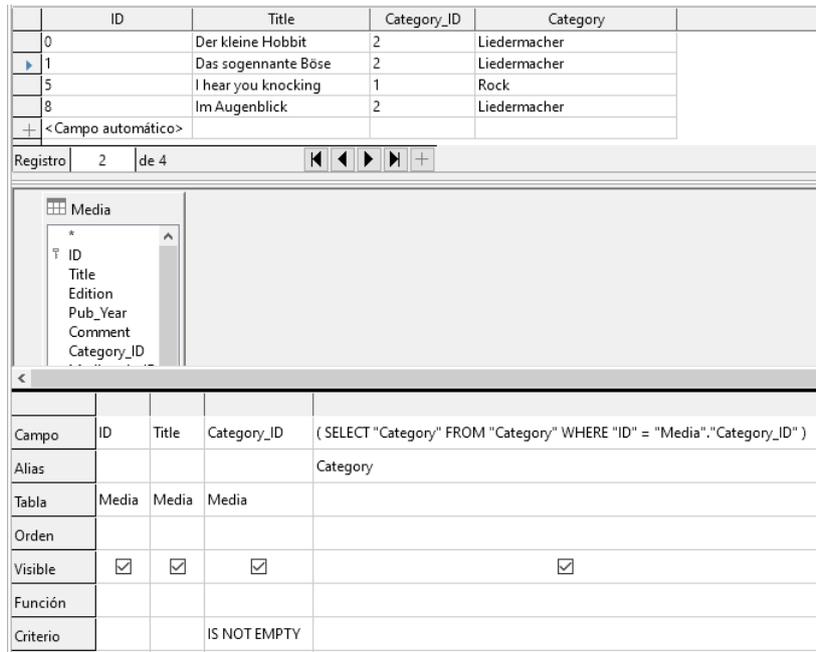
Campo	ID	Title	Category_ID	ID	Category
Alias				KatID	
Tabla	Media	Media	Media	Category	Category
Orden					
Visible	<input checked="" type="checkbox"/>				

En cambio, sí que es posible editar el contenido de un registro perteneciente a *Category*. Por ejemplo, para reemplazar «Fantasy» por «Liedermacher». Se debe tomar en cuenta que el nombre de la categoría se modificará para todos los registros vinculados con esta categoría.

La siguiente consulta accede a la tabla de artículos usando un alias. La consulta no se puede editar.

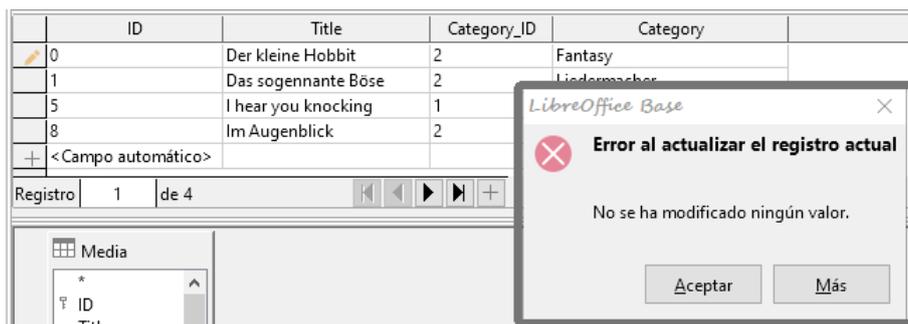
```
SELECT "m"."ID", "m"."Title", "Category"."Category", "Category"."ID" AS "katID"
FROM "Media" AS "m", "Category"
WHERE "m"."Category_ID" = "Category"."ID";
```

En el ejemplo anterior, este problema se puede evitar, si realmente necesita usar un alias de tabla, utilizando una subconsulta relacionada (consulte la página 39). En ese caso, una consulta solo se puede editar si únicamente contiene una tabla en la consulta principal.



En la vista de diseño solo aparece una tabla. La tabla *Media*. En la consulta se ha empleado un alias «m» para la tabla y acceder al contenido del campo *Category_ID* utilizando una subconsulta relacionada.

En una consulta como esta, ahora es posible cambiar el campo de un registro (de clave externa *Category_ID*) a otra categoría. En el ejemplo anterior, el campo *Category_ID* del artículo «Der Kleine Hobbit» se cambia de 0 a 2. El artículo pertenecerá ahora a la categoría «Liedermacher».



Sin embargo, ahora ya no es posible cambiar un valor en el campo que ha recibido su contenido a través de una subconsulta relacionada. Se muestra un intento de volver el registro con título «Der Kleine Hobbit» a su categoría anterior, este cambio no se registra y tampoco se puede guardar. En la tabla que muestra la vista de diseño, el campo *Category* no está presente porque pertenece a la subconsulta.

Uso de parámetros en consultas

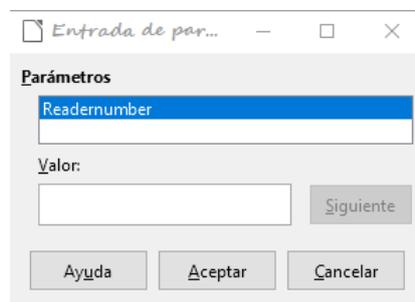
Si a menudo se usa la misma consulta básica pero en determinados momentos se utilizan valores diferentes, se pueden crear consultas con parámetros. En principio, las consultas con parámetros funcionan igual que las consultas para un subformulario:

```
SELECT "ID", "Reader_ID", "Media_ID", "Loan_Date"
FROM "Loan"
WHERE "Return_Date" IS NULL AND "Reader_ID"=2;
```

Esta consulta muestra solo los artículos prestados al lector con ID 2.

```
SELECT "ID", "Reader_ID", "Media_ID", "Loan_Date"
FROM "Loan"
WHERE "Return_Date" IS NULL AND "Reader_ID" = :Readernumber;
```

Con esta modificación, cuando ejecute la consulta, aparecerá un diálogo que le pedirá un valor (número de lector). Cuando se introduzca un valor, se mostrarán los artículos actualmente prestados a ese lector.



```
SELECT
  "Loan"."ID",
  "Reader"."LastName"||', '||"Reader"."FirstName",
  "Loan"."Media_ID",
  "Loan"."Loan_date"
FROM "Loan", "Reader"
WHERE "Loan"."Return_Date" IS NULL
  AND "Reader"."ID" = "Loan"."Reader_ID"
  AND "Reader"."LastName" LIKE '% ' || :Readername || '%'
ORDER BY "Reader"."LastName"||', '||"Reader"."FirstName" ASC;
```

Esta otra opción es claramente más fácil de usar que la anterior. Ya no es necesario saber el número del lector. Solo necesita ingresar parte del apellido en el control de listado y se muestran todos los artículos prestados a los lectores coincidentes.

Con otra pequeña modificación, si se reemplaza

```
"Reader"."LastName" LIKE '% ' || :Readername || '%'
```

por

```
LOWER("Reader"."LastName") LIKE '% ' || LOWER( :Readername ) || '%'
```

Ya no importa si el nombre se ingresa en mayúscula o minúscula.

Si el parámetro en la consulta anterior se deja vacío, todas las versiones de LibreOffice hasta 4.4 mostrarán a todos los lectores, ya que a partir de la Versión 4.4 un campo de parámetro vacío se lee como NULL en lugar de una cadena vacía. Si no desea este comportamiento, debe usar un truco para evitarlo:

```
LOWER ("Reader"."LastName")
```

```
LIKE '%' || IFNULL( NULLIF ( LOWER (:Readername), '' ), '$$' ) || '%'
```

El campo de parámetro vacío devuelve una cadena vacía y no un NULL a la consulta. Por lo tanto, al campo de parámetro vacío se le debe asignar la propiedad NULL usando NULLIF. Luego, dado que la entrada del parámetro ahora produce NULL, se puede restablecer a un valor que normalmente no aparece en ningún registro. En el ejemplo anterior, esto es '\$\$'. Este valor, por supuesto, no se encontrará en la búsqueda.

A partir de la versión 4.4, son necesarias adaptaciones a esta técnica de consulta:

```
LOWER ("Reader"."LastName") LIKE '%' || LOWER (:Readername) || '%'
```

al no introducir ningún valor, la combinación:

```
'%' || LOWER (:Readername) || '%'
```

devuelve inevitablemente el valor NULL.

Para evitar esto, agregue una condición adicional, que para un campo vacío, todos los valores se muestren realmente:

```
(LOWER ("Reader"."LastName") LIKE '%' || LOWER (:Readername) || '%') OR :Readername IS NULL)
```

Todo esto debe ponerse entre paréntesis. Primero se busca un nombre, si el campo está vacío (NULL de LibreOffice 4.4), se aplicará la segunda condición.

Cuando se usan formularios, el parámetro se puede pasar del formulario principal a un subformulario. Sin embargo, a veces puede que no se actualicen las consultas que usan parámetros en subformularios si los datos se cambian o se ingresan nuevamente.

Sería aconsejable alterar el contenido de los controles de listado utilizando cierta configuración del formulario principal. Por ejemplo, podríamos evitar que los artículos de la biblioteca se presten a determinados lectores que están bloqueados. Lamentablemente, no es posible controlar esta configuración del control de listado mediante el uso de parámetros.

Subconsultas

Las subconsultas integradas en campos solo pueden devolver un único registro. El campo también devuelve solo un valor.

```
SELECT "ID", "Income", "Expenditure",  
  ( SELECT SUM( "Income" ) - SUM( "Expenditure" )  
    FROM "Checkout" ) AS "Balance"  
FROM "Checkout";
```

Esta consulta permite la entrada de datos (clave principal incluida). La subconsulta arroja precisamente un único valor: el saldo total. Esto permite leer el saldo en la caja registradora después de cada entrada.

Esto todavía no es comparable con el formulario de pago del supermercado descrito en "Consultas como base para información adicional en formularios" en la página 32. Naturalmente, carece de los cálculos individuales de Total*Unit_price, pero también de la presencia del número de factura (Receipt_ID) Solo se da la suma total. Se puede incluir al menos el número de factura mediante el uso de un parámetro de consulta:

```
SELECT "ID", "Income", "Expenditure",  
  ( SELECT SUM( "Income" ) - SUM( "Expenditure" )  
    FROM "Checkout"  
    WHERE "Receipt_ID" = :Receipt_Number ) AS "Balance"  
FROM "Checkout" WHERE "Receipt_ID" = :Receipt_Number;
```

En una consulta con parámetros, el parámetro debe ser el mismo en ambas declaraciones de consulta si se va a reconocer como un parámetro.

Se pueden incluir dichos parámetros en subformulario. El subformulario recibe, el nombre del parámetro correspondiente en lugar de un nombre de campo. Este enlace solo se puede especificar en las propiedades del subformulario, no se puede especificar cuando se usa el asistente.



Nota

Los subformularios basados en consultas no se actualizan automáticamente en función de sus parámetros. Es más apropiado pasar el parámetro directamente desde el formulario principal.

Subconsultas relacionadas

Usando una consulta aún más refinada, una consulta editable le permite incluso llevar el saldo corriente para la caja registradora:

```
SELECT "ID", "Income", "Expenditure",  
( SELECT SUM( "Income" ) - SUM( "Expenditure" )  
  FROM "Checkout"  
  WHERE "ID" <= "a"."ID" ) AS "Balance"  
FROM "Checkout" AS "a"  
ORDER BY "ID" ASC
```

La tabla de la factura *Checkout* tiene un alias (AS "a"). "a" solo devuelve la relación con los valores actuales en este registro. De esta manera, el valor actual de *ID* de la consulta externa se puede evaluar dentro de la subconsulta. Por lo tanto, dependiendo de la *ID*, se determina el saldo anterior en el momento correspondiente, si se cuenta con el hecho de que la *ID*, que es un valor automático, se incrementa por sí misma.



Nota

Si la subconsulta se va a filtrar por contenido utilizando la función de filtro del editor de consultas, solo funciona si usan paréntesis dobles en lugar de sencillos al principio y al final de la subconsulta: ((SELECT)) AS "Balance"

Consultas como tablas fuente para consultas

Se necesita una consulta para establecer un bloqueo a los lectores que hayan recibido un tercer aviso de retraso para un artículo.

```
SELECT "Loan"."Reader_ID", '3rd Overdue-the reader is blacklisted' AS "Lock" FROM (SELECT  
COUNT("Date") AS "Total_Count", "Loan_ID"  
  FROM "Recall" GROUP BY "Loan_ID") AS "a", "Loan"  
WHERE "a"."Loan_ID" = "Loan"."ID" AND "a"."Total_Count" > 2
```

Primero examinemos la consulta interna, con la que se relaciona la consulta externa:

```
(SELECT COUNT("Date") AS "Total_Count", "Loan_ID" FROM "Recall" GROUP BY "Loan_ID") AS  
"a"
```

En esta consulta, se determina el número de entradas de *Date* agrupadas por la clave externa *Loan_ID*. Esto no debe hacerse dependiente de *Reader_ID*, ya que a la hora del recuento devolvería no solo tres avisos de vencimiento para un solo artículo, sino también tres artículos con

un aviso de vencimiento cada uno. La consulta interna recibe un alias (*a*) para que se pueda vincular con el *Reader_ID* desde la consulta externa.

La consulta externa se relaciona en este caso solo con la fórmula condicional de la consulta interna. Muestra solo un *Reader_ID* y el texto para el campo *Lock* cuando "*Loan*".*ID*" y "*a*".*Loan_ID*" son iguales y "*a*" es mayor que 2 ("*Total_Count*" > 2).

En principio, todos los campos de la consulta interna están disponibles para la externa. Entonces, la suma "*a*".*Total_Count*" se puede combinar en la consulta externa para dar el total de multas reales.

Este tipo de construcciones son válidas en *vista SQL* y se deben formular en el editor de consultas SQL, pero es posible que al intentar volver a la *vista Diseño* de consulta no se pueda mostrar, recibirá el mensaje de la figura 7, pero se puede guardar y es totalmente operativo. El modo de vista de diseño no puede encontrar el campo contenido en la consulta interna *Loan_ID*, que gobierna la relación entre las consultas internas y externas.

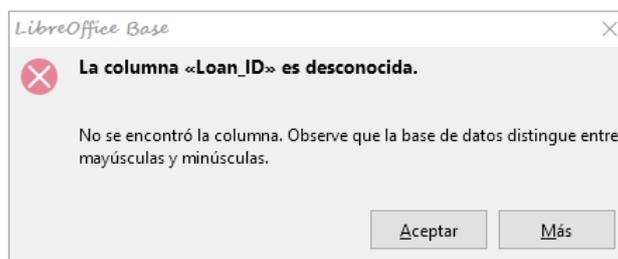
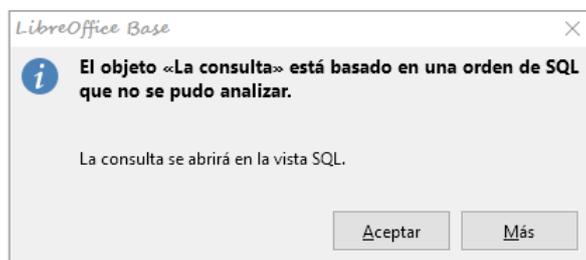


Figura 7: Mensaje de error

Al cerrar la consulta e intentar abrirla posteriormente recibirá el siguiente mensaje:



Si luego abre la consulta para editarla en la *vista SQL* e intenta cambiar de allí a la *Vista de diseño*, recibirá el mismo mensaje de error de la figura 7.

Cuando la consulta se ejecuta en modo SQL, el contenido correspondiente de la subconsulta se reproduce sin error. Por lo tanto, no necesita usar el modo SQL directo en este caso.

La consulta externa utiliza los resultados de la consulta interna para producir los resultados finales. Esta es una lista de los valores de *Loan_ID* que deben ser bloqueados y por qué. Si desea limitar aún más los resultados finales, utilice las funciones de ordenación y filtro de la interfaz gráfica de usuario.

Las siguientes capturas de pantalla muestran otra forma de actuación de una consulta con subconsultas.

En este caso, una consulta a una base de datos de *Stock* está tratando de determinar lo que el cliente debe pagar en la caja registradora. Los precios individuales se multiplican por el número de artículos comprados dando un subtotal. La suma de estos subtotales necesita ser determinada. Todo esto debe poderse editar para que la consulta se pueda utilizar como base de un formulario.

	ID	quantity	articles_ID	articleID	price	subtotal
	44	1	6	6	\$398.00	398
	45	10	2	2	\$0.46	4,6
	46	1	5	5	\$889.00	889
	47	1	6	6	\$398.00	398
	48	1	7	7	\$599.00	599
+	<Campo automé			<Campo automático>		

Registro 1 de 49

```

SELECT
  "sale"."ID",
  "sale"."quantity",
  "sale"."articles_ID",
  "articles"."ID" AS "articleID",
  "articles"."price",
  "quantity" * "price" AS "subtotal"
FROM "sale",
     "articles"
WHERE "sale"."articles_ID" = "articles"."ID"

```

Figura 8: Consulta usando dos tablas. Para hacerla editable, se deben incluir ambas claves principales.



Nota

Debido a un error (61871), Base no actualiza el resultado parcial automáticamente.

	ID	quantity	articles_ID	articleID	price	subtotal
	44	1	6	6	\$398.00	398
	45	10	2	2	\$0.46	4,6
	46	1	5	5	\$889.00	889
	47	1	6	6	\$398.00	398
▶	48	1	7	7	\$599.00	599
+	<Campo automé			<Campo automático>		

Registro 49 de 49

```

SELECT
  "sale"."ID", "sale"."quantity",
  "sale"."articles_ID", "articles"."price",
  "quantity" * "price" AS "subtotal"
FROM "sale",
     ( SELECT
         "ID",
         "price"
       FROM "articles" )
     AS "articles"
WHERE "sale"."articles_ID" = "articles"."ID"

```

Figura 9: La tabla articles se mueve a una subconsulta, que se crea en el área de la tabla (después de FROM) y se le asigna un alias. La clave principal de la tabla articles no es estrictamente necesaria para que la consulta sea editable.

	bill_ID	total
▶	0	439,48
	1	31,71
	2	58,4
	3	3607,7

Registro 1 de 4

```

SELECT
    "sale"."bill_ID",
    SUM( "quantity" * "price" ) AS "total"
FROM "sale", "articles"
WHERE "sale"."articles_ID" = "articles"."ID"
GROUP BY "sale"."bill_ID"

```

Figura 10: La suma calculada debe aparecer en la consulta. La consulta sencilla para la calcular la suma ya no es editable, por que esta agrupada y realiza una suma.

	ID	quantity	articles_ID	price	subtotal
	47	1	6	398	398
	48	1	7	599	599
+	<Campo automa				

Registro 1 de 49

```

SELECT
    "sale"."ID", "sale"."quantity",
    "sale"."articles_ID", "articles"."price",
    "quantity" * "price" AS "subtotal",
    "billtotal"."total"
FROM "sale",
    ( SELECT
        "ID",
        "price"
      FROM "articles" )
  AS "articles",
    ( SELECT
        "sale"."bill_ID",
        SUM( "quantity" * "price" ) AS "total"
      FROM "sale",
        "articles"
      WHERE "sale"."articles_ID" = "articles"."ID"
      GROUP BY "sale"."bill_ID" )
  AS "billtotal"
WHERE "sale"."articles_ID" = "articles"."ID"
      AND "sale"."bill_ID" = "billtotal"."bill_ID"
ORDER BY "sale"."bill_ID", "sale"."ID"

```

Figura 11: Con la segunda subconsulta, lo aparentemente imposible se vuelve posible. La consulta anterior se inserta como una subconsulta en la definición de tabla de la consulta principal (después de "FROM"). Como resultado, toda la consulta permanece editable. En este caso, las entradas solo son posibles en las columnas "quantity" y "articles_ID". Esto se aclara posteriormente en el formulario de consulta.

Resumiendo datos con consultas

Cuando se buscan datos en una base de datos completa, el uso de las funciones de formulario sencillo a menudo genera problemas. Después de todo, un formulario se refiere a una sola tabla, y la función de búsqueda se mueve solo a través de los registros subyacentes para este formulario.

Obtener todos los datos es más fácil mediante el uso de consultas que puedan proporcionar una imagen de todos los registros. La sección «Definir relaciones en la consulta» sugiere una construcción de consulta de este tipo. Esto se construye para la base de datos de ejemplo de la siguiente manera:

```
SELECT "Media"."Title", "Subtitle"."Subtitle", "Author"."Author"
FROM "Media"
  LEFT JOIN "Subtitle"
    ON "Media"."ID" = "Subtitle"."Media_ID"
  LEFT JOIN "rel_Media_Author"
    ON "Media"."ID" = "rel_Media_Author"."Media_ID"
  LEFT JOIN "Author"
    ON "rel_Media_Author"."Author_ID" = "Author"."ID"
```

Aquí todos los títulos, subtítulos y autores se muestran juntos.

La tabla *Media* contiene un total de 9 títulos. Para dos de estos títulos, hay un total de 8 subtítulos. Sin una instrucción LEFT JOIN, ambas tablas que se muestran juntas generarían solo 8 registros. Para cada subtítulo, se busca el título correspondiente y ese es el final de la consulta. Los títulos sin subtítulos no se mostrarían.

Para mostrar todos los artículos, incluidos aquellos sin subtítulos: los artículos están en el lado izquierdo de la tarea, los subtítulos en el lado derecho. Un LEFT JOIN mostrará todos los títulos de los artículos, pero solo un subtítulo para aquellos que tengan un título. Los artículos se convierten en la tabla decisiva para determinar qué registros se mostrarán. Esto ya estaba planeado cuando se construyó la tabla (vea «Capítulo 3, Tablas»). Como existen subtítulos para dos de los nueve títulos, la consulta ahora mostrará $9 + 8 - 2 = 15$ registros.



Nota

La vinculación normal de las tablas, después de contar todas las tablas, sigue a la palabra clave WHERE. Si hay un LEFT JOIN o un RIGHT JOIN, la asignación se define directamente después de los dos nombres de tabla usando ON. La secuencia tiene que ser:

```
Table1 LEFT JOIN Table2 ON Table1.Field1 = Table2.Field1 LEFT JOIN Table3 ON
Table2.Field1 = Table3.Field1 ...
```

Dos títulos de la tabla de artículos aún no tienen una entrada de autor o un subtítulo. Al mismo tiempo, un título tiene un total de tres autores. Si la tabla del autor está vinculada sin un LEFT JOIN, no se mostrarán los dos artículos sin un autor. Pero como un artículo tiene tres autores en lugar de uno, el número total de registros mostrados seguirá siendo 15.

Solo al usar LEFT JOIN se le indicará a la consulta que use la tabla *Media* para determinar qué registros mostrar. Ahora los registros sin subtítulo o autor aparecen nuevamente, dando un total de 17 registros.

El uso de combinaciones apropiadas generalmente aumenta la cantidad de datos que se muestran. Este conjunto de datos ampliado puede explorar fácilmente, ya que se muestran los autores y los subtítulos además de los títulos. En la base de datos de ejemplo, se puede acceder a todas las tablas dependientes de los artículos.

Acceso más rápido a consultas mediante vistas de tabla

Las vistas en SQL son más rápidas que las consultas, especialmente para bases de datos externas, ya que están ancladas directamente en la base de datos y el servidor devuelve los resultados directamente. En el caso de consultas, estas se envían primero al servidor, el servidor las tiene que procesar y luego devolver los datos.

Si una consulta nueva se relaciona con otra consulta, la vista SQL en Base hace que la otra consulta se vea como una tabla. Si crea una Vista a partir de ella, puede ver que realmente está trabajando con una subconsulta (Selección utilizada dentro de otra Selección). Debido a esto, una *Consulta 2* que se relaciona con otra *Consulta 1* no se puede ejecutar mediante la orden **Editar > Ejecutar SQL** directamente, ya que solo la interfaz gráfica de usuario y no la base de datos en sí misma conoce la *Consulta 1*.

La base de datos no le da acceso directo a las consultas. Esto también se aplica al acceso mediante macros. Por otro lado, se puede acceder a las vistas desde macros y tablas. Sin embargo, no se pueden editar registros en una vista. (Deben editarse en una tabla o formulario).



Consejo

Una consulta creada con *Crear consulta en modo SQL* tiene la desventaja de que no se puede ordenar o filtrar con la interfaz. Por lo tanto, hay límites para su uso.

Una vista, por otro lado, se puede administrar en Base igual que una tabla normal, con la excepción de que no es posible cambiar los datos. De esta manera, todas las posibilidades para ordenar y filtrar están disponibles, incluso en órdenes SQL directas.

Además, el formato de las columnas en una vista se conserva cuando se cierra la base de datos, a diferencia de las columnas en una consulta.

Las vistas son una solución para muchas consultas si desea obtener algún resultado. Si, por ejemplo, se va a utilizar una Subselección en los resultados de una consulta, cree una Vista que le brinde estos resultados. Luego use la subselección en la Vista. Los ejemplos correspondientes se encuentran en el «Capítulo 8, Tareas de base de datos».

Crear una vista desde una consulta es bastante fácil y directo: (Debe tener creada una consulta que responda a sus necesidades para pasarla a modo vista).

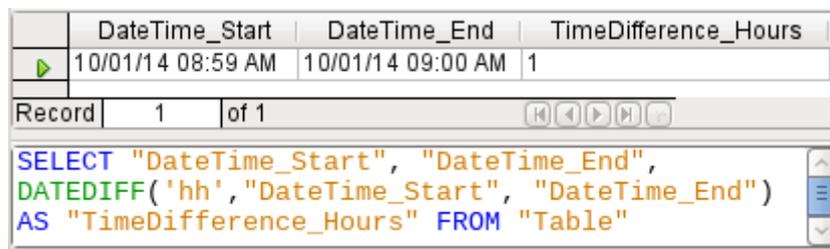
- 1) Haga clic en el objeto *Tablas* en la sección Base de datos.
- 2) Haga clic en *Crear vista*.
- 3) Cierre el diálogo *Agregar tabla*.
- 4) Haga clic en el icono de *Activar o desactivar la vista Diseño*. (Este es el modo SQL para una vista).
- 5) Obtenga el SQL de la consulta para crear la Vista:
 - a) Edite la consulta creada en Vista SQL.
 - b) Use *Control+A* para seleccionar todo el código el SQL de la consulta.
 - c) Use *Control+C* para copiar ese código.
- 6) En el modo SQL de la vista que va a crear use *Control+V* para pegar el código SQL obtenido de la consulta.
- 7) Cierre la vista, (se le preguntará si quiere guardarla) asigne un nombre a la vista y guárdela.

Errores de cálculo en consultas

Las consultas también se utilizan para calcular valores. A veces, la base de datos interna HSQLDB produce errores aparentes, que en un examen más detallado resultan ser interpretaciones correctas (lógicamente) de los datos. También hay problemas de redondeo, que pueden causar confusión fácilmente.

Los datos horarios dentro de HSQLDB se formatean correctamente solo hasta una diferencia de 23:59:59 horas. Si se van a agregar varias veces, por ejemplo, para calcular las horas trabajadas, se debe encontrar otra forma. Aquí hay varios enfoques para resolver estas complicaciones:

- Los datos horarios se expresan directamente solo como un total de minutos o incluso segundos. **Ventaja:** los valores permiten un cálculo posterior sin problemas.
- Los datos se dividen en horas, minutos y segundos, y se concatenan posteriormente como texto usando «:» como separador. **Ventaja:** el texto aparece en las consultas como hora con el formato correcto desde el principio.
- La datos se crean como un número decimal. Un día es 1, una hora es 1/24 y así sucesivamente. **Ventaja:** los valores pueden formatearse posteriormente como dato horario en la consulta y presentarse como un campo formateado de formulario.

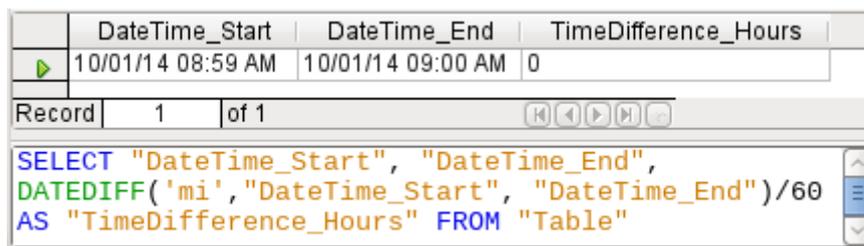


	DateTime_Start	DateTime_End	TimeDifference_Hours
▶	10/01/14 08:59 AM	10/01/14 09:00 AM	1

Record 1 of 1

```
SELECT "DateTime_Start", "DateTime_End",  
DATEDIFF('hh',"DateTime_Start", "DateTime_End")  
AS "TimeDifference_Hours" FROM "Table"
```

DATEDIFF permite determinar los intervalos de tiempo. Pide explícitamente la diferencia que debe determinarse. En el ejemplo anterior, no se han solicitado minutos, pero se consideran todos los elementos que son mayores que un minuto. Eso da una hora (!) Como la diferencia entre 8:59 y 9:00. Por el contrario, una diferencia de fecha se calcula como una diferencia de tiempo en horas. Si, por ejemplo, el campo *Date_time_end* se establece en 2.10.14 09:00, eso se calcula como 25 horas.

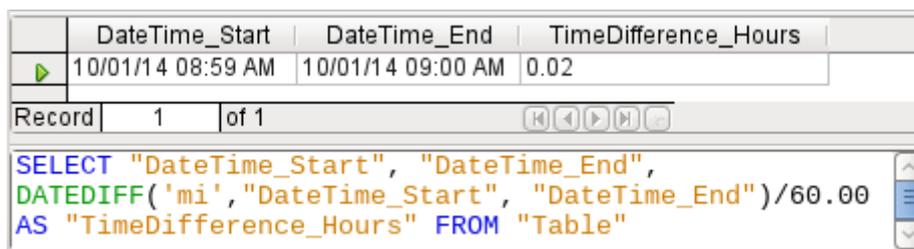


	DateTime_Start	DateTime_End	TimeDifference_Hours
▶	10/01/14 08:59 AM	10/01/14 09:00 AM	0

Record 1 of 1

```
SELECT "DateTime_Start", "DateTime_End",  
DATEDIFF('mi',"DateTime_Start", "DateTime_End")/60  
AS "TimeDifference_Hours" FROM "Table"
```

Si, en cambio, el intervalo de tiempo se calcula en minutos y luego se divide por 60, la diferencia de tiempo en horas se convierte en cero. Esto se parece más al valor correcto, excepto: ¿a dónde fue ese minuto?



	DateTime_Start	DateTime_End	TimeDifference_Hours
▶	10/01/14 08:59 AM	10/01/14 09:00 AM	0.02

Record 1 of 1

```
SELECT "DateTime_Start", "DateTime_End",  
DATEDIFF('mi',"DateTime_Start", "DateTime_End")/60.00  
AS "TimeDifference_Hours" FROM "Table"
```

El intervalo de tiempo se ha dado como un entero. Un entero ha sido dividido por un entero. El resultado de la consulta en tales casos también debe ser un número entero, no un número

decimal. Esto se puede corregir fácilmente. La diferencia de tiempo en minutos se divide por un número decimal con dos lugares decimales (60.00). Esto da un resultado que también tiene dos decimales. 0.02 horas todavía no es exactamente un minuto, pero está mucho más cerca que antes. El número de lugares decimales podría aumentarse usando más ceros. Un período de 0.016 es una aproximación más cercana aún, pero los errores de cálculo posteriores no siempre pueden excluirse.

En lugar de tener que trabajar con muchos ceros agregados, se puede influir en el tipo de datos de DATEDIFF directamente. Usando:

```
(CONVERT(DATEDIFF('mi',"Date_time_start","Date_time_end"), DECIMAL(50,49)))/60
```

puede lograr una precisión de 49 decimales.

Cuando utilice funciones de cálculo, siempre debe comprender que los tipos de datos en HSQLDB tienen una precisión limitada. Independientemente de la cantidad de decimales que utilice, el hecho es que los resultados intermedios que intervienen en un cálculo de tiempo solo se pueden usar de forma limitada para cálculos posteriores.

Si posteriormente se va a utilizar un valor de hora en un formulario o informe como hora formateada, debe asegurarse de que el día sea válido como una base para el formato de hora.

	DateTime_Start	DateTime_End	Time_formatable
	10/01/14 08:59 AM	10/01/14 09:00 AM	00:01

Record 1 of 1

```
SELECT "DateTime_Start", "DateTime_End",
DATEDIFF('mi',"DateTime_Start", "DateTime_End")/1440.0000
AS "Time_formatable" FROM "Table"
```

Aquí la diferencia se calcula en minutos. El resultado se da como una fracción de un día. Un día tiene 60*24 minutos. Si simplemente se divide por 1440, el resultado será cero, por lo que una vez más se deben expresar los lugares decimales explícitamente. Aquí aparece como un tiempo formateado de 0 horas y 1 minuto.

El código de formato para un tiempo superior a un día es [HH]:MM. Si se usa el formato incorrecto, una diferencia horaria de 1 día y 1 minuto podría mostrarse como solo 1 minuto.

	DateTime_Start	DateTime_End	TimeDifference_Minutes	Time_formatable
	10/01/14 08:59 AM	10/01/14 09:09 AM	10	00:09

Record 1 of 1

```
SELECT "DateTime_Start", "DateTime_End",
DATEDIFF('mi',"DateTime_Start", "DateTime_End") AS
"TimeDifference_Minutes",
DATEDIFF('mi',"DateTime_Start", "DateTime_End")/1440.0000
AS "Time_formatable" FROM "Table"
```

¡El demonio del error ataca de nuevo! Una diferencia de tiempo de 10 minutos no debe aparecer como 9 minutos cuando se formatea correctamente. Para descubrir dónde radica el problema, debemos considerar exactamente cómo se realiza el cálculo:

- 1) $10/1440 = 0.00694$. El resultado se redondea a 0.0069 porque solo se especificaron cuatro decimales.
- 2) $0.0069 * 1440 = 9.936$ minutos, que son 9 minutos 56.16 segundos. ¡Y los segundos no se muestran en el formato elegido!

	DateTime_Start	DateTime_End	TimeDifference_Minutes	Time_formatable
	10/01/14 08:59 AM	10/01/14 09:09 AM	10	00:10

Record 1 of 1

```

SELECT "DateTime_Start", "DateTime_End",
DATEDIFF('mi', "DateTime_Start", "DateTime_End") AS
"TimeDifference_Minutes",
DATEDIFF('mi', "DateTime_Start", "DateTime_End")/1440.00000
AS "Time_formatable" FROM "Table"

```

Este error se corrige alargando el divisor con tan solo un decimal (1440.00000 en lugar de 1440.0000). Ahora el redondeo es con cinco cifras $0.00694 * 1440$ da 9.9936 que son 59.616 segundos. El número de segundos se redondea a 60 segundos, por lo que los 9 minutos tienen 1 minuto agregado, lo que hace 10 minutos en total.

Aún puede haber más problemas. ¿Podría haber más decimales que, cuando se formateen, no produzcan 1 minuto?

Para resolver esto, un cálculo corto, el uso de Calc con redondeos similares puede ayudar. En la columna A pondremos una secuencia de números enteros a partir del 1 (serán los minutos). Y en la columna B la fórmula = ROUND (A1 / 1440; 4), formateamos la columna B para que muestre horas y minutos. Si vamos bajando, vemos en la columna A 10 minutos y en la columna B 00:09. Similar desfase se da en el minuto 28, etc. Si redondea a 5 lugares, estos errores desaparecen.

Por muy bueno que sea tener una presentación con un formato adecuado en un formulario, debe tener en cuenta que se trata de valores con redondeo que no son adecuados para su uso posterior en cálculos mecánicos a ciegas. Si se necesita usar un valor para un cálculo posterior, es preferible usar solo una representación de la diferencia de tiempo en las unidades más pequeñas disponibles, en este caso, minutos.