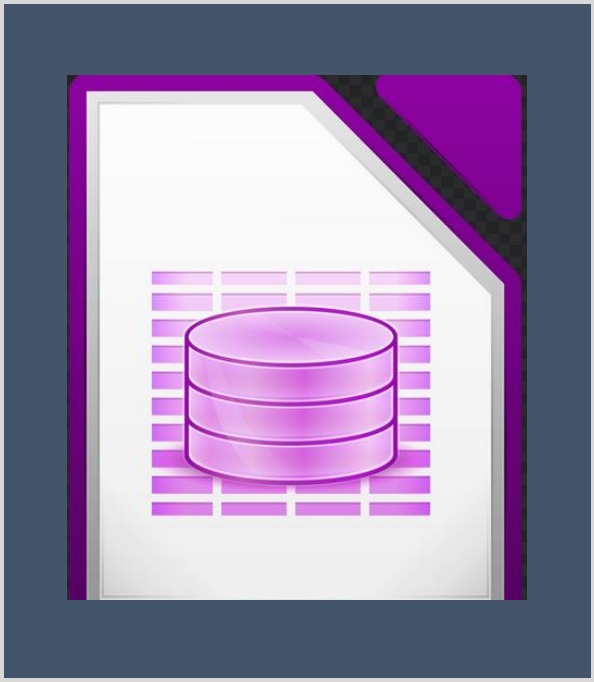
Base



LibreOffice

Working with Database Queries using the Query Design Dialog

General information on queries

Queries to a database are the most powerful tool that we have to use databases in a practical way. They can bring together data from different tables, calculate results where necessary, and quickly filter a specific record from a mass of data. The large Internet databases that people use every day exist mainly to deliver a quick and practical result for the user from a huge amount of information by thoughtful selection of keywords – including the search-related advertisements that encourage people to make purchases.

Entering queries

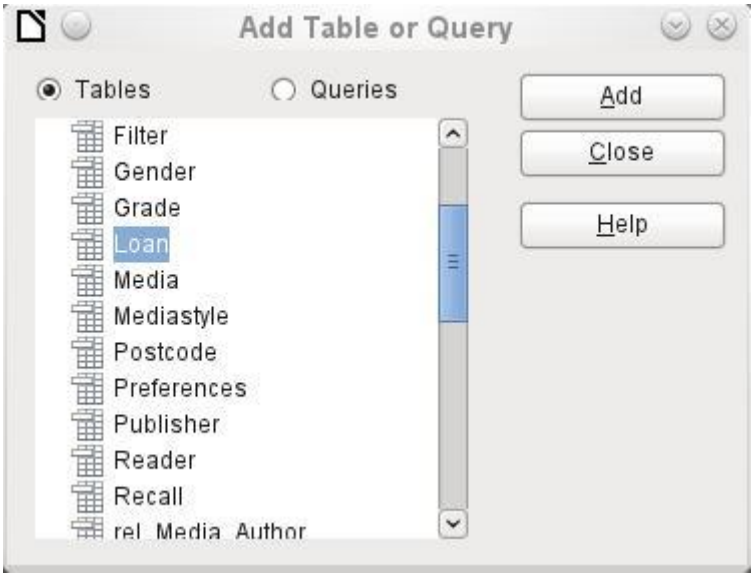
Queries can be entered both in the GUI and directly as SQL code. In both cases a window opens, where you can create a query and also correct it if necessary.

Creating queries using the Query Design dialog

The creation of queries using the Wizard is briefly described in Chapter 8 of the *Getting Started* guide, Getting Started with Base. Here we shall explain the direct creation of queries in Design View.

In the main database window, click the Queries icon in the Databases section, then in the Tasks section, click *Create Query in Design View*. Two dialogs appear. One provides the basis for a design-view creation of the query; the other serves to add tables, views, or queries to the current query.

As our simple form refers to the Loan table, we will first explain the creation of a query using this table.



From the tables available, select the Loan table. This window allows multiple tables (and also views and queries) to be combined. To select a table, click its name and then click the **Add** button. Or, double-click the table's name. Either method adds the table to the graphical area of the Query Design dialog.

When all necessary tables have been selected, click the **Close** button. Additional tables and queries can be added later if required. However no query can be created without at least one table, so a selection must be made at the beginning.

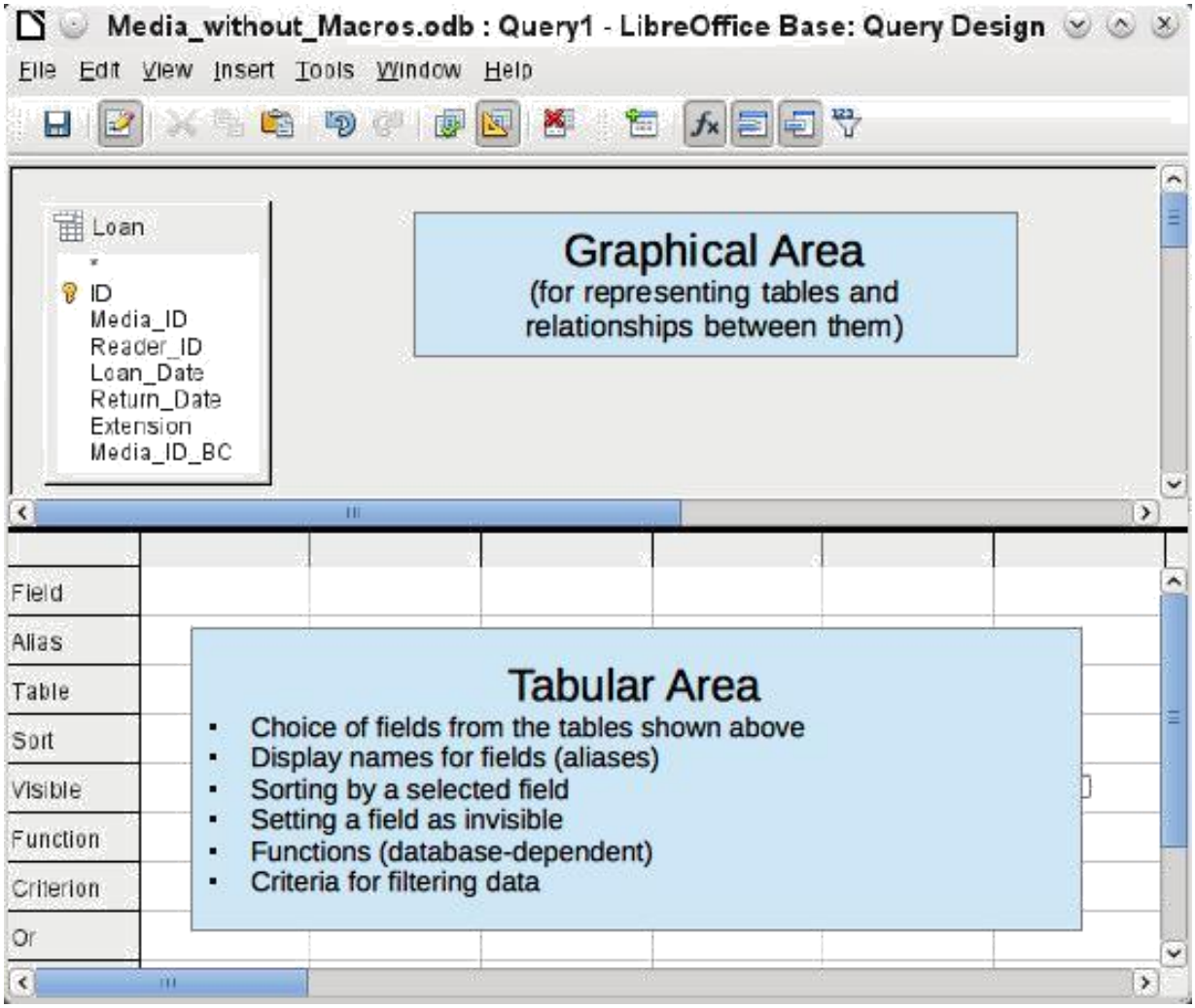
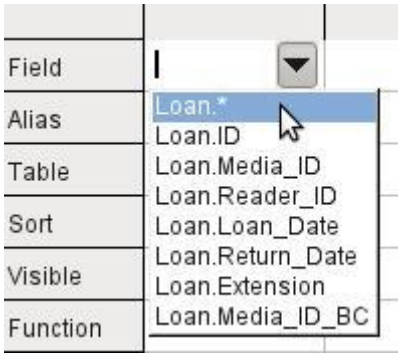


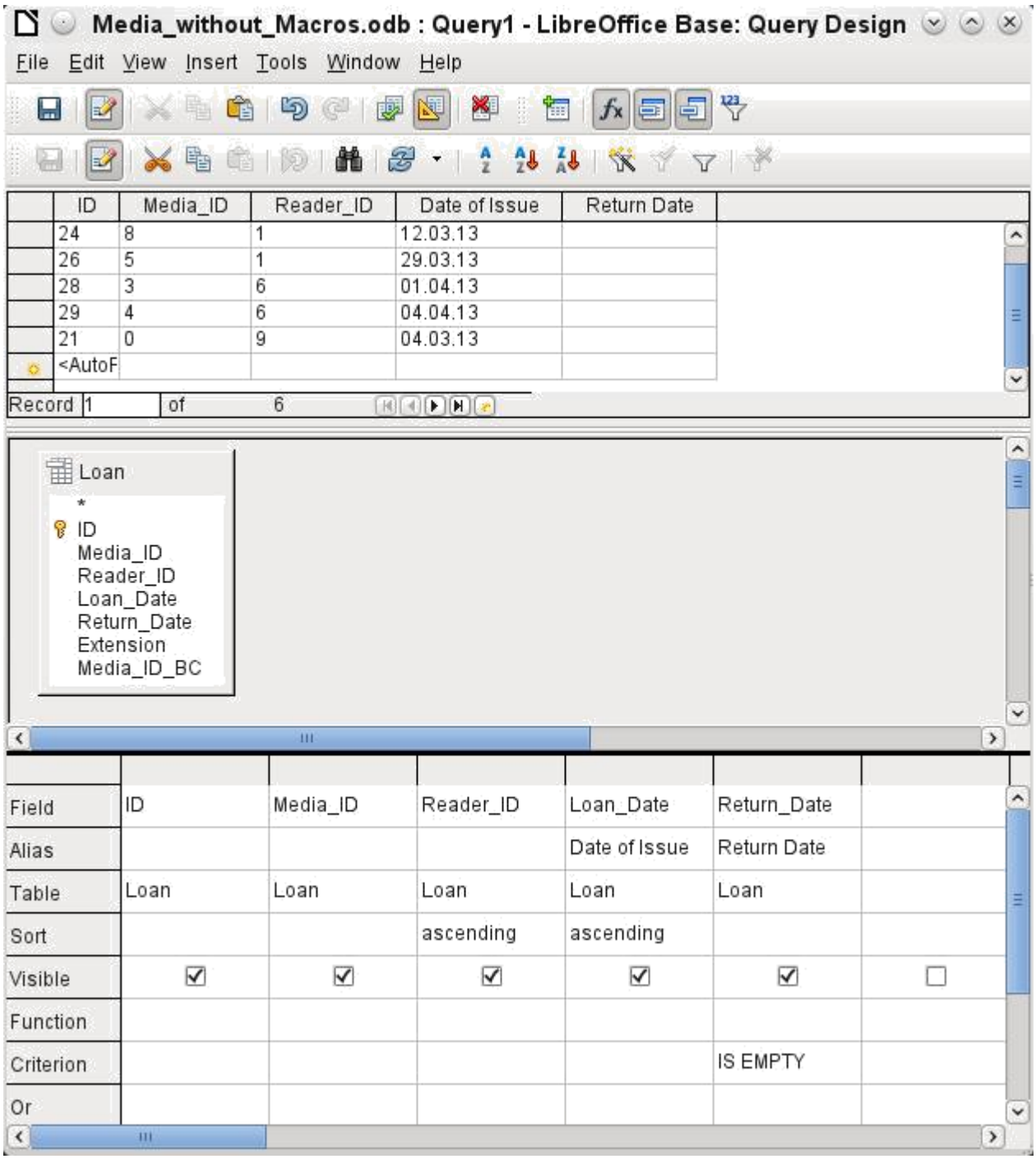
Figure 42: Areas of the Query Design dialog

Figure 42 shows the basic divisions of the Query Design dialog: the graphical area displays the tables that are to be linked to the query. Their relationships to each other in relation to the query may also be shown. The table area is for the selection of fields for display, or for setting conditions related to these fields.

Click on the field in the first column in the table area to reveal a down arrow. Click this arrow to open the drop-down list of available fields. The format is **`Table_name.Field_name`** – which is why all field names here begin with the word *Loan*.

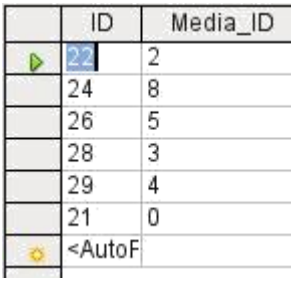


The selected field designation **`Loan.*`** has a special meaning. Here one click allows you to add all fields from the underlying table to the query. When you use this field designation with the *wildcard* * for all fields, the query becomes indistinguishable from the table.



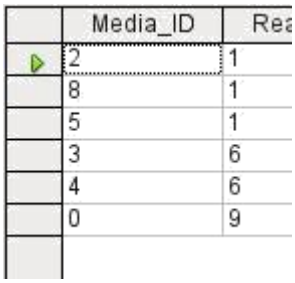
The first five fields of the Loan table are selected. Queries in Design Mode can always be run as tests. This causes a tabular view of the data to appear above the graphical view of the Loan table with its list of fields. A test run of a query is always useful before saving it, to clarify for the user whether the query actually achieves its goal. Often a logical error prevents a query from retrieving any data at all. In other cases it can happen that precisely those records are displayed that you wished to exclude.

In principle a query that produces an error message in the underlying database cannot be saved until the error is corrected.



	ID	Media_ID
▶	22	2
	24	8
	26	5
	28	3
	29	4
	21	0
✱	<AutoF	

Figure 43: An editable query

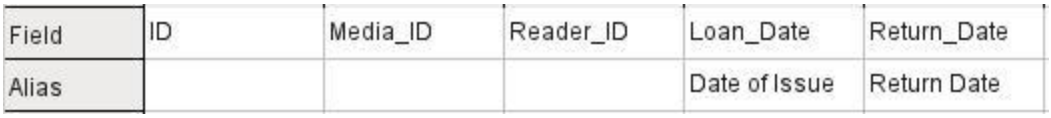


	Media_ID	Rea
▶	2	1
	8	1
	5	1
	3	6
	4	6
	0	9

Figure 44: A non-editable query

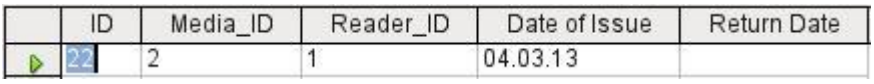
In the above test, special attention should be paid to the first column of the query result. The active record marker (green arrow) always appears on the left side of the table, here pointing to the first record as the active record. While the first field of the first record in Figure 43 is highlighted, the corresponding field in Figure 44 shows only a dashed border. The highlight indicates that this field can be modified. The records, in other words, are editable. The dashed border indicates that this field cannot be modified. Figure 43 also contains an extra line for the entry of a new record, with the ID field already marked as *<AutoField>*. This also shows that new entries are possible.

Tip	A basic rule is that no new entries are possible if the primary key in the queried table is not included in the query.



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

The Loan_Date and Return_Date fields are given aliases. This does not cause them to be renamed but only to appear under these names for the user of the query.

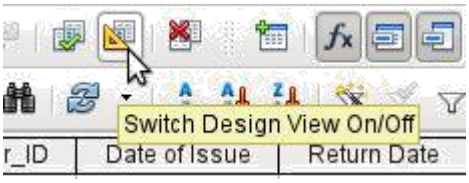


	ID	Media_ID	Reader_ID	Date of Issue	Return Date
▶	22	2	1	04.03.13	

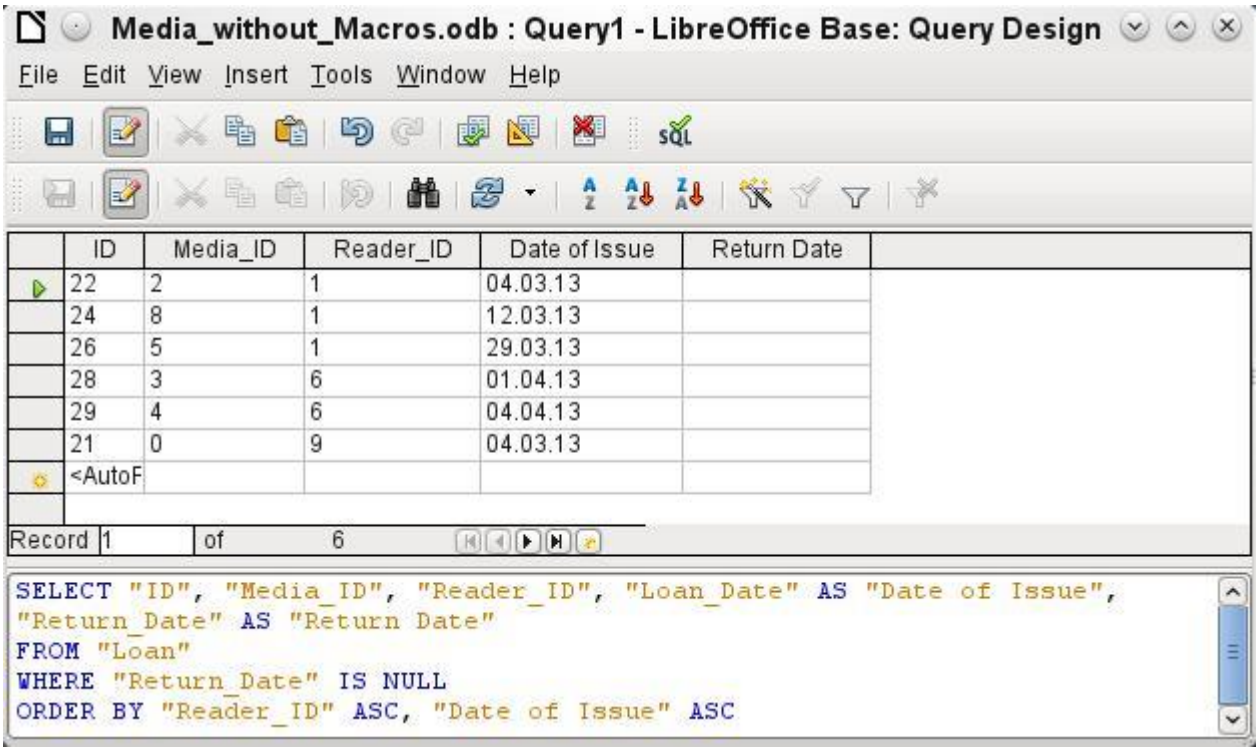
The table view above shows how the aliases replace the actual field names.

Return_Date
Return Date
Loan
☑
IS EMPTY

The Return_Date field is given not just an alias but also a search criterion, which will cause only those records to be displayed for which the Return_Date field is empty. (Enter *IS EMPTY* in the Criterion row of the Return_Date field.) This exclusion criterion will cause only those records to be displayed that relate to media that have not yet been returned from loan.



To learn the SQL language better, it is worth switching from time to time between Design Mode and SQL Mode.



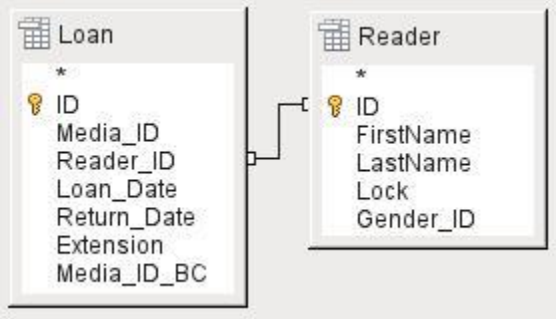
Here the SQL formula created by our previous choices is revealed. To make it easier to read, some line breaks have been included. Unfortunately the editor does not store these line breaks, so when the query is called up again, it will appear as a single continuous line breaking at the window edge.

SELECT begins the selection criteria. *AS* specifies the field aliases to be used. *FROM* shows the table which is to be used as the source of the query. *WHERE* gives the conditions for the query, namely that the Return_date field is to be empty (*IS NULL*). *ORDER BY* defines the sort criteria, namely ascending order (ASC – ascending) for the two fields Reader_ID and Loan date. This sort specification illustrates how the alias for the Loan_Date field can be used within the query itself.

Tip	When working in Design View Mode, use *IS EMPTY* to require a field be empty. When working in SQL Mode, use *IS NULL* which is what SQL (Structured Query Language) requires. When you want to sort by descending order using SQL, use *DESC* instead of *ASC*.

So far the Media_ID and Reader_ID fields are only visible as numeric fields. The readers' names are unclear. To show these in a query, the Reader table must be included. For this purpose we return to Design Mode. Then a new table can be added to the Design view.

Here further tables or queries can subsequently be added and made visible in the graphical user interface. If links between the tables were declared at the time of their creation (see Chapter 3, Tables), then these tables are shown with the corresponding direct links.



If a link is absent, it can be created at this point by dragging the mouse from "Loan"."Reader_ID" to "Reader"."ID".

Now fields from the Reader table can be entered into the tabular area. The fields are initially added to the end of the query.

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

The position of the fields can be corrected in the tabular area of the editor using the mouse. So for example, the First_name field has been dragged into position directly before the Loan_date field.

	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	

Record 1 of 6

Now the names are visible. The Reader_ID has become superfluous. Also sorting by Surname and First_name makes more sense than sorting by Reader_ID.

This query is no longer suitable for use as a query that allows new entries into the resulting table, since it lacks the primary key for the added Reader table. Only if this primary key is built in, does the query become editable again. In fact it then becomes completely editable so that the readers' names can also be altered. For this reason, making query results editable is a facility that should be used with extreme caution, if necessary under the control of a form.

also **Caution**

Having a query that you can edit can create problems. Editing data in the query edits data in the underlying table and the records contained in the table. The data may not have the same meaning. For example, change the name of the reader, and you have also changed what books the reader has borrowed and returned.

If you have to edit data, do so in a form so you can see the effects of editing data.

Even when a query can be further edited, it is not so easy to use as a form with list boxes, which show the readers' names but contain the Reader_ID from the table. List boxes cannot be added to a query; they are only usable in forms.

```
SELECT "Loan"."ID", "Loan"."Media_ID", "Loan"."Reader_ID",
"Reader"."FirstName", "Reader"."LastName", "Loan"."Loan_Date" AS "Date of
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 of Issue" ASC
```

If we now switch back to SQL View, we see that all fields are now shown in double quotes: **"Table_name"."Field_name"**. This is necessary so that the database knows from which table the previously selected fields come from. After all, fields in different tables can easily have the same field names. In the above table structure this is particularly true of the ID field.

Note

The following query works without putting table names in front of the field names:

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

Here the ID is taken from the table which comes first in the FROM definition. The table definition in the WHERE Formula is also superfluous, because stockID only occurs once (in the Dispatch table) and ID was clearly taken from the Stock table (from the position of the table in the query).

If a field in the query has an alias, it can be referred to – for example in sorting – by this alias without a table name being given. Sorting is carried out in the graphical user interface according to the sequence of fields in the tabular view. If instead you want to sort first by "Loan date" and then by "Loan"."Reader_ID", that can be done if:

- The sequence of fields in the table area of the graphical user interface is changed (drag and drop "Loan date" to the left of "Loan"."Reader_ID", or
- An additional field is added, set to be invisible, just for sorting (however, the editor will register this only temporarily if no alias was defined for it) [add another "Loan date" field just before "Loan"."Reader_ID" or add another "Loan"."Reader_ID" field just after "Loan date"], or
- The text for the ORDER BY command in the SQL editor is altered correspondingly (ORDER BY "Loan date", "Loan"."Reader_ID").

Specifying the sort order *may not be completely error-free*, depending on the LibreOffice version. From version 3.5.3, sorting from the SQL view is correctly registered and displayed in the graphical user interface, including the fields that are used in the query but are not visible in the query output. (These fields do not have a check in the Visible row.)

Tip

A query may require a field that is not part of the query output. In the graphic in the next section, Return_Date is an example. This query is searching for records that do **not** contain a return date. This field provides a criterion for the query but no useful visible data.

Using functions in a query

The use of functions allows a query to provide more than just a filtered view of the data in one or more tables. The following query calculates how many media have been loaned out, depending on the Reader_ID.

Field	ID	Reader_ID	Return_Date
Alias	Count		
Table	Loan	Loan	Loan
Sort			
Visible	☑	☑	☐
Function	Count	Group	
Criterion			IS EMPTY

For the ID of the Loan table, the *Count* function is selected. In principle it makes no difference which field of a table is chosen for this. The only condition is: *The field must not be empty in any of the records*. For this reason, the primary key field, which is never empty, is the most suitable choice. All fields with a content other than NULL are counted.

For the Reader_ID, which gives access to reader information, the *Grouping* function is chosen. In this way, the records with the same Reader_ID are grouped together. The result shows the number of records for each Reader_ID.

As a search criterion, the Return_Date is set to “IS EMPTY”, as in the previous example. (Below, the SQL for this is *WHERE “Return_Date” IS NULL*.)

	Count	Reader_ID
▶	1	9
	3	1
	2	6

Record 1 of 3

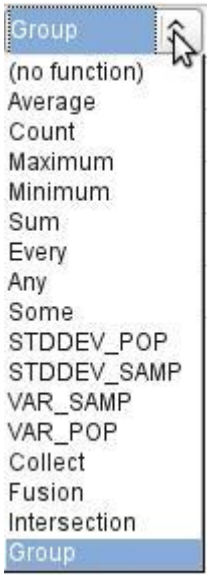
```
SELECT COUNT( "ID" ) AS "Count", "Reader_ID"
FROM "Loan"
WHERE "Return_Date" IS NULL
GROUP BY "Reader_ID"
```

The result of the query shows that Reader_ID '0' has a total of 3 media on loan. If the *Count* function had been assigned to the Return_Date instead of the ID, every Reader_ID would have '0' media on loan, since Return_date is predefined as NULL.

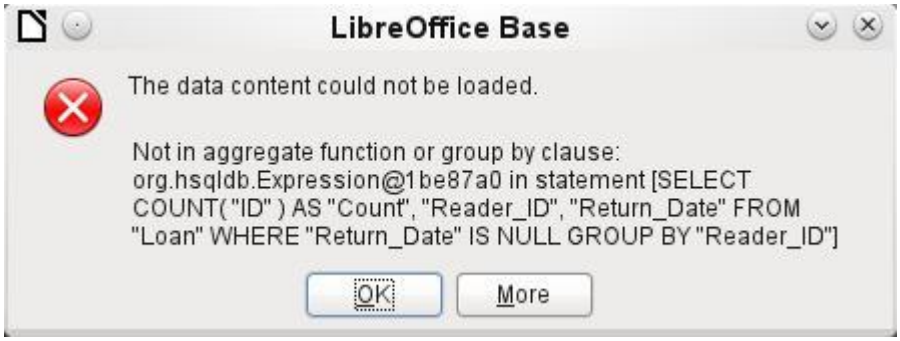
The corresponding Formula in SQL code is shown above.

Altogether the graphical user interface provides the following functions, which correspond to functions in the underlying HSQLDB.

For an explanation of the functions, see “Query enhancement using SQL Mode“ on page 134.



If one field in a query is associated with a function, all the remaining fields mentioned in the query must also be associated with functions if they are to be displayed. If this is not ensured, you get the following error message:



A somewhat free translation would be: The following expression contains no aggregate function or grouping.

Tip	When using Design View Mode, a field is only visible if the *Visible* row contains a check mark for the field. When using SQL Mode, a field is only visible when it follows the keyword, SELECT.

Note	When a field is **not** associated with a function, the number of rows in the query output is determined by the search conditions. When a field is associated with a function, the number of rows in the query output is determined by whether there is any grouping or not. If there is no grouping, there is only one row in the query output. If there is grouping, the number of rows matches the number of distinct values that the grouping field has. So, all of the visible fields must either be associated with a function or **not be associated with a function** to prevent this conflict in the query output. After this, the complete query is listed in the error message, but unfortunately without the offending field being named specifically. In this case the field Return_Date has been added as a displayed field. This field has no function associated with it and is not included in the grouping statement either. The information provided by using the More button is not very illuminating for the normal database user. It just displays the SQL error code.

To correct the error, remove the check mark in the Visible row for the Return_Date field. Its search condition (Criterion) is applied when the query is run, but it is not visible in the query output.

Using the GUI, basic calculations and additional functions can be used.

Field	ID	Media_ID	Reader_ID	Date	Count("Recall"."Date") * 2	Return_Date
Alias				RecallCount	RecallAmount	
Table	Loan	Loan	Loan	Recall		Loan
Sort						
Visible	☑	☑	☑	☑	☑	☐
Function	Group	Group	Group	Count		
Criterion						IS EMPTY

Suppose that a library does not issue recall notices when an item is due for return, but issues overdue notices in cases where the loan period has expired and the item has not been returned. This is common practice in school and public libraries that issue loans only for short, fixed periods. In this case the issue of an overdue notice automatically means that a fine must be paid. How do we calculate these fines?

In the query shown above, the Loan and Recalls tables are queried jointly. From the count of the data entries in the table Recalls, the total number of recall notices is determined. The fine for overdue media is set in the query to 2.00 €. Instead of a field name, the field designation is given as **`Count(Recalls.Date)*2`**. The graphical user interface adds the quotation marks and converts the term “count” into the appropriate SQL command.

Caution

Only for people who use a comma for their decimal separator:

If you wish to enter numbers with decimal places using the graphical user interface, you must ensure that a decimal point rather than a comma is used to separate the decimal places within the final SQL statement. Commas are used as field separators, so new query fields are created for the decimal part.

An entry with a comma in the SQL view always leads to a further field containing the numerical value of the decimal part.

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

Record 1 of 2

```
SELECT "Loan"."ID", "Loan"."Media_ID", "Loan"."Reader_ID",
COUNT( "Recall"."Date" ) AS "RecallCount",
COUNT( "Recall"."Date" ) * 2 AS "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"
```

The query now yields for each medium still on loan the fines that have accrued, based on the recall notices issued and the additional multiplication field. The following query structure will also be useful for calculating the fines due from individual users.

Field	Reader_ID	Date	Count("Recall"."Date") * 2	Return_Date
Alias		RecallCount	RecallAmount	
Table	Loan	Recall		Loan
Sort				
Visible	☑	☑	☑	☐
Function	Group	Count		
Criterion				IS EMPTY

The "Loan"."ID" and "Loan"."Media_ID" fields have been removed. They were used in the previous query to create by grouping a separate record for each medium. Now we will be grouping only by the reader. The result of the query looks like this:

	Reader_ID	RecallCount	RecallAmount
▶	1	2	4

Record 1 of 1

Instead of listing the media for Reader_ID = 0 separately, all the "Recalls"."Date" fields have been counted and the total of 8.00€ entered as the fine due.

Relationship definition in the query

When data is searched for in tables or forms, the search is usually limited to one table or one form. Even the path from a main form to a subform is not navigable by the built-in search function. For such purposes, the data to be searched for are better collected by using a query.

	Title
▶	Der kleine Hobbit
	Das sogennante 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

Record 1 of 9

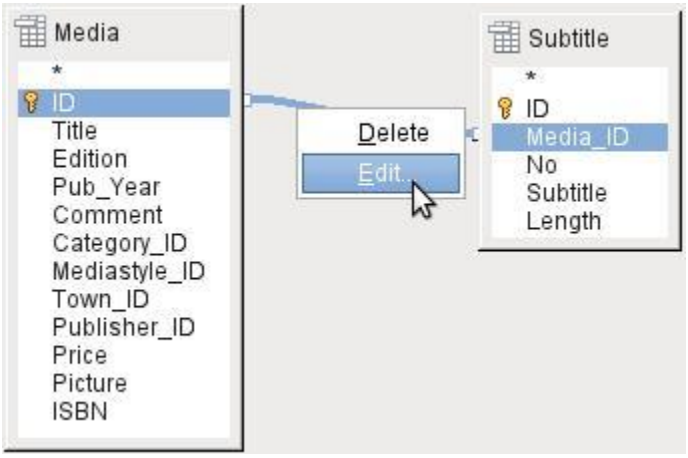
Field	Title	
Alias		
Table	Media	
Sort		
Visible	☑	☐

	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

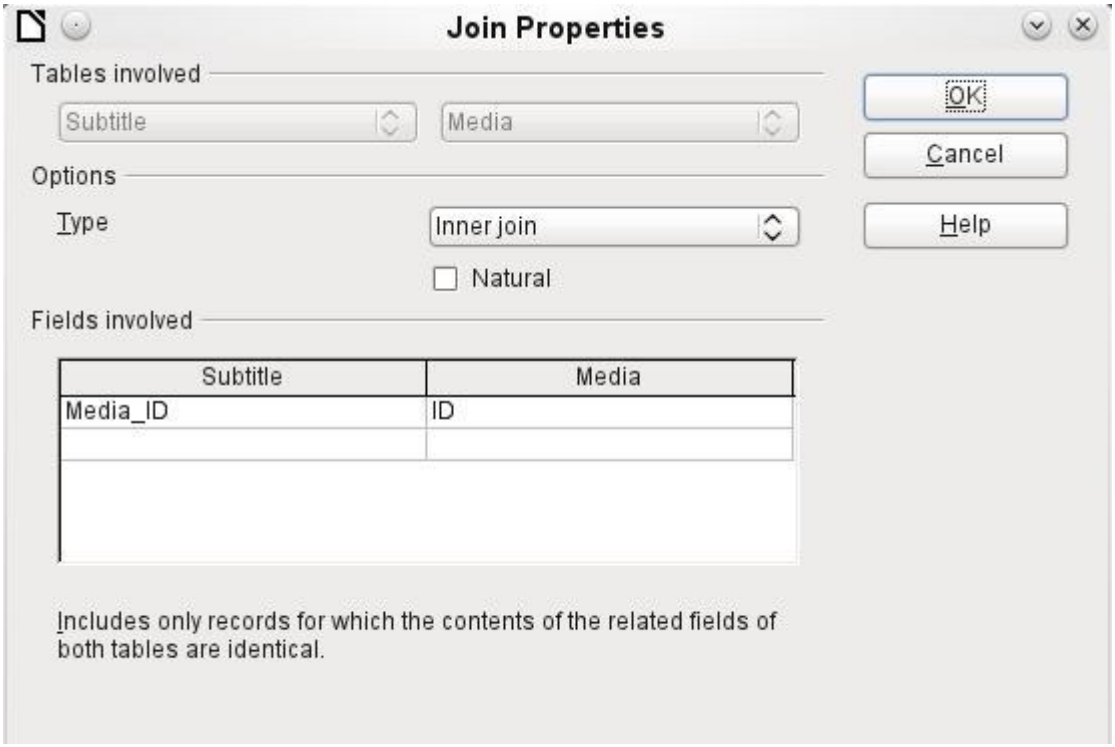
Record 1 of 8

Field	Title	Subtitle	
Alias			
Table	Media	Subtitle	
Sort			
Visible	☑	☑	☐

The simple query for the Title field from the Media table shows the test entries for this table, 9 records in all. But if you enter Subtitle into the query table, the record content of the Media table is reduced to only 2 Titles. Only for these two Titles are there also Subtitles in the table. For all the other Titles, no subtitles exist. This corresponds to the join condition that only those records for which the Media_ID field in the Subtitle table is equal to the ID field in the Media table should be shown. All other records are excluded.



The join conditions must be opened for editing to display all the desired records. We refer here **not** to joins between tables in *Relationship design* **but** to joins within *queries*.



By default, relationships are set as *Inner Joins*. The window provides information on the way this type of join works in practice.

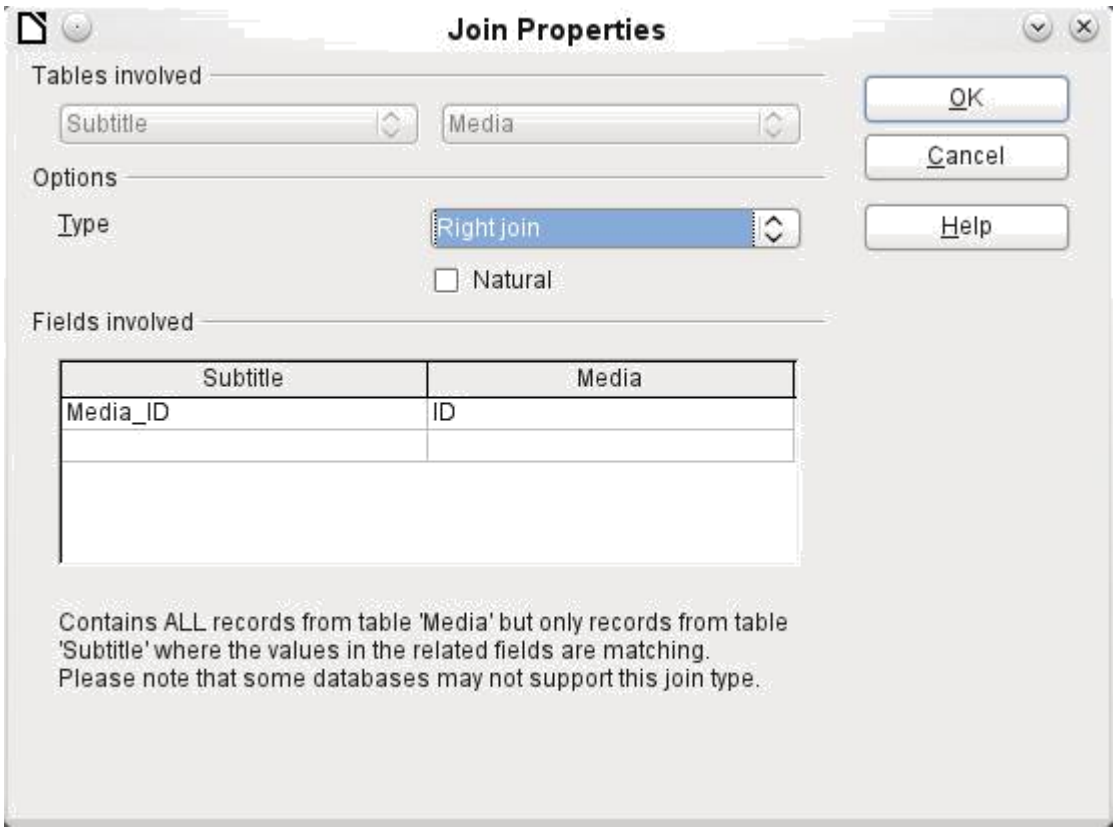
The two previously selected tables are listed as Tables Involved. They are not selectable here. The relevant fields from the two tables are read from the table definitions. If there is no relationship specified in the table definition, one can be created at this point for the query. However, if you have planned your database in an orderly manner using HSQLDB, there should be no need to alter these fields.

The most important setting is the *Join* option. Here relationships can be so chosen that all records from the Subtitle table are selected, but only those records from Media which have a subtitle entered in the Subtitle table.

Or you can choose the opposite: that in any case all records from the table Media are displayed, regardless of whether they have a subtitle.

The *Natural* option specifies that the linked fields in the tables are treated as equal. You can also avoid having to use this setting by defining your relationships properly at the very start of planning your database.

For the type *Right join* , the description shows that all records from the Media table will be displayed (Subtitle RIGHT JOIN Media). As there is no Subtitle that lacks a title in Media but there are certainly Titles in Media that lack a Subtitle, this is the right choice.



After confirming the *right join* the query results look as we wanted them. Title and Subtitle are displayed together in one query. Naturally Titles appear more than once as with the previous relationship. However as long as hits are not being counted, this query can be used further as a basis for a search function. See the code fragments in this chapter, in Chapter 9 (Macros), and in Chapter 8 (Database Tasks).

	Title	Subtitle
▶	Der kleine Hobbit	
	Das sogennante 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

Record 1 of 15

Source: - Libre Office Base Handbook Version 4.0
https://documentation.libreoffice.org/assets/Uploads/Documentation/en/BH4.0/PDF/BH40BaseHandbook.pdf