LibreOffice Community

LibreOffice Documentation Team

Base Guide

7.3

Copyright

Contributors

To this edition

Steve Fanning
Olivier Hallot
Vasudev Narayanan
flywire

To previous editions

Steve Fanning
Pulkit Krishna
Peter Schofield
Jost Lange
Steve Schwettman
Jean-Pierre Ledure
Jenna Sargent
Jean Hollis Weber
Jochen Schiffers
Martin Fox
Alain Romedenne
Drew Jensen
Dan Lewis
Robert Großkopf
Hazel Russman
Andrew Pitonyak
Randolph Gamo

Feedback

Please direct any comments or suggestions about this document to the Documentation Team's mailing list: documentation@global.libreoffice.org.

Note

Everything you send to a mailing list, including your email address and any other personal information that is written in the message, is publicly archived and cannot be deleted.

Publication date and software version

Published June 2022. Revision 0. Based on LibreOffice 7.3 Community.
Other versions of LibreOffice may differ in appearance and functionality.

book=1

Documentation for LibreOffice is available at https://documentation.libreoffice.org/en/

Contents

LibreOffice Community

Base Guide 7.3

Preface

Who is this book for?

Anyone who wants to get up to speed quickly with LibreOffice Base will find this book valuable. Whether you have never worked with databases before, or have worked with them in a DBMS (Database Management System), or you are used to another database system from an office suite or a stand-alone database system such as MySQL, this book is for you. You may wish to first read Chapter 8, Getting Started with Base, in the *Getting Started Guide*.

What's in this book?

This book introduces Base, the database component of LibreOffice. Base uses the HSQLDB (HyperSQL DataBase) database engine to create database documents. It can access databases created by many database programs, including Microsoft Access, MySQL, Oracle, Firebird, and PostgreSQL. Base includes additional functionality that allows you to create full data-driven applications.

✓ **Note**

In addition to the HSQLDB engine, an experimental facility is available to create database documents using an embedded Firebird database engine. Advanced users can enable experimental facilities by selecting the **Enable experimental facilities** checkbox on the **Tools > Options > LibreOffice > Advanced** dialog.

This book introduces the features and functions of Base, using a set of sample databases.

- Creating a database
- Accessing external databases
- Creating and using tables in relational databases
- Creating and using forms for data entry
- Using queries to bring together data from different tables, calculate results where necessary, and quickly filter a specific record from a mass of data
- Creating reports using the Report Builder
- Linking databases to other documents and external forms, including use in mail merge
- Filtering and searching data
- Using macros to prevent input errors, simplify tasks, and improve usability of forms
- Maintaining databases

Sample databases

A set of sample databases has been created to accompany this book. You can find them here: https://wiki.documentfoundation.org/images/5/52/Sample-databases.zip.

Where to get more help

This book, the other LibreOffice user guides, the built-in Help system, and user support systems assume that you are familiar with your computer and basic functions such as starting a program, opening and saving files.

Help system

LibreOffice comes with an extensive Help system. This is the first line of support for using LibreOffice. Windows and Linux users can choose to download and install the offline Help for use when not connected to the Internet; the offline Help is installed with the program on macOS.

To display the Help system, press *F1* or select **Help > LibreOffice Help** on the Menu bar. If you do not have the offline help installed on your computer and you are connected to the Internet, your default browser will open the online Help pages on the LibreOffice website.

The Help menu (Figure 1) includes links to other LibreOffice information and support facilities.

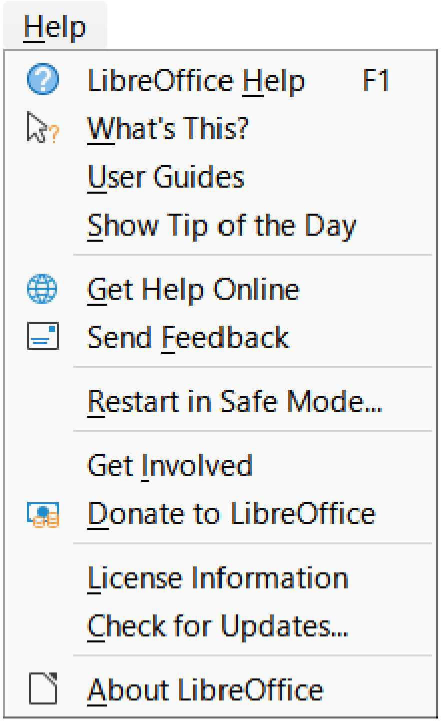


Figure 1: The Help menu

Note

The options indicated by a ‡ symbol in the list below are only accessible if your computer is connected to the Internet.

- **What’s This?** – For quick tips when a toolbar is visible, place the mouse pointer over any of the icons to see a small box (“tooltip”) with a brief explanation of the icon’s function. For a more detailed explanation, select **Help > What’s This?** and hold the pointer over the icon. In addition, you can choose whether to activate extended tips using **Tools > Options > LibreOffice > General > Extended tips**.
- **User Guides**‡ – Opens your default browser at the English Documentation page of the LibreOffice website https://documentation.libreoffice.org/en/english-documentation/. There you will find copies of user guides and other useful information.
- **Show Tip of the Day** – Opens a dialog showing a useful bit of information to help expand your knowledge of LibreOffice.
- **Get Help Online**‡ – Opens your default browser at the Ask LibreOffice forum of questions and answers from the LibreOffice community https://ask.libreoffice.org/en/questions/.
- **Send Feedback**‡ – Opens your default browser at the Feedback page of the LibreOffice website https://www.libreoffice.org/get-help/feedback/. From there you can report bugs, suggest new features, and communicate with others in the LibreOffice community.

- **Restart in Safe Mode** – Opens a dialog where you can restart LibreOffice and reset the software to its default settings.
- **Get Involved**‡ – Opens your default browser at the Get Involved page of the LibreOffice website https://www.libreoffice.org/community/get-involved/. There you can choose a topic of interest to help improve the program.
- **Donate to LibreOffice**‡ – Opens your default browser at the Donation page of the LibreOffice website https://donate.libreoffice.org/.
- **License Information** – Outlines the licenses under which LibreOffice is made available.
- **Check for Updates**‡ – Opens a dialog and checks the LibreOffice website for updates to your version of the software.
- **About LibreOffice** – Opens a dialog, which displays information about the version of LibreOffice and the operating system you are using. This information will often be requested if you ask the community for help or assistance with the software. A button is provided to enable you to copy this information to the clipboard so that you can subsequently paste it into a forum post, an email, or a bug report. (On macOS, this item is under **LibreOffice** on the Menu bar.)

Other free online support

The LibreOffice community not only develops software, but provides free, volunteer-based support. In addition to the Help menu links above, other online community support options are available, see Table 1.

Table 1: Free LibreOffice support

Service	***Description***
FAQs	Answers to frequently asked questions https://wiki.documentfoundation.org/Faq
Mailing lists	Free community support is provided by a network of experienced users https://www.libreoffice.org/get-help/mailing-lists/
Questions & Answers and Knowledge Base	Free community assistance is provided in a question and answer formatted web service. Search similar topics or open a new one in https://ask.libreoffice.org/en/questions. The service is available in several other languages; just replace /en/ with de, es, fr, ja, ko, nl, pt, tr, and many others in the web address above.
Native language support	The LibreOffice website in various languages https://www.libreoffice.org/community/nlc/ Mailing lists for native languages https://wiki.documentfoundation.org/Local_Mailing_Lists Information about social networking https://wiki.documentfoundation.org/Website/Web_Sites_services
Accessibility options	Information about available accessibility options https://www.libreoffice.org/get-help/accessibility/

Paid support and training

You can also pay for support through service contracts from a vendor or consulting firm specializing in LibreOffice. For information about certified professional support, see The Document Foundation’s website: https://www.documentfoundation.org/gethelp/support/.

For schools, educational and research institutions, and large organizations, see https://www.libreoffice.org/download/libreoffice-in-business/.

What you see may be different

Illustrations

LibreOffice runs on Windows, Linux, and macOS operating systems, each of which has several versions and can be customized by users (fonts, colors, themes, window managers). The illustrations in this guide were taken from a variety of computers and operating systems. Therefore, some illustrations will not look exactly like what you see on your computer display.

Also, some of the dialogs may be different because of the settings selected in LibreOffice. You can either use dialogs from your computer's operating system or from LibreOffice. The differences affect mainly Open, Save, and Print dialogs. To change which dialogs are used, go to **Tools > Options > LibreOffice > General** and select or deselect the option **Use LibreOffice dialogs**.

Icons

The LibreOffice community has created icons for several icon sets: Colibre, Breeze, Breeze Dark, Elementary, Karasa Jaga, Sifr, Sifr Dark, and Sukapura. Each user can select a preferred set. The icons in this guide have been taken from a variety of LibreOffice installations that use different sets of icons. The icons for some of the many tools available in LibreOffice may then differ from the ones used in this guide.

To change the icon set used, go to **Tools > Options > LibreOffice > View**. Choose from the drop-down list under *Icon Style*.

✓ **Note**

The Galaxy, Oxygen, and Tango icon sets are no longer included as part of the standard installation package. They can be added back by downloading and installing the following extensions:
https://extensions.libreoffice.org/extensions/galaxy-icon-theme
https://extensions.libreoffice.org/extensions/oxygen-icon-theme
https://extensions.libreoffice.org/en/extensions/show/tango-icon-theme-for-libreoffice
Some of the previously included gallery backgrounds are now only available as an extension from:
https://extensions.libreoffice.org/extensions/legacy-gallery-backgrounds

Using LibreOffice on macOS

Some keystrokes and menu items are different on macOS from those used in Windows or Linux. Table 2 gives some common substitutions for the instructions in this document. For a more detailed list, see the application Help.

Table 2: Example keystrokes for different environments

Windows or Linux	*macOS equivalent*	*Effect*
Tools > Options menu selection	**LibreOffice > Preferences**	Access setup options
Right-click	*Control*+click and/or right-click depending on computer setup	Open a context menu
Ctrl (Control)	⌘ (Command)	Used with other keys

Windows or Linux	*macOS equivalent*	*Effect*
F11	⌘+*T*	Open the Styles deck in the Sidebar

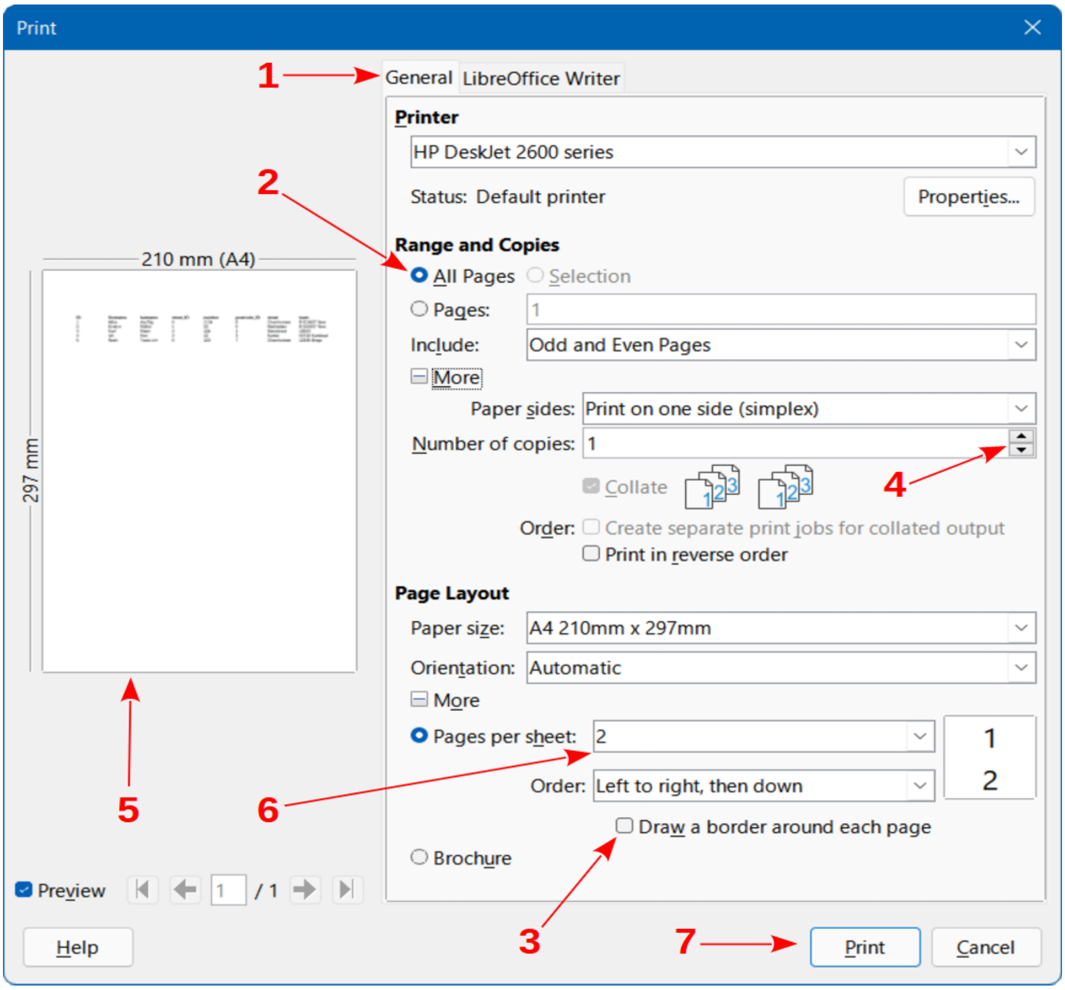
What are all these things called?

The terms used in LibreOffice for most parts of the user interface (the parts of the program you see and use, in contrast to the behind-the-scenes code that actually makes it work) are the same as for most other programs.

A dialog is a special type of window. Its purpose is to inform you of something, or request input from you, or both. It provides controls to use to specify how to carry out an action. Some common controls are highlighted by callouts in Figure 2, with a brief technical description given in the key that follows the figure. In most cases the technical terms are not used in this book, but it is useful to know them because the Help and other sources of information often use them.

In most cases, you can interact only with the dialog (not the document itself) as long as the dialog remains open. When you close the dialog after use (usually, clicking **OK** or another button saves your changes and closes the dialog), then you can again work with the document.

Some dialogs can be left open as you work, so you can switch back and forth between the dialog and your document.



Print

General | LibreOffice Writer

Printer

HP DeskJet 2600 series

Status: Default printer

Range and Copies

● All Pages ○ Selection

○ Pages: 1

Include: Odd and Even Pages

⊟ More

Paper sides: Print on one side (simplex)

Number of copies: 1

☑ Collate

Order: □ Create separate print jobs for collated output

□ Print in reverse order

Page Layout

Paper size: A4 210mm x 297mm

Orientation: Automatic

⊟ More

● Pages per sheet: 2

Order: Left to right, then down

□ Draw a border around each page

○ Brochure

☑ Preview 1 / 1

Figure 2: Dialog showing common controls

1) Tabbed page (not strictly speaking a control).
2) Radio buttons (only one can be selected at a time).

3) Checkbox (more than one can be selected at a time).

4) Spin box (click the up and down arrows to change the number shown in the text box next to it, or type in the text box).

5) Thumbnail or preview.

6) Drop-down list from which to select an item.

7) Push buttons.

Frequently asked questions

How is LibreOffice licensed?

LibreOffice is distributed under the Open Source Initiative (OSI) approved Mozilla Public License (MPL). See https://www.libreoffice.org/about-us/licenses/.

It is based on code from Apache OpenOffice made available under the Apache License 2.0 but also includes software that differs from version to version under a variety of other Open Source licenses. New code is available under GNU Lesser General Public License (LGPL) 3.0 and MPL 2.0.

May I distribute LibreOffice to anyone? May I sell it? May I use it in my business?

Yes.

How many computers may I install it on?

As many as you like.

Is LibreOffice available in my language?

LibreOffice has been translated (localized) into over 100 languages, so your language is probably supported. Additionally, there are over 100 spelling, hyphenation, and thesaurus dictionaries available for languages, and dialects that do not have a localized program interface. The dictionaries are available by selecting the Dictionary tag filter from the LibreOffice Extension Center https://extensions.libreoffice.org/.

How can you make it for free?

LibreOffice is developed and maintained by volunteers and has the backing of several organizations.

I am writing a software application. May I use programming code from LibreOffice in my program?

You may, within the parameters set out in the licenses: https://www.mozilla.org/MPL/2.0/ https://www.libreoffice.org/about-us/licenses/.

Why do I need Java to run LibreOffice? Is it written in Java?

LibreOffice is not written in Java; it is mostly written in the C++ language. Java is one of several languages that can be used to extend the software. The Java JRE (Java Runtime Environment) is only required for some features. The most notable one is the HSQLDB relational database engine used by the Base component of LibreOffice.

Java is available at no cost. More information and download links to the appropriate edition for your operating system can be found at: https://java.com/en/download/manual.jsp.

Note

If you want to use LibreOffice features that require Java it is important that the correct 32 bit or 64 bit edition matches the installed version of LibreOffice.

How can I contribute to LibreOffice?

You can help with the development and user support of LibreOffice in many ways, and you do not need to be a programmer. To start, check out this webpage: https://www.libreoffice.org/community/get-involved/.

May I distribute the PDF of this book, or print and sell copies?

Yes, as long as you meet the requirements of one of the licenses in the copyright statement at the beginning of this book. You do not have to request special permission. We request that you share with the project some of the profits you make from sales of books, in consideration of all the work we have put into producing them.

Donate to LibreOffice: https://www.libreoffice.org/donate/.

What’s new in LibreOffice Base 7.3 Community?

LibreOffice 7.3 Community includes many changes not visible in the user interface. These changes include improved interoperability with Microsoft’s proprietary file formats and performance improvements in handling large files, opening certain .docx and .xlsx files, managing font caching, and opening presentations and drawings that contain large images.

More information is in the announcement:

https://blog.documentfoundation.org/blog/2022/02/02/libreoffice-73-community/

Release Notes are here: https://wiki.documentfoundation.org/ReleaseNotes/7.3

This user guide is based on *Base Guide 7.2*. The information in the chapters and appendixes that follow this Preface is unchanged from *Base Guide 7.2* – the main work carried out on Base since 7.2 has been concerned with on-going development of the experimental facility to create database documents with Firebird.

What's new in this book

Since the publication of the Base Guide 7.2, this book underwent a review for accuracy and completeness in a continuous effort to bring you the best Guide for the LibreOffice Base module.

Chapter 2

Added SQLite settings.

Review of the chapter.

Chapter Appendix A

Added introduction text.

This book is in permanent update and revision for publication.

Feedback

The LibreOffice Documentation team is happy to receive feedback from our readers. Please send your comments and reports on mistakes to the documentation mailing list, by sending a mail to documentation@global.libreoffice.org or better, subscribe to the mailing list and be part of the documentation team.

Subscription: send an empty mail to documentation+subscribe@global.libreoffice.org .

Digest subscription: send an empty mail to documentation+subscribe-digest@global.libreoffice.org .

Archives: http://listarchives.libreoffice.org/global/documentation/ .

Mail-Archive.com: http://www.mail-archive.com/documentation@global.libreoffice.org/ .

Spend a little time lurking at the archives and send a mail to the list introducing yourself.

Base Guide 7.3

Chapter 1
Introducing Base

Introduction

What is Base?

Base is the database front-end component of LibreOffice, a freely available, fully-featured, open-source office productivity suite. Base provides these important features:

- **Database creation**. New embedded databases can be created using an embedded HSQLDB database engine.
- **Connection to other data sources**. Connections can be made to many widely employed database engines and other data sources, including spreadsheets, text documents, and address books. External data sources must be created and properly configured before Base can make such a connection.
- **Tables**. In many databases data is organized in tables with rows and columns. In general terms, the columns represent categories of data, while the rows represent individual instances. Tables in a relational database are connected by relationships and these allow the data to be shared between tables, eliminating data redundancy.
- **Forms**. A form is a user-friendly window that is used to view and edit the data in the tables of a database. The form helps the user to enter data quickly and accurately, ensuring that data updates are implemented consistently across all affected tables within the database.
- **Queries**. A query retrieves data from the database, calculates results where necessary, and filters records of interest from a large volume of data. Queries can bring together data from one or more tables, sometimes via another query or view.
- **Views**. For some connected database types, it is possible to create database views. A view is a read-only, virtual table containing rows and columns. It is constructed using a query that gathers data physically stored in one or more “real” tables. Most queries created with Base are stored in the Base document, but a view’s underlying query is stored within the database engine.
- **Reports**. A report is used to extract key data from the database and present it in an easy to read format, often for consumption by people (such as decision makers) not otherwise directly involved with the database.
- **Linking**. Databases can be linked to other documents and external forms, including use in mail merge.
- **Filtering and searching**. Filters are applied to limit the volume of data displayed, focusing attention on records that match specific filter criteria. Searching is used to locate records of particular interest that match specific search criteria.
- **Macros**. LibreOffice Basic and Python macros can be used to simplify running repetitive tasks preventing input errors, increase functionality, and improve usability. LibreOffice includes the ScriptForge and Access2Base macro libraries, which provide an extensive collection of resources that can be invoked from Basic or Python scripts.

✓ **Note**

At this issue, the majority of macros in this document are coded in LibreOffice Basic. More information about using Python scripts can be found in the Help system.

- **Database maintenance**. This includes compacting a database, resetting autovalues, exporting data, and locating unnecessary entries.

Using the correct tool for the job

In everyday office operation, spreadsheets are regularly used to aggregate sets of data and perform analyses on them. As the data in a spreadsheet is laid out in a table view, plainly visible, and easily edited or extended, many users ask why they should use a database instead of a spreadsheet. This book explains the differences between these tools and provides readers with information that will help them decide on the best solution for their specific requirements.

A database will provide advantages to many users and a well-designed database will minimize duplication, errors, and inconsistencies which may be hard to identify and eliminate in a spreadsheet. However, some potential database users may be put off by a lack of familiarity with database concepts and terminology. Users who are concerned about a steep learning curve should be reassured that:

- The database concepts and terminology used in Base are largely consistent with those used in many other common database systems. This means that much of the information about database technologies that is freely available on the Internet can be helpful in understanding aspects of Base.
- As a key component of the integrated LibreOffice suite, users who are familiar with the other components should have no trouble interacting with Base, which utilizes the menus and toolbars of the classic LibreOffice user interface. Because of the nature of Base, it does not provide (or need) the user interface selection options (such as tabbed variants) that are available in some of the other LibreOffice components.
- Base provides wizards to help users who are new to database design (or just new to the software) to create databases, tables, queries, forms, and reports, and it comes supplied with a set of predefined table definitions for tracking assets, customers, sales orders, invoices, and many other common and useful items.
- In addition there are many other sources of assistance, as described in the “Where to get more help” section of the Preface to this book.

Sample databases

This chapter introduces one of the sample databases that is used throughout this book – *Media_without_Macros.odb*. This database is designed to assist with operating a media library and everything concerned with it. Facilities include:

- Capture of media details.
- Capture of information about authors and borrowers.
- Circulation desk management functions.

In addition, this chapter uses the sample *Example_Sport.odb* database, which is designed to assist with the organization of a sports competition.

Useful background information

You can use Base in two distinct ways:

1) Base can act as a front end for a wide range of database engines, providing a user interface that supports the creation and use of elements such as queries, forms, reports, and macros. In this role as a database front-end, core (back-end) database functions such as the storage and organization of data are carried out by the database engine rather than Base. In this way a Base user can interact with existing local or networked databases, which may be multi-user assets shared with other members of their organization.

2) Base can also operate in a standalone mode with an embedded Java-based HSQLDB (HyperSQL DataBase, version 1.8.0) relational database engine in a standard LibreOffice distribution. This is an ideal solution for everyday jobs, and for people requiring an easy-to-understand, simple-to-use, single-user database system instead of communicating with an external database engine. More information about HSQLDB can be found at http://hsqldb.org/.

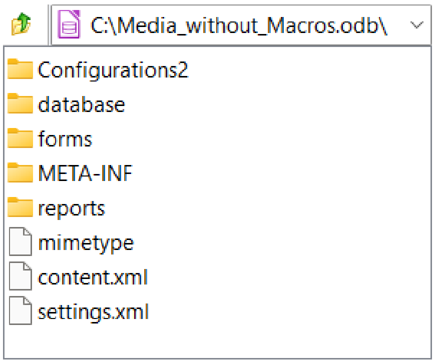
Work continues to improve the integration of the non-Java Firebird relational database (version 3.0.7) as an alternative embedded engine within Base. Until this work is completed, the ability to create a database with Firebird is available only as an experimental feature. However, Base can connect to a Firebird database engine through an external server without enabling experimental features. More information about Firebird can be found at https://firebirdsql.org/.

Base can connect to some of the most widely employed multi-user database engines. The list of available database types varies according to operating system, but the following should always be included:

- MariaDB – https://mariadb.org/
- MySQL – https://www.mysql.com/
- PostgreSQL – https://www.postgresql.org/

In addition, the built-in support for JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity) standard drivers allows Base to connect to many other existing database engines. Base can also connect to other data sources, including spreadsheets, text documents, and address books.

Base stores its database information in files having the extension ".odb" that are formatted in accordance with the standard *Open Document Format for Office Applications (OpenDocument) Version 1.2. Part 3: Packages*. ODB files created by Base contain a collection of files and folders that are compressed using ZIP lossless data compression (Figure 3). An ODB file might contain many files, including, for example, data relating to an embedded database engine, forms, queries, reports, and macros. If the file relates to an external database rather than an embedded engine, then it will contain information about the database connection rather than the data stored in the database's tables.



C:\Media_without_Macros.odb\

Configurations2
database
forms
META-INF
reports
mimetype
content.xml
settings.xml

Figure 3: Top-level structure of the folders and files within an ODB file

Correct operation of some Base functionality is dependent on a Java Runtime Environment (JRE). If a JRE is not available to LibreOffice, then the following facilities will be unavailable:

- The embedded HSQLDB database.
- The Table, Query, and Form Wizards.
- The Report Builder.
- Database access via Java Database Connectivity (JDBC).

Base will present appropriate error dialogs to warn of the unavailability of a JRE. If a JRE is installed but not enabled in LibreOffice, it can be enabled by selecting **Tools > Options > LibreOffice > Advanced** on the Menu bar and clicking the **Use a java runtime environment** option.

Oracle Java can be downloaded from https://www.java.com/en/download/manual.jsp. If you use the 32-bit version of LibreOffice, you should install the JRE 32-bit version. If you use the 64-bit version of LibreOffice, you should install the JRE 64-bit version.

Caution

The Oracle download site defaults to serving the 32-bit JRE – take care to follow the correct links if you need to access the 64-bit JRE download, which should contain the string “x64” within its filename. A 64-bit version of LibreOffice will not recognize a 32-bit JRE (and vice versa) and this is a common source of problems.

For macOS, you need to install Oracle’s Java Development Kit (JDK), not just the JRE. Download links can be found by selecting the macOS option at: https://www.oracle.com/java/technologies/downloads/.

If your data is stored in a relational database, you will inevitably want to take advantage of facilities for the direct entry of SQL commands. SQL stands for Structured Query Language and is a standard language (ISO/IEC 9075) for accessing and manipulating data in relational databases. This book contains many examples of SQL statements. Do not worry if you are initially unfamiliar with the syntax and semantics of SQL, there are good, free training materials available on the Internet. For example, the SQL Tutorial provided by W3Schools at https://www.w3schools.com/sql/default.asp is a good starting point.

Parts of the main Base window

The main Base window is shown in Figure 4 and its features are described in the following subsections.

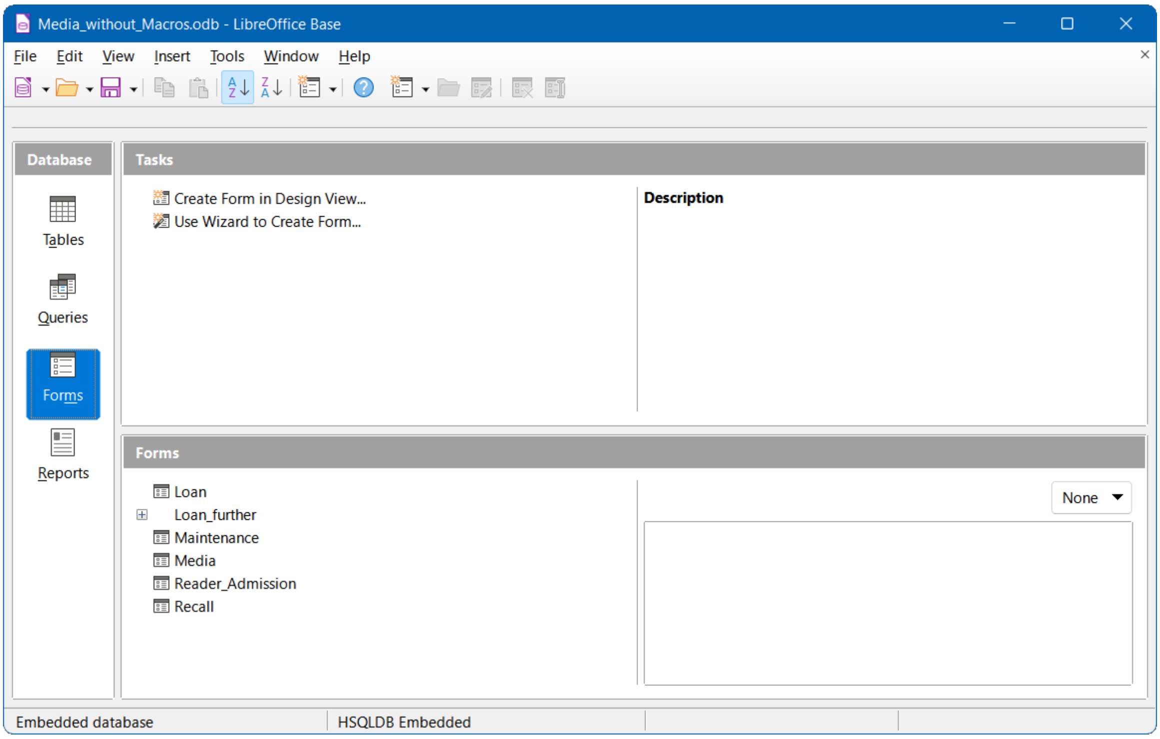


Figure 4: The main Base window

Title bar

The Title bar is located at the top of the Base window. It shows the file name of the current database document.

Menu bar

The Menu bar is located just below the Title bar in Windows and Linux and at the top of the screen in macOS. When you select one of the menus, a submenu drops down to show further options, including:

- Commands that directly cause an action, such as **Close** or **Save**, in the **File** menu.
- Commands that open dialogs. These are indicated by an ellipsis (...) following a command, such as **Open...** in the **File** menu.
- Commands that open further submenus. These are indicated by a right-pointing arrow following a command, such as **Toolbars** and **Preview**, in the **View** menu. Moving the cursor onto one of these options causes its submenu to open.

The default configuration of the menus on the Menu bar is likely to be suitable for most Base users. However, you can customize the Menu bar using the *Menus* tab of the **Tools > Customize** dialog – see the system Help for more details of this facility.

Toolbars

In addition to the Menu bar, Base provides several toolbars that provide quick access to frequently used options. Placing the mouse cursor over a toolbar icon displays a small box called a tooltip that names the button's function. To close a tooltip, move away from the underlying component or press the *Esc* key.

The default setting when Base opens a database document is for the Standard toolbar to be displayed at the top of the workspace (below the Menu bar) along with the Form toolbar. The Form toolbar is replaced by either the Table, Query, or Report toolbar as different selections are made in the *Database* area of the main window.

The default Standard toolbar is shown in Figure 5 and provides the following options (left to right): New; Open; Save; Copy; Paste; Sort Ascending; Sort Descending; Form; LibreOffice Help.

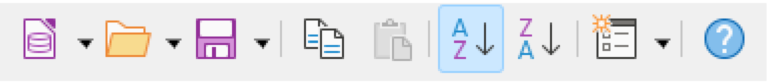


Figure 5: Standard toolbar

The default Table toolbar is shown in Figure 6 and provides the following options (left to right): New Table Design; Open Database Object; Edit; Delete; Rename.

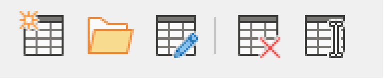


Figure 6: Table toolbar

The default Query toolbar is shown in Figure 7 and provides the following options (left to right): New Query (Design View); New Query (SQL View); Open Database Object; Edit; Delete; Rename.

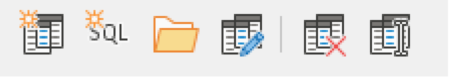


Figure 7: Query toolbar

The default Form toolbar is shown in Figure 8 and provides the following options (left to right): Form; Open Database Object; Edit; Delete; Rename.

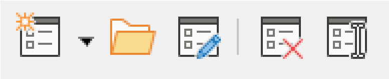


Figure 8: Form toolbar

The default Report toolbar is shown in Figure 9 and provides the following options (left to right): Report; Report Wizard; Open Database Object; Edit; Delete; Rename.

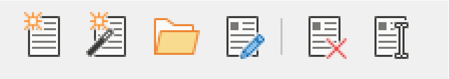


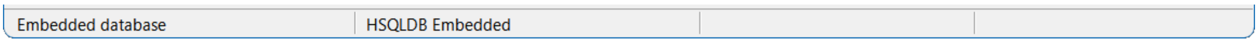
Figure 9: Report toolbar

Options to dock / undock and lock / unlock toolbars are provided in the menus of the main Base window. However, these are included only for consistency with toolbars in other components of LibreOffice and are not normally needed for Base.

The default configuration of the toolbars is likely to be suitable for most Base users. However, you can customize the toolbars using the *Toolbars* tab of the **Tools > Customize** dialog – see the system Help for more details of this facility.

Status bar

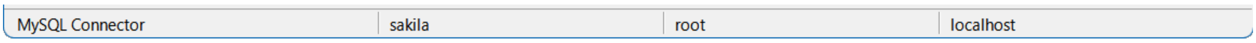
The Status bar at the bottom of the workspace provides information about the open database, with the information displayed dependent on the type of that database. For example, Figure 10 shows the Status bar when the open database was created with the embedded HSQLDB database engine.



Embedded database | HSQLDB Embedded

Figure 10: Status bar for database created using embedded HSQLDB

Figure 11 shows the Status bar when the open database is a MySQL database. Base has connected to a database called *sakila* (a sample database supplied as part of the MySQL distribution) that is available through a local MySQL server (connection name *localhost*, logged in with the *root* username).



MySQL Connector | sakila | root | localhost

Figure 11: Status bar for MySQL database

Database area

The *Database* area provides four buttons labeled **Tables**, **Queries**, **Forms**, and **Reports** (the same options are also available through the **View** menu on the Menu bar). Use these buttons to select the work area that is relevant to your current task. An alternative keyboard shortcut to action one of the buttons in the Database area is to press the *Alt* key together with the underlined letter in the button's legend (for example, pressing *Alt+Q* on the keyboard is equivalent to clicking the **Queries** button with the mouse).

Depending on the work area selected, various tasks—creating new content or calling up existing elements—may be carried out. As different selections are made in the *Database* area, Base updates the contents of the two areas to the right to reflect the new work area.

Although tables are at the heart of any database, Base defaults to the forms work area when opening an existing database because forms are the elements most used when working with databases. With forms, you can make entries into the tables and analyze table content.

Tasks area

The *Tasks* area provides several options to select common actions within the selected work area. The options for each work area are as follows:

Tables

- Create Table in Design View…
- Use Wizard to Create Table…
- Create View…

Queries

- Create Query in Design View…
- Use Wizard to Create Query…
- Create Query in SQL View…

Forms

- Create Form in Design View…
- Use Wizard to Create Form…

Reports

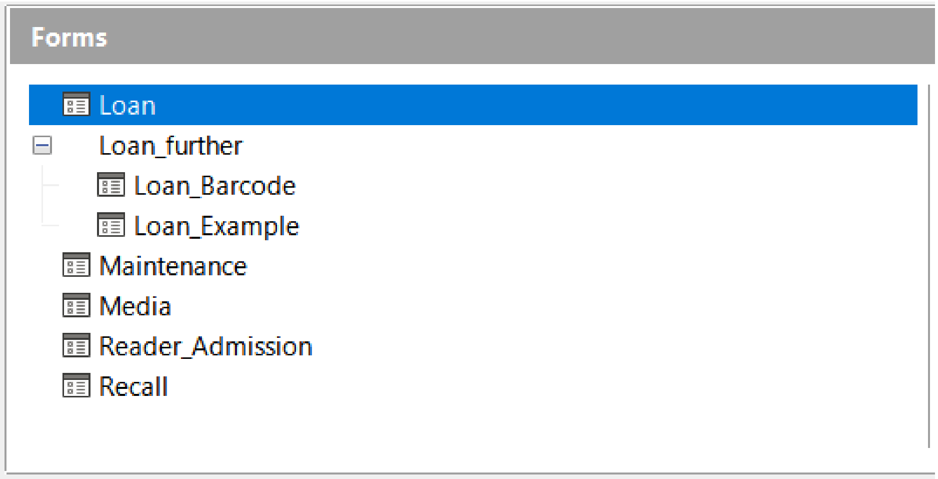
- Create Report in Design View…
- Use Wizard to Create Report…

To the right of these links is an area entitled *Description*. As the mouse moves over one of the available options, this area is updated to give a brief text description of the cursor-indicated option.

Tables / Queries / Forms / Reports area

The name at the top of the area below *Tasks* changes to reflect the selected work area and hence is one of *Tables*, *Queries*, *Forms*, or *Reports*.

The left part of the area lists the defined elements of the relevant type. For the tables and queries work areas, this is an alphabetical list of the available elements. For the forms and reports work areas, the respective elements are arranged alphabetically within a folder structure (Figure 12). This is achieved either directly when saving the elements using the Save dialog or by creating new folders using **Insert > Folder** on the Menu bar.



Forms

Loan

Loan_further

Loan_Barcode

Loan_Example

Maintenance

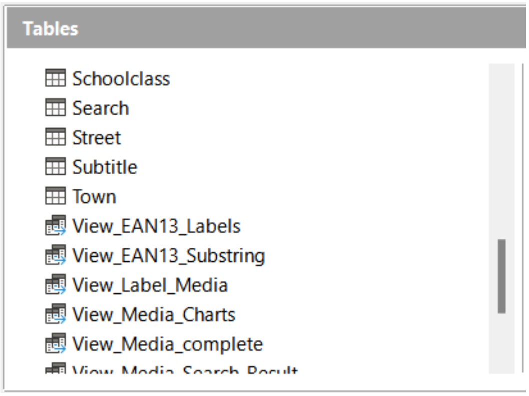
Media

Reader_Admission

Recall

Figure 12: Folder structure in the Forms area

For the tables work area, if the database contains views then they are listed with the tables. The two types are distinguished by the use of different icons. Figure 13 shows both tables and views in the *Media_without_Macros.odb* database.



Tables

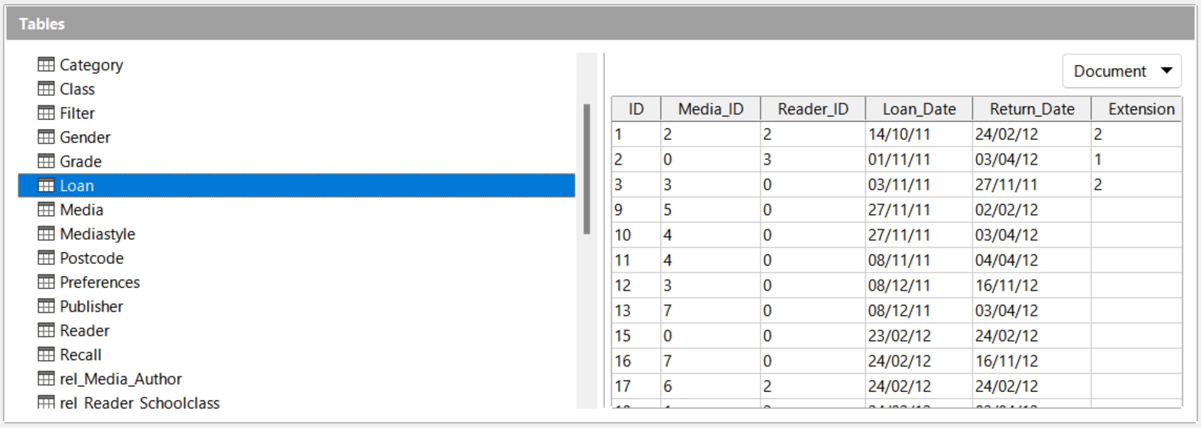
Schoolclass
Search
Street
Subtitle
Town
View_EAN13_Labels
View_EAN13_Substring
View_Label_Media
View_Media_Charts
View_Media_complete
View_Media_Search_Result

Figure 13: Tables area including views

The right section of the area provides additional information controlled by a drop-down menu of two or three options, depending on the work area selected.

For all work areas, the option **None** may be selected if you do not wish to clutter the screen with additional information.

For all work areas, the option **Document** may be selected to provide a preview of the selected element, perhaps assisting in identification of an element of interest. Figure 14 for example shows a preview of the *Loan* table of the *Media_without_Macros.odb* database.



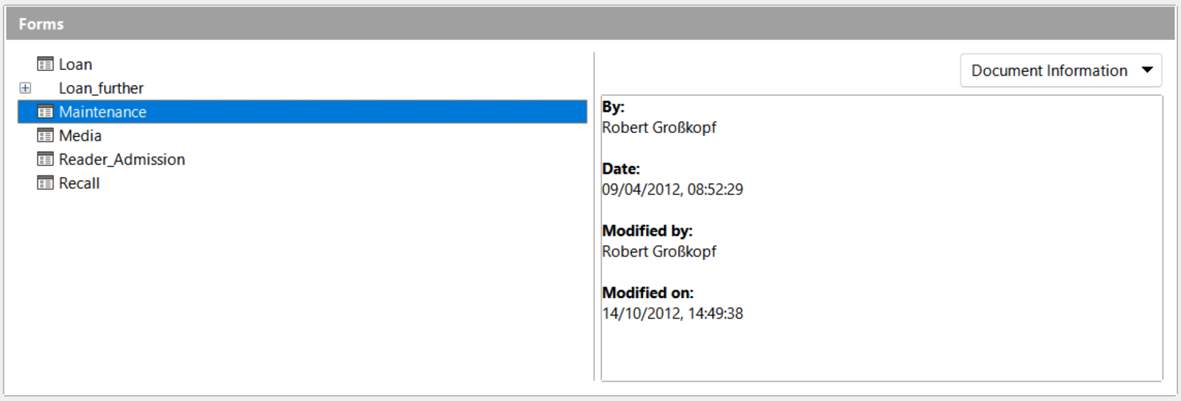
Tables

Category
Class
Filter
Gender
Grade
Loan
Media
Mediastyle
Postcode
Preferences
Publisher
Reader
Recall
rel_Media_Author
rel_Reader_Schoolclass

ID	Media_ID	Reader_ID	Loan_Date	Return_Date	Extension
1	2	2	14/10/11	24/02/12	2
2	0	3	01/11/11	03/04/12	1
3	3	0	03/11/11	27/11/11	2
9	5	0	27/11/11	02/02/12	
10	4	0	27/11/11	03/04/12	
11	4	0	08/11/11	04/04/12	
12	3	0	08/12/11	16/11/12	
13	7	0	08/12/11	03/04/12	
15	0	0	23/02/12	24/02/12	
16	7	0	24/02/12	16/11/12	
17	6	2	24/02/12	24/02/12	

Figure 14: Document preview in Tables area

For the forms and reports work areas, an additional **Document Information** option is available. Figure 15 shows an example of the date and name information provided for the *Maintenance* form of the *Media_without_Macros.odb* database.



Forms

Loan
Loan_further
Maintenance
Media
Reader_Admission
Recall

By:
Robert Großkopf

Date:
09/04/2012, 08:52:29

Modified by:
Robert Großkopf

Modified on:
14/10/2012, 14:49:38

Figure 15: Document information in Forms area

Base – a container for database content

Data input using forms

A form is an element that is used to enter, display, or edit data in a database. Although it is possible to enter data directly into tables, for large tables it can sometimes be hard to be certain that the data is being placed in the correct location. An effective form can speed up and simplify the user’s tasks, reducing the possibility of entering erroneous data.

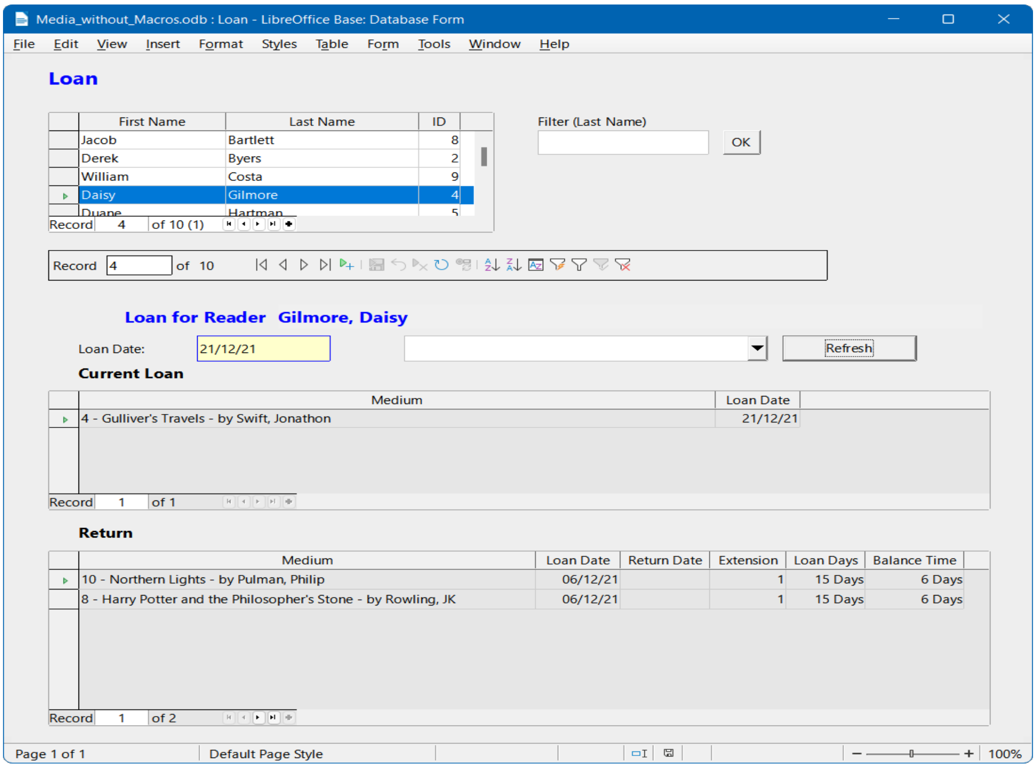
To open a form, click **Forms** in the *Database* area of the main Base window, right-click the name of the form in the *Forms* area, and select **Open** in the context menu. It is common in all LibreOffice components for the user interface to provide multiple methods to achieve the same objective. For example, once **Forms** has been clicked in the *Database* area of the main Base window, Base provides the following alternative methods to open a specific form:

1) Double-click the name of the form in the *Forms* area.
2) Click the name of the form in the *Forms* area and then click the **Open Database Object** icon on the Form toolbar.
3) Click the name of the form in the *Forms* area and then select **Edit > Open Database Object** on the Menu bar.

Note

To avoid unnecessary repetition, the remainder of this book will usually identify just one of the user interface methods provided to achieve a particular objective. The described method is not intended as a recommendation of “best practice” – that is a matter of personal preference.

A simple form may show information from just one table; more complex forms can contain much more. Figure 16 shows the *Loan* form in the *Media_without_Macros.odb* sample database, which has been extended to show information from multiple tables and include additional controls.


Media_without_Macros.odb : Loan - LibreOffice Base: Database Form
File Edit View Insert Format Styles Table Form Tools Window Help
Filter (Last Name)
OK
Refresh
Page 1 of 1
Default Page Style
100%

Loan

First Name	Last Name	ID
Jacob	Bartlett	8
Derek	Byers	2
William	Costa	9
Daisy	Gilmore	4
Duane	Hartman	5

Record 4 of 10 (1)

Record 4 of 10

Loan for Reader Gilmore, Daisy

Loan Date: 21/12/21

Current Loan

Medium	Loan Date
4 - Gulliver's Travels - by Swift, Jonathon	21/12/21

Record 1 of 1

Return

Medium	Loan Date	Return Date	Extension	Loan Days	Balance Time
10 - Northern Lights - by Pulman, Philip	06/12/21		1	15 Days	6 Days
8 - Harry Potter and the Philosopher's Stone - by Rowling, JK	06/12/21		1	15 Days	6 Days

Record 1 of 2

Figure 16: The Loan form

The following notes highlight some features of the *Loan* form:

- The scrollable list of registered borrowers at the top of the form reflects the information in the *Reader* table, ordered first by last name and then by first name.
- The number of entries shown in the list of borrowers can be reduced where necessary to make it more manageable. If a user inputs the letter “M” in the *Filter (Last Name)* field at the right of the list of borrowers and then clicks the **OK** button, only people whose last name begins with “M” will be included. Similarly, if the user inputs the letters “Mc” and then clicks the **OK** button, only people whose last name begins with “Mc” will be shown.
- New borrower information can be input directly into the fields of the list of borrowers. Note however that a new library user registered in this way cannot borrow media items – further action would be required by the library administrator to clear a lock flag in the *Reader* table.
- Details of items to be borrowed may be entered in the middle area of the form. After selecting the appropriate entry in the list of borrowers, the form updates to display the string “Loan for Reader” followed by the name of the selected borrower, with both written in emphasized text. If a previously borrowed item is overdue and must be returned before further loans are allowed, the middle area is locked (no input possible) and the string “Loan temporarily locked!” is appended after the borrower’s name. Items already on loan are shown in the lower area of the form.
- The *Loan Date* is automatically set to the current date. The media items that are available for loan are listed in the drop-down menu at the left of the **Refresh** button, listed in *ID* order (*ID* is a field in the *Media* table). Items that are already on loan to any reader are not available for selection in this menu.
- Media items selected for loan are added to the *Current Loan* area by clicking the **Refresh** button.
- In the lower section of the form (labeled *Return*) it is not possible to delete a data row. Only the fields *Return Date* and *Extension* can be edited. If a borrower was previously locked and has subsequently returned the overdue item(s), the middle area can be unlocked by clicking the **Refresh** button.

All functions of the *Loan* form in this sample database are performed without using macros.

Data input directly into a table – basics for data entry

Tables are database elements that contain the data in a database, organized in rows and columns forming a grid. An important part of database design is to define the links between the various tables in the database. A relationship between two tables is created using fields that are common to both tables.

To view the relationships defined for the sample *Media_without_Macros.odb* database, select the **Tools > Relationships** option on the Menu bar. Base displays the Relation Design window, which shows the relationships graphically with lines between the tables connecting a field from each. At first sight this diagram can be daunting and so it is advisable to focus on just a part of it.

The *Loan* table is directly related to the *Media* and *Reader* tables, as shown in Figure 17. In both cases, the relationship is one-to-many (or many-to-one), as indicated by the “1” and “n” at each end of the joining line. This reflects that one reader may borrow multiple media items at the same time, and that one media item may be loaned out multiple times.

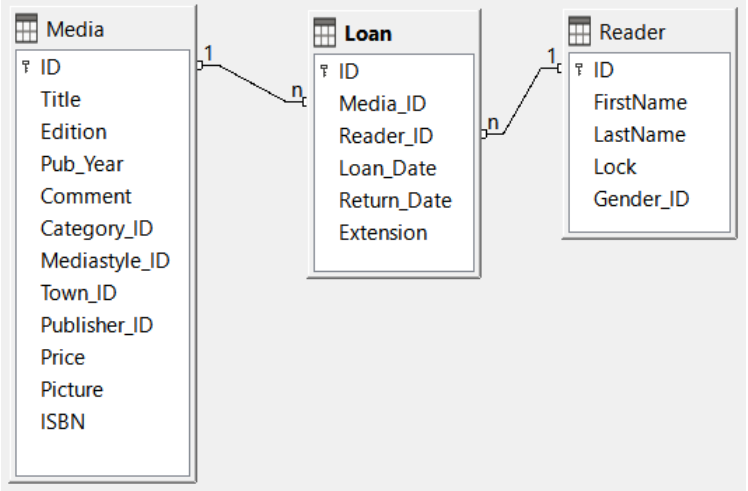


Figure 17: Relationships between the Loan table and the Media and Reader tables

Note

The *Loan*, *Media*, and *Reader* tables all have relationships with other tables but, for clarity, these are intentionally omitted from Figure 17.

When a book is borrowed, instead of its title being saved in the *Loan* table, only one number is saved in the *Media_ID* field. The *ID* field of the *Media* table stores the unique identifier for each record of this table. This field is a key field of the *Media* table: the primary key.

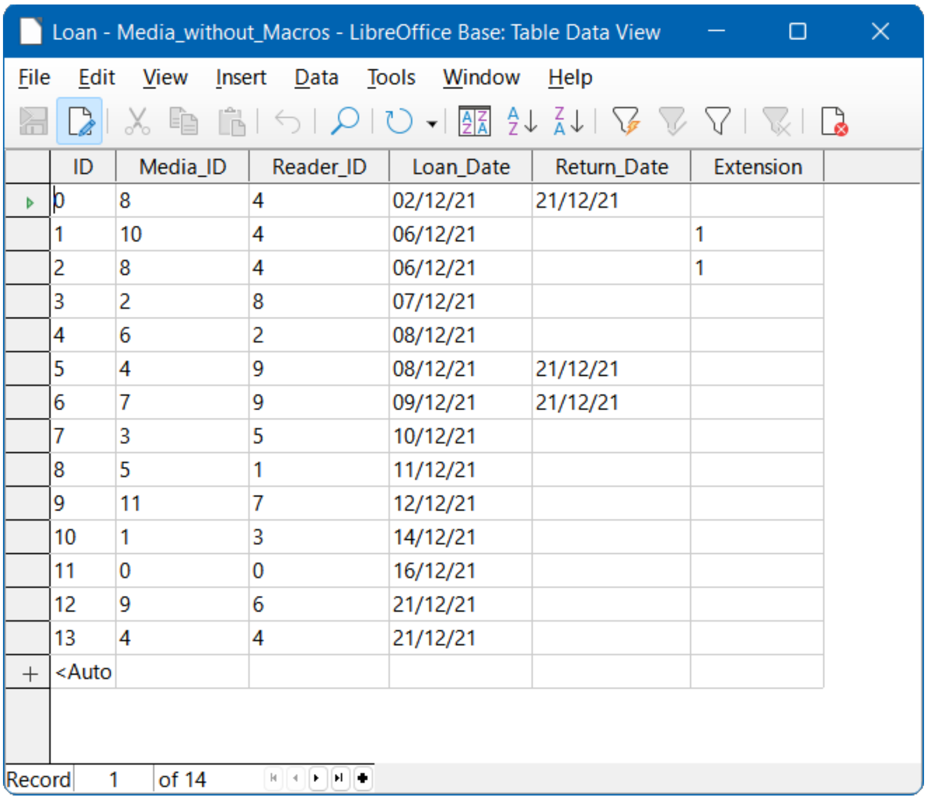
Note

The primary key uniquely determines the values for each field in each record of a table. So, when an item is borrowed, the number entered into the *Media_ID* field of the *Loan* table matches the number in the *ID* field of the *Media* table that identifies the record containing the information about the borrowed item.

The borrower’s name is not entered in the *Loan* table every time – this information is saved in the *Reader* table. It also has a primary key field which identifies each person who borrows an item. The value of this field can then be entered in the *Loan* table with the *Reader_ID* field identifying the specific person.

The relationships between the tables have the advantage that the desk work using the appropriate form is greatly reduced. Instead of having to enter the media title and first and last names without any errors, these can be entered by selecting the correct numbers for the *Media_ID* and *Reader_ID* fields, which allows the selection of the correct media items and first and last names. Finally, the same media item can be borrowed again later and the same reader can borrow several more media items at any loan event.

Figure 18 shows the *Loan* table of the *Media_without_Macros.odb* database. To open a table click **Tables** in the *Database* area of the main Base window, right-click the name of the table in the *Tables* area, and select **Open** in the context menu.


Loan - Media_without_Macros - LibreOffice Base: Table Data View
File
Edit
View
Insert
Data
Tools
Window
Help
Record
1
of 14

ID	Media_ID	Reader_ID	Loan_Date	Return_Date	Extension
0	8	4	02/12/21	21/12/21	
1	10	4	06/12/21		1
2	8	4	06/12/21		1
3	2	8	07/12/21		
4	6	2	08/12/21		
5	4	9	08/12/21	21/12/21	
6	7	9	09/12/21	21/12/21	
7	3	5	10/12/21		
8	5	1	11/12/21		
9	11	7	12/12/21		
10	1	3	14/12/21		
11	0	0	16/12/21		
12	9	6	21/12/21		
13	4	4	21/12/21		
<Auto					

Figure 18: Loan table data structure

The same data can be directly input in the rows and columns of the *Loan* table as was previously entered using the form. The relationships between this table and other tables of the database are used in the form.

- The most important field is the primary key (*ID*). This is generated automatically and shows the indispensable, unique content for most databases. For more on this topic, see the section “Relationships between tables” in Chapter 3, Tables.
- The second field, *Media_ID*, stores values of the primary key of the *Media* table. It refers to the number in the corresponding field, *ID*, in the *Media* table. Such a reference to a primary key is called a foreign key. In the form, the title and the author will be displayed instead of the foreign key in a list box. The list box transmits the value in the background to the foreign key of the table.
- The third field, *Reader_ID*, stores the primary key values of the *Reader* table. In this example, this key is only a number that refers to the reader who borrows media items. In the form, the last and first names of the reader are shown. As seen in the table, the reader with the primary key number '4' has borrowed multiple media items. The table can save the unique primary key of the *Reader* table as a foreign key *Reader_ID* many times. But in no case may a reader, who is listed in the foreign key of the *Loan* table, be deleted in the *Reader* table. Otherwise it would no longer be comprehensible as to who is now borrowing media. The database makes the default settings so that a deletion is impossible. The technical term for this is the requirement of *referential integrity*.
- The loan date is stored in the fourth field (*Loan_Date*). If this date is present and does not correspond to the current date, the corresponding data set for the reader is shown in the bottom table of the form under the **Return** button.
- The fourth field, *Return_Date*, stores the date on which a media item was returned from a loan period.

- The *Extension* field contains information about extensions of the loan for an item. The meaning of the values 1, 2... is explained later. The database contains a *Preferences* table that defines the number of days per extension and other associated parameters.

The input of this data permits the management of a simple media library.

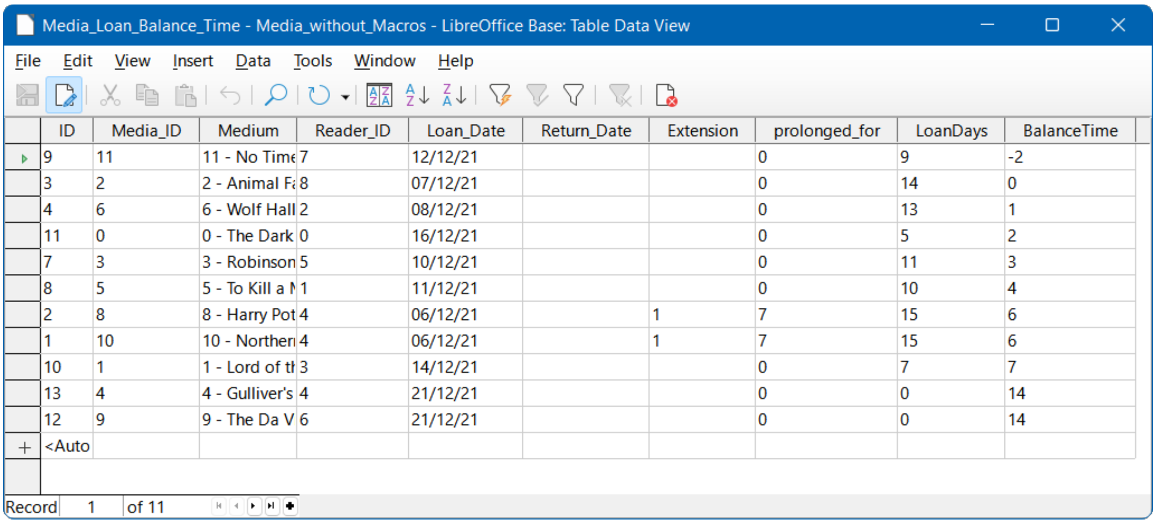
Queries – getting information from data in tables

A query is a request to retrieve or manipulate data from a database, and that data may come from tables, views, or other queries. Queries bring together content from multiple tables in an overview format. Base stores queries using SQL, which is a standard language for accessing and manipulating databases.

Note

Queries are not the same as tables or views, even if sometimes they appear to be similar.

To open a query click **Queries** in the *Database* area of the main Base window, right-click the name of the query in the *Queries* area, and select **Open** in the context menu.


Media_Loan_Balance_Time - Media_without_Macros - LibreOffice Base: Table Data View
File
Edit
View
Insert
Data
Tools
Window
Help

ID	Media_ID	Medium	Reader_ID	Loan_Date	Return_Date	Extension	prolonged_for	LoanDays	BalanceTime
9	11	11 - No Time	7	12/12/21			0	9	-2
3	2	2 - Animal Fa	8	07/12/21			0	14	0
4	6	6 - Wolf Hall	2	08/12/21			0	13	1
11	0	0 - The Dark	0	16/12/21			0	5	2
7	3	3 - Robinson	5	10/12/21			0	11	3
8	5	5 - To Kill a M	1	11/12/21			0	10	4
2	8	8 - Harry Pot	4	06/12/21		1	7	15	6
1	10	10 - Northern	4	06/12/21		1	7	15	6
10	1	1 - Lord of th	3	14/12/21			0	7	7
13	4	4 - Gulliver's	4	21/12/21			0	0	14
12	9	9 - The Da V	6	21/12/21			0	0	14
<Auto									

Record 1 of 11

Figure 19: Media_Loan_Balance_Time query

The *Media_Loan_Balance_Time* query shown in Figure 19 lists all media items that are currently out on loan. It calculates how long each item has been on loan and the balance of the loan period. Based on the *Media_ID* foreign key field, the query merges the media item’s primary key, title, and author into a single text string displayed in the *Medium* field. This field is used in the *Loan* form. Combined fields in the query also serve as connecting fields from the *Loan* form to the *Loan* table, namely through the *Media_ID* and *Reader_ID* fields.

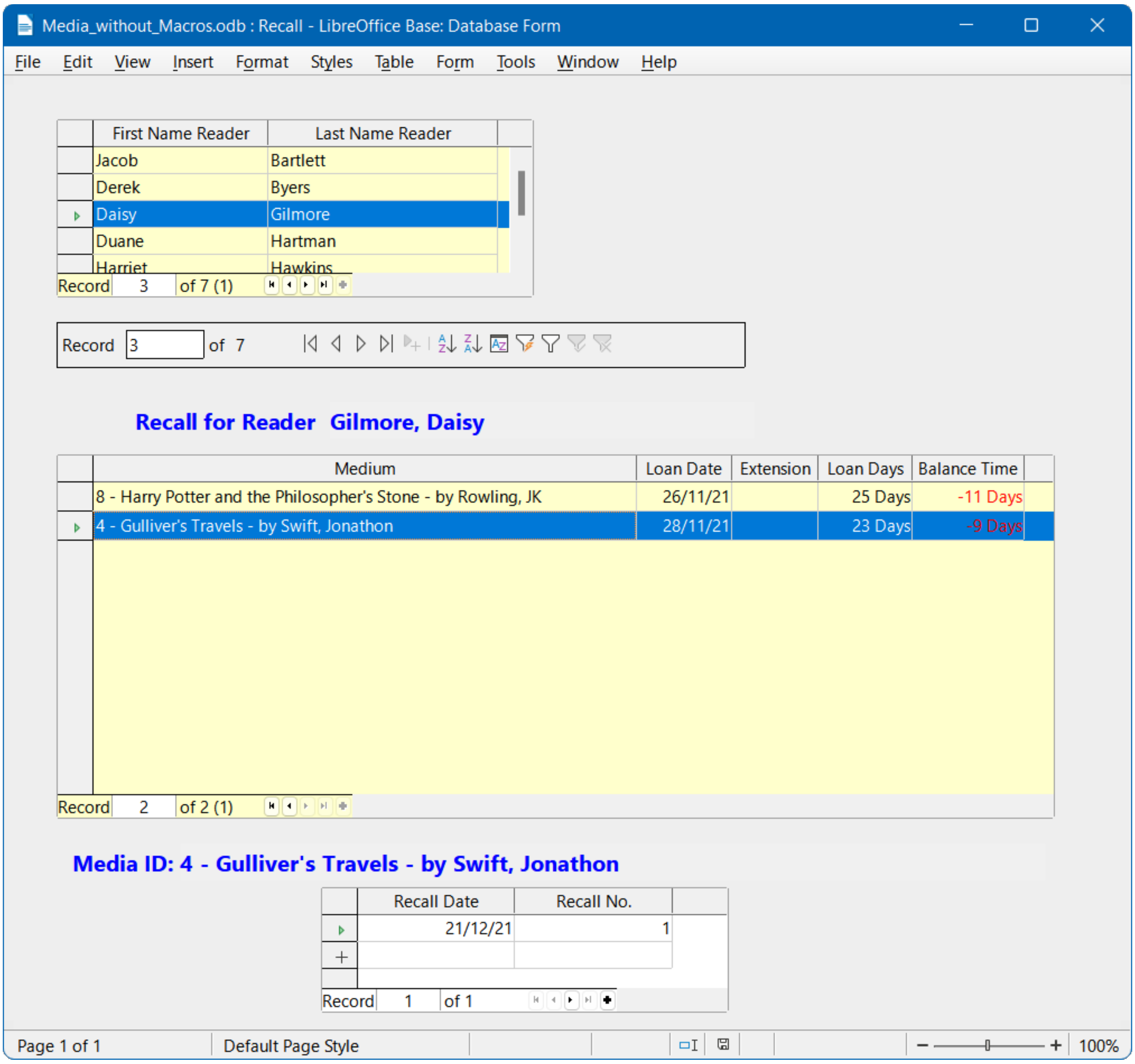
- All media items are listed for which the return date is not entered in the *Loan* table. The *Media_ID* and *Medium* query fields provide extra information to increase usability.
- A reference to the borrower is established by including the primary key of the *Reader* table (*Reader_ID*).
- The time difference in days is specified as *LoanDays*, calculated between the date of loan (*Loan_Date*) and the current date (Figure 19 shows a query that was run on 2021-12-21).
- The number of *LoanDays* is subtracted from the loan time to give the remaining number of days in the loan period. The loan time can vary with different media types and this is controlled by fields in the *Mediastyle* table.

- A value of 1 for the *Extension* field corresponds to an extension of the loan period of 7 days. In the dataset shown in Figure 19, the lines with *ID* set to 1 and 2 each have one extension of 7 days. The library administrator can grant multiple extensions and so values in this column are not limited to 1.

Reports – presentation of data

Reports enable you to retrieve data from the database and output the results in a helpful, user-friendly format. For example, your address book database may be full of details such as names, addresses, telephone numbers, dates of birth, and so on, but you might want to run a report to present only the addresses formatted for printing on labels.

Forms are not usually suitable to be used instead of reports. For example, consider the *Recall* form displayed in Figure 20, which is clearly not suitable for generating a cleanly formatted letter asking a borrower to return overdue media items.



Media_without_Macros.odb : Recall - LibreOffice Base: Database Form
File Edit View Insert Format Styles Table Form Tools Window Help

First Name Reader	Last Name Reader
Jacob	Bartlett
Derek	Byers
Daisy	Gilmore
Duane	Hartman
Harriet	Hawkins

Record 3 of 7 (1)

Record 3 of 7

Recall for Reader Gilmore, Daisy

Medium	Loan Date	Extension	Loan Days	Balance Time
8 - Harry Potter and the Philosopher's Stone - by Rowling, JK	26/11/21		25 Days	-11 Days
4 - Gulliver's Travels - by Swift, Jonathon	28/11/21		23 Days	-9 Days

Record 2 of 2 (1)

Media ID: 4 - Gulliver's Travels - by Swift, Jonathon

Recall Date	Recall No.
21/12/21	1

Record 1 of 1

Figure 20: Recall form

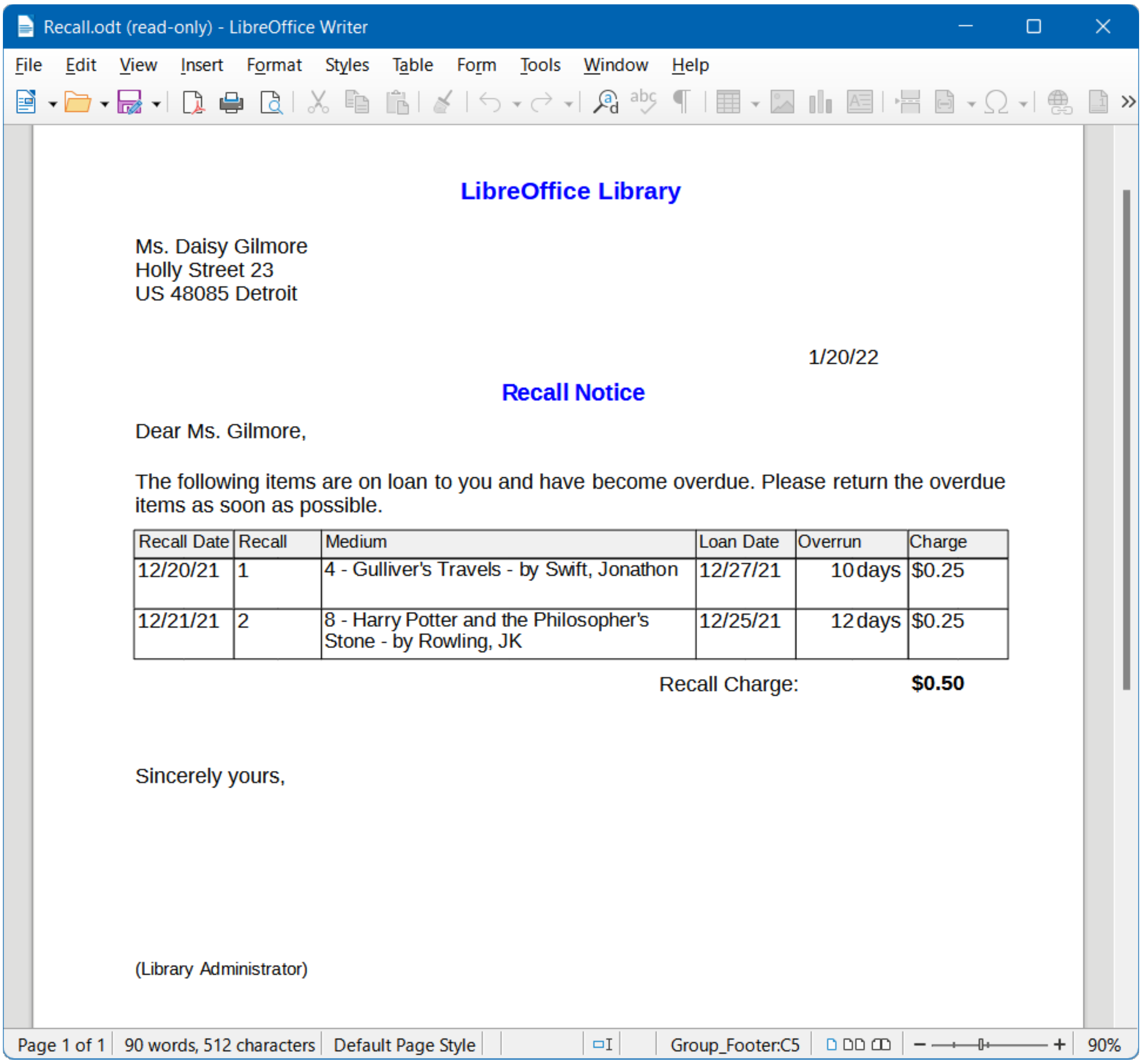
Before a report formatted as a recall notice can be printed, the recall information must be entered into the *Recall* form. The table in the form shows all persons who have borrowed items with a negative remaining loan time.

The recall date and recall notice number are entered for each media item to be recalled. The recall date defaults to the current date. The recall number is an integer incremented by 1 with each successive recall notice for a particular lender/media item.

This form, in the *Media_without_Macros.odb* database example, requires user input to create recall notices. In the macro version (*Media_with_Macros.odb*), the date is automatically entered and the recall notice printed.

The recall notice (Figure 21) is generated by means of a query from the previously input data. The user of the database needs only to select the *Recall* report and a recall letter can be printed out and sent to all persons who have a recall entry made in the form on the previous page.

To open a report click **Reports** in the *Database* area of the main Base window, right-click the name of the report in the *Reports* area, and select **Open** in the context menu.


Recall.odt (read-only) - LibreOffice Writer
File Edit View Insert Format Styles Table Form Tools Window Help

LibreOffice Library

Ms. Daisy Gilmore
Holly Street 23
US 48085 Detroit

1/20/22

Recall Notice

Dear Ms. Gilmore,

The following items are on loan to you and have become overdue. Please return the overdue items as soon as possible.

Recall Date	Recall	Medium	Loan Date	Overrun	Charge
12/20/21	1	4 - Gulliver's Travels - by Swift, Jonathon	12/27/21	10 days	$0.25
12/21/21	2	8 - Harry Potter and the Philosopher's Stone - by Rowling, JK	12/25/21	12 days	$0.25

Recall Charge: **$0.50**

Sincerely yours,

(Library Administrator)

Figure 21: Sample recall notice

In such a report there may be multiple entries (overdue items) for a particular person. If the table containing the items for this person exceeds the space available on a page, it is extended to include an additional page.

Such a report is more powerful than a mail merge letter produced with LibreOffice Writer. It automatically gathers together the data sets for printing and arranges the necessary accompanying text accordingly.

A similar letter to that shown in the above figure can be otherwise only implemented with macros, as described in the “Creating a report” section on page 66.

Safe handling of a Base file

Tables, queries, forms, and reports of the internal database, HSQLDB, are stored in a Base file. Because the database file is written into memory, the multiple objects in it require you to deal carefully with it. Bug reports make it clear that a database file requires just a bit more careful treatment than, for example, a text file that is written in Writer.

The following instructions should therefore be taken into consideration when dealing with a Base file:

- An open database file should not be saved with a different name by using **File > Save As** or an equivalent interaction. When there is no other choice, the tables, queries, forms, and reports should first be closed. It is better to close the database file and create a copy of the file.
- The Report Builder in Base was originally an add-on and although it no longer appears to be a separate extension, it operates largely independently of the database file. Renaming the database file removes the Report Builder of its foundation.
- Because a table, query, form, or report is saved, it does not follow that the entire database file has been saved. This saving must be done separately. When an object (table, query, form, report) is saved, the information is written to the database file in memory; when the database file is saved, everything contained in the database file in memory is written to the *.odb file.
- This memory behavior is especially true for working with the Report Builder. The preparation of a report is still the most unstable component inside the Base file. Therefore, after every step, the report and the *.odb file should both be saved. Once the report is created, it functions by itself without any particular problems.
- Once a *.odb file is finished, data added to the database is only written in the database file in memory, but not in the *.odb file. Only when you close the *.odb file do you save the data to it. The content of the HSQLDB database will be written back into the file. A crash at this point can result in data loss. Therefore, a strategy should be developed so that backup copies are made on time. Chapter 9, Macros, includes macros to automate some of these activities.
- A significantly higher level of security can be achieved using external server databases such as MySQL, MariaDB, or PostgreSQL. For this, Base can then serve as a front end, implementing the queries, forms, and reports for the database.

A simple database – a detailed example

Overview

In the remainder of this chapter are instructions to build a database that is intended to help with organizing a sports competition into different disciplines. The functionality of this database is intentionally limited because the sole objective is to familiarize the reader with aspects of the user interface and functionality of Base. The database will be saved as *Example_Sport.odb*.

Creating the database

The creation of a database and operation of the Database Wizard are discussed in detail in Chapter 2, Creating a Database. For now, the following steps are followed to create the *Example_Sport.odb* database:

1) Launch the Database Wizard using one of the following methods:

   a) From the LibreOffice Start Center, click on the **Base Database** option in the *Create* area.

   b) In any LibreOffice component, select **File > New > Database** on the Menu bar or click the arrow at the right of the **New** icon on the Standard toolbar and select the **Database** option from the drop-down menu.

   c) In Base, click on the **New** icon in the Standard toolbar or press *Ctrl+N*.

Tip

Like many other software applications, LibreOffice provides keyboard shortcuts for certain interactions, such as *Ctrl+N* in this case. For some users, these can improve efficiency, precision and accessibility. The majority of keyboard shortcuts available within Base are common across multiple components of LibreOffice. The best way to learn about the available shortcuts is while using menus and toolbars, when you can observe the equivalent key sequence that is stated within the legend written on the menu option or within the tooltip that appears when the pointer is over a toolbar icon. You can also view a list of keyboard shortcuts on the *Keyboard* tab of the **Tools > Customize** dialog (select the **LibreOffice** radio button).

To avoid unnecessary repetition, the remainder of this book will normally avoid identifying the key sequences associated with menu and toolbar options.

2) At step 1 of the Database Wizard, select the **Create a new database** option and select **HSQLDB Embedded** in the *Embedded database* menu.
3) Click on the **Next >** button.
4) At step 2 of the Database Wizard, select the **No, do not register the database** option, select the **Open the database for editing** option, and make sure that the **Create tables using the table wizard** option is not selected.
5) Click the **Finish** button. Base displays the Save As dialog.
6) Navigate to the folder where the new database is to be stored.
7) Enter the text "Example_Sport.odb" in the *File name* field.
8) Click the **Save** button to save the new database and close the Save As dialog.

Creating tables

Once the database has been saved, the main Base window is displayed, with *Tables* selected in the *Database* area on the left side of the window (Figure 22). The tables are the central storage for data; without tables, there is no database.

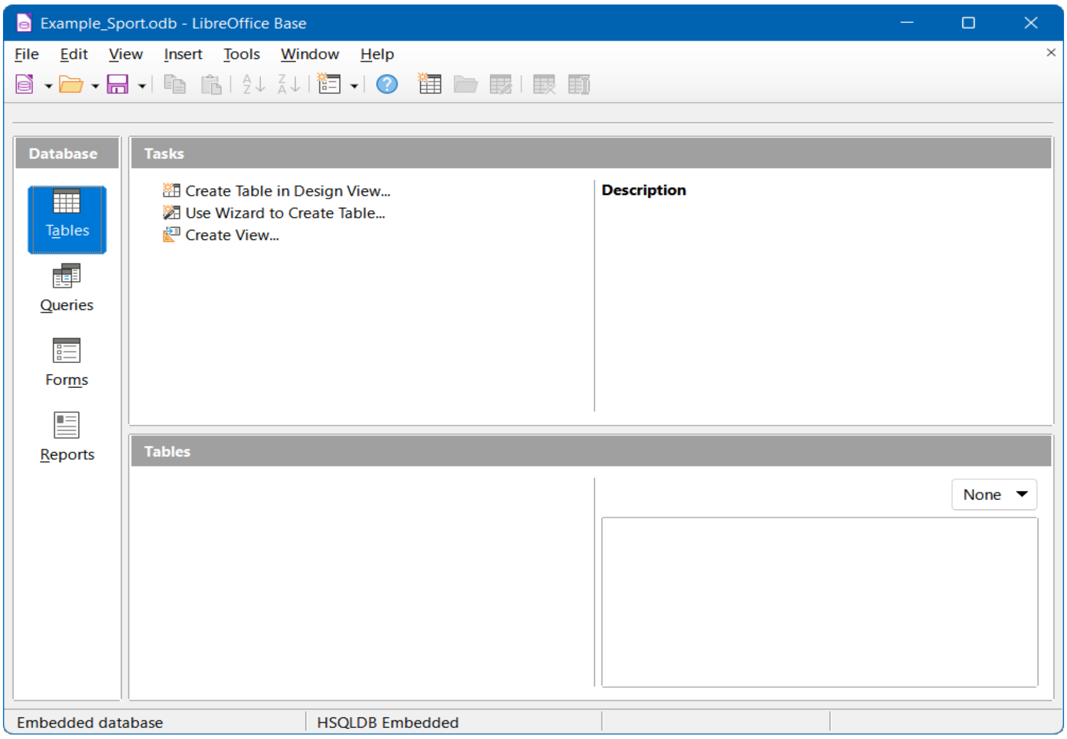
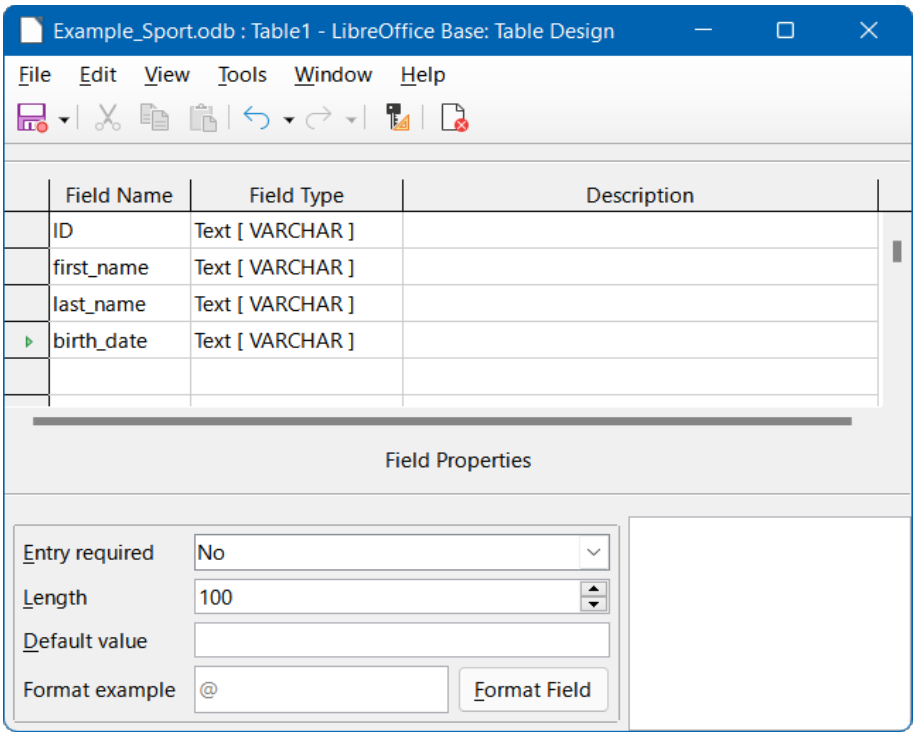

Example_Sport.odb - LibreOffice Base
File Edit View Insert Tools Window Help
Database
Tables
Queries
Forms
Reports
Tasks
Create Table in Design View...
Use Wizard to Create Table...
Create View...
Description
Tables
None
Embedded database
HSQLDB Embedded

Figure 22: Main Base window with Tables work area selected

The *Example_Sport.odb* database will comprise three tables; one containing details of the entrants, one containing details of the available sports, and one to define which sports each entrant wishes to participate in.

Creating a table to store the details for competition entrants

Click **Create Table in Design View** to open the Table Design window shown in Figure 23.


Example_Sport.odb : Table1 - LibreOffice Base: Table Design
File Edit View Tools Window Help
Field Properties
Format Field

Field Name	Field Type	Description
ID	Text [VARCHAR]	
first_name	Text [VARCHAR]	
last_name	Text [VARCHAR]	
birth_date	Text [VARCHAR]	

Entry required	No
Length	100
Default value	
Format example	@

Figure 23: Table Design window

1) The first step is to enter the field names for the table. The table will eventually include both male and female entrants.

   a) Enter the first field name as “ID”. The corresponding field type is set by default to *Text[VARCHAR]*. This field will later take a value that is unique for each record. A unique key field is necessary for the embedded database, otherwise no records can be entered in the table. This key field is called the *primary key* in databases.

Note

Another field could be used for this property, with the *ID* field omitted. However, if for example *last_name* was used, two people with the same last name could not be saved. It is possible to declare two fields together as one composite key but, in this table, there is no guarantee that the combination of fields would be unique. Therefore, in this case, the simple design proposed is preferred.

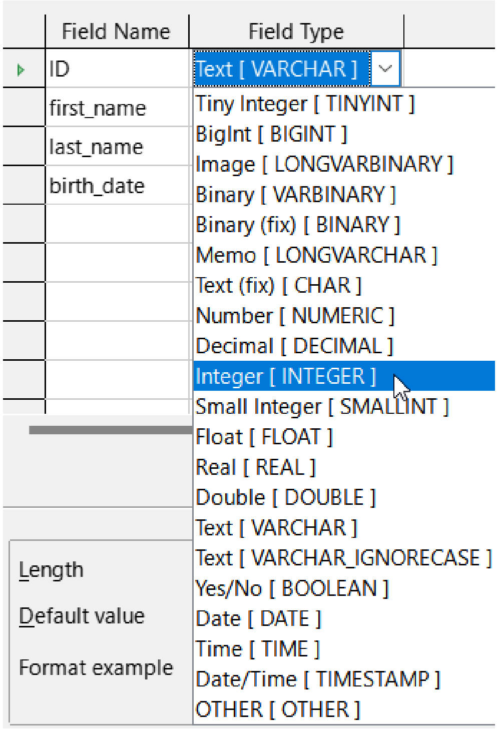
   b) Enter the second field name as “first_name”.

   c) Enter the third field name as “last_name”.

   d) Enter the fourth field name as “birth_date”.

2) The next step is to select the required field types for the four named fields.

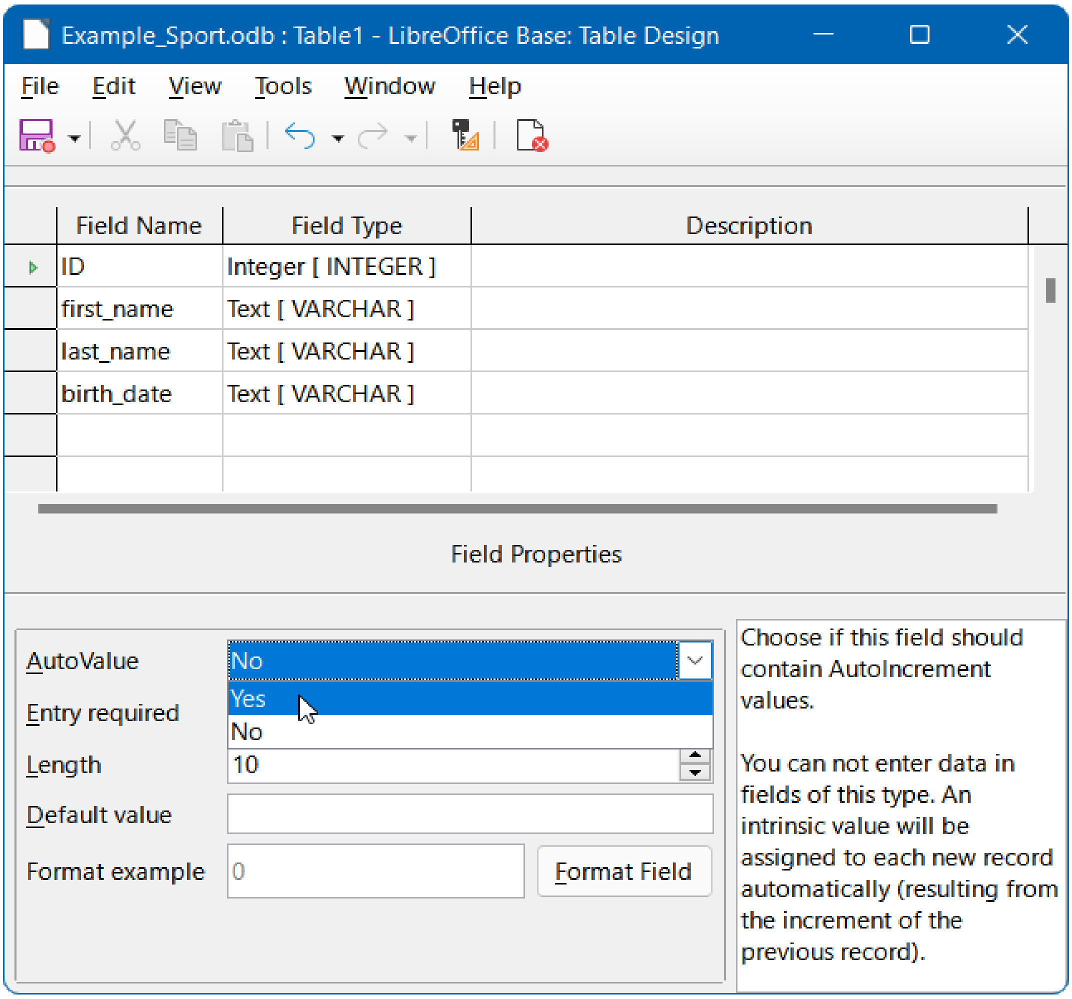
   a) Set the *ID* field to the field type *Integer[INTEGER]* (Figure 24). This field type has the advantage that it can be automatically set by the embedded HSQLDB engine to the next highest integer.



Field Name	Field Type
ID	Text [VARCHAR]
first_name	
last_name	
birth_date	

Figure 24: Set ID field type to Integer[INTEGER]

   b) Set the *AutoValue* property to **Yes** for the *ID* field, activating the automatic setting of ascending numerical values (Figure 25).



Field Name	Field Type	Description
ID	Integer [INTEGER]	
first_name	Text [VARCHAR]	
last_name	Text [VARCHAR]	
birth_date	Text [VARCHAR]	

Field Properties

AutoValue: No (Yes / No)
Entry required
Length: 10
Default value
Format example: 0
Format Field

Choose if this field should contain AutoIncrement values.

You can not enter data in fields of this type. An intrinsic value will be assigned to each new record automatically (resulting from the increment of the previous record).

Figure 25: Set AutoValue property for ID field

c) After setting *AutoValue*, a key icon should appear on the row header when leaving the *Field Type* selection. This indicates that this field is the primary key of the table. If *AutoValue* is not set, the primary key can also be selected in the context menu (Figure 26).

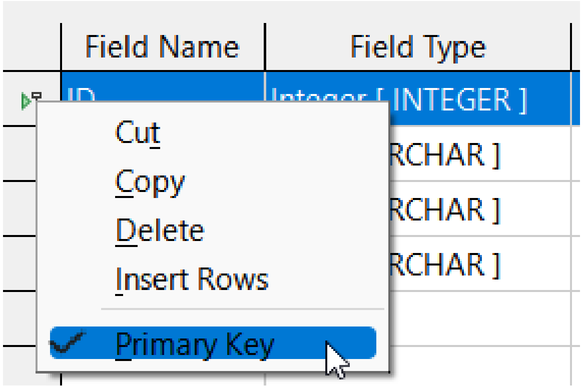
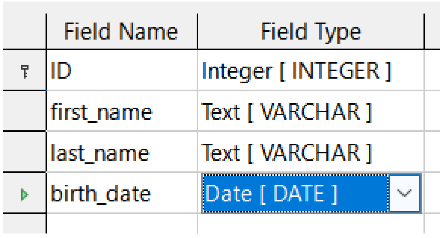


Figure 26 Set primary key via context menu

d) Select the field type for *birth_date* to *Date[DATE]* (Figure 27). This ensures that only valid date entries are added. It is also used for sorting dates, or, for example, calculating age.



Field Name	Field Type
ID	Integer [INTEGER]
first_name	Text [VARCHAR]
last_name	Text [VARCHAR]
birth_date	Date [DATE]

Figure 27: Set birth_date field to the Date[DATE] type

e) On the Table Design window, click the **Save** icon on the Standard toolbar. Enter the name “tblEntrants” on the Save As dialog and click the **OK** button. The table is now saved under the name *tblEntrants*, so that subsequently data can be entered. It is not necessary to input values in the *ID* field because this is done automatically when you save the record.

Tip

Any non-trivial use of Base, especially with macros, can lead to confusion between the names of tables, queries, forms, and reports. To ensure a clear distinction, it can be helpful to prefix each name with a short abbreviation, such as “tbl” (for tables), “qry” (for queries), “frm” (for forms), and “rpt” (for reports).

Most of the sample databases used in this book do not follow this convention because they were developed before the guidelines were proposed. However, it is good practice to adopt this or some other naming convention before embarking on the design of any significant database.

The above naming convention is followed in the *Example_Sport.odb* database described in this chapter.

Note

The database file is a zipped folder of individual files. Storing a single object such as the table is therefore not directly written to the database file itself. That’s why the **Save** button for the database file itself must be clicked even after the creation of tables, queries, forms, and reports.

Only when leaving the data row is the entered data saved automatically.

It is now possible to enter details of athletes in the *tblEntrants* table. However, we have (deliberately!) neglected the requirement to be able to distinguish between male and female competitors, so another field must be added to the table as follows:

1) Open the *tblEntrants* table for editing by clicking **Tables** in the *Database* area of the main Base window, right-clicking the name of the table in the *Tables* area, and selecting **Edit** in the context menu.

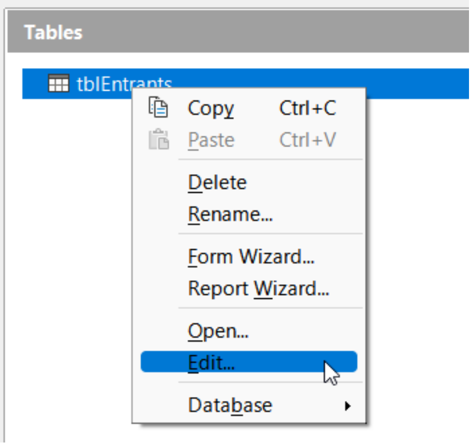
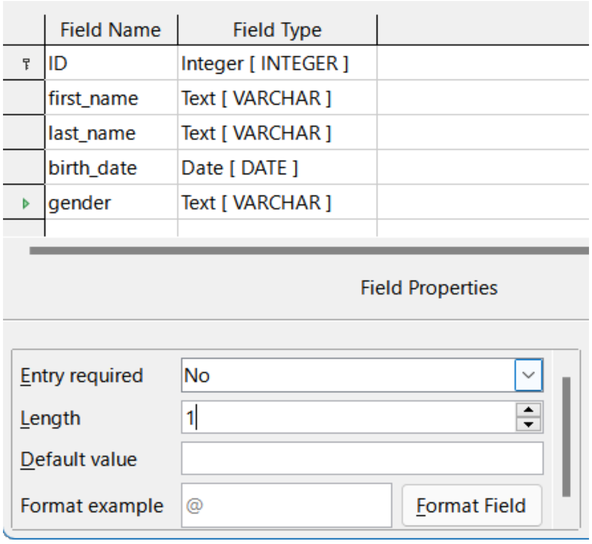


Figure 28: Select Edit in context menu

2) Add a new field *gender* to the table – new fields may be added only at the end of the table when using the Table Design window. Note however that it is also possible using SQL to add new fields to certain positions.

3) Leave the *Field Type* as *Text[VARCHAR]* but limit the length of the text in this field to one character by changing *gender*'s *Length* field property to 1 (Figure 29). One character is sufficient for 'm' and 'f' as input.



Field Name	Field Type
ID	Integer [INTEGER]
first_name	Text [VARCHAR]
last_name	Text [VARCHAR]
birth_date	Date [DATE]
gender	Text [VARCHAR]

Field Properties

Entry required: No

Length: 1

Default value:

Format example: @

Figure 29: Add gender field to tblEntrants table

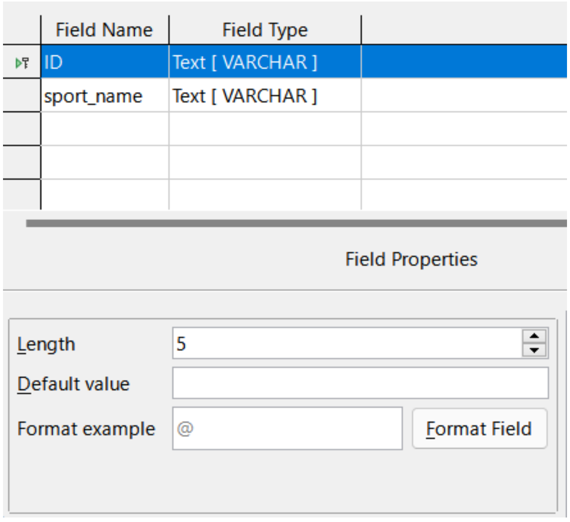
4) Save the updated *tblEntrants* table.

Creating a table to store the details of the sports to be played

To define the sports that are to be included in the competition, create a *tblSports* table, as follows:

1) Click **Create Table in Design View** in the *Tasks* area of the main Base window to open the Table Design window again.

2) Create two fields named *ID* and *sport_name*. Set *ID* to be the primary key, using the context menu accessed by right-clicking on the field's row header. Since there are not many different sports, the *AutoValue* option is not selected for the primary key. Instead, the field type is left as *Text[VARCHAR]*, but limited to five characters. The five characters are sufficient, in order to find a suitable but unique abbreviation for each sport (Figure 30).



Field Name	Field Type
ID	Text [VARCHAR]
sport_name	Text [VARCHAR]

Field Properties

Length: 5

Default value:

Format example: @

Figure 30: Create tblSports table

3) Save the table with the name *tblSports*.

Creating a table to link entrants to their sports

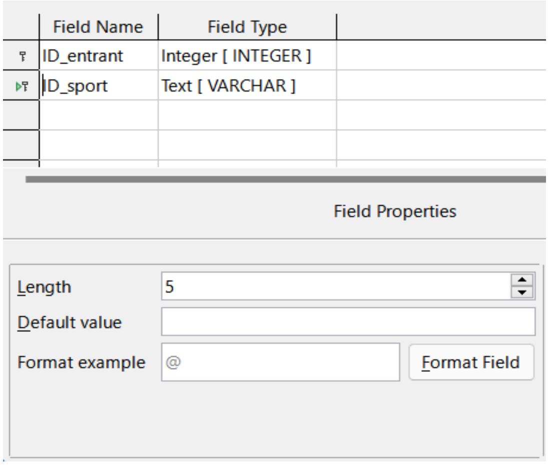
A further table is needed to link the *tblEntrants* and *tblSports* tables, so that each entrant can be registered in several sports and multiple entrants can be registered for any sport. This is achieved through a third table, in which the values of the two primary keys of the *tblEntrants* and *tblSports* tables are saved. Since only the combination of these fields will be saved together, these fields are combined to form a composite key for this table. Perform the following actions:

1) Click **Create Table in Design View** in the *Tasks* area of the main Base window to open the Table Design window again.

2) Create two fields named *ID_entrant* and *ID_sport*.

3) Set the field type for *ID_entrant* to *Integer[INTEGER]* to match the *ID* field in the *tblEntrants* table. Do not change the field type of the *ID_sport* field as it already matches the *ID* field in the *tblSports* table. However the *Length* property should be changed to 5 for the *ID_sport* field.

4) Click the row header for the *ID_entrant* field, then shift-click the row header for the *ID_sport* field; this selects both fields. Right-click either row header, then select **Primary Key** in the context menu to specify the composite key (Figure 31).



Field Name	Field Type
ID_entrant	Integer [INTEGER]
ID_sport	Text [VARCHAR]

Field Properties

Length: 5

Default value:

Format example: @

Figure 31: Create tblRel_entrant_sport table

5) Save the table with the name “tblRel_entrant_sport”.

Tip

The results of a competition could also be included in this table. However, if several competitions are held, a race date must be attached to the common primary key.

Defining relationships between tables

With the tables for the *Example_Sport.odb* database defined, to harness the real power of a relational database the relationships between those tables will be defined. All tables created so far are necessary for the relationship definition. The relationships allow the database to ensure that data is valid and non-repetitive, with no possibility of unexpected duplication. This can prevent a number for an entrant from appearing in the *tblRel_entrant_sport* table that is not listed in the *tblEntrants* table, for example. Carry out the following steps:

1) Select **Tools > Relationships** on the Menu bar.
2) Base opens the Relation Design window and the Add Tables dialog in readiness for you to start defining the relationships (Figure 32).

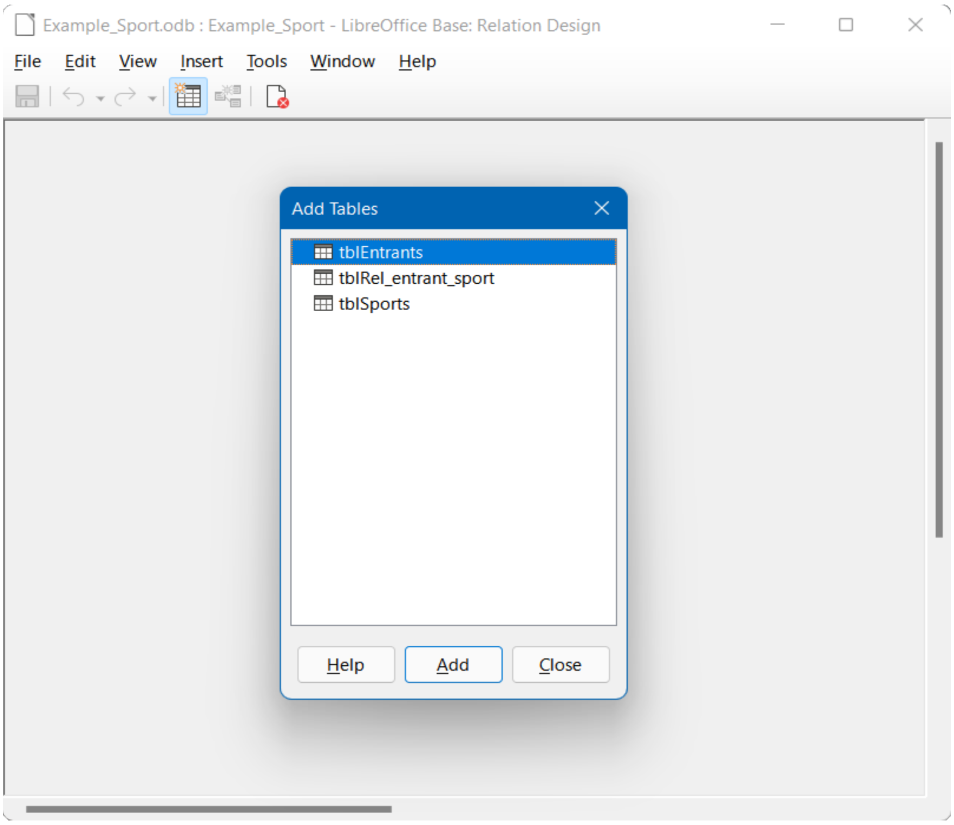

Example_Sport.odb : Example_Sport - LibreOffice Base: Relation Design
File Edit View Insert Tools Window Help
Add Tables
tblEntrants
tblRel_entrant_sport
tblSports
Help
Add
Close

Figure 32: Initial configuration of Relation Design window

3) Click the name of the *tblRel_entrant_sport* table and then click the **Add** button. Notice how a rectangle titled *tblRel_entrant_sport* appears in the main area of the window.
4) Click the name of the *tblSports* table and then click the **Add** button.
5) Click the name of the *tblEntrants* table and then click the **Add** button.
6) Click the **Close** button.
7) Base closes the Add Tables dialog, leaving the three tables represented by rectangles in the Relation Design window (Figure 33). All the fields are listed in each of the added tables. The primary key fields are marked with a key symbol.

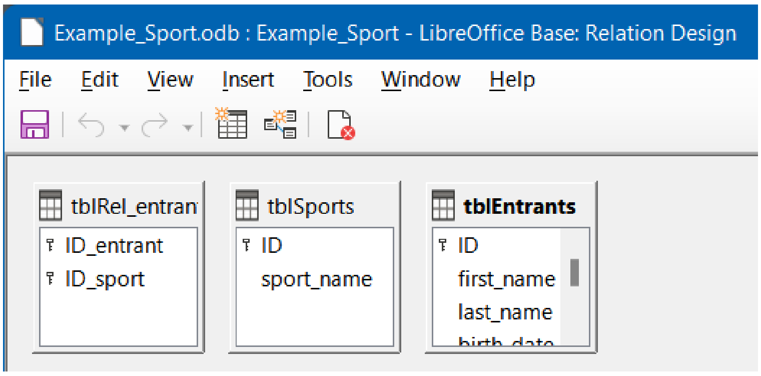

Example_Sport.odb : Example_Sport - LibreOffice Base: Relation Design
File Edit View Insert Tools Window Help
tblRel_entran
ID_entrant
ID_sport
tblSports
ID
sport_name
tblEntrants
ID
first_name
last_name
birth_date

Figure 33: Three tables added to Relation Design window

8) Move and resize the tables as required to improve the layout of the window (Figure 34).

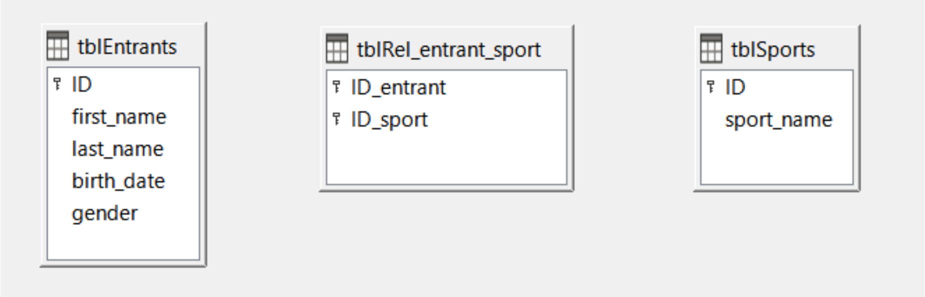


Figure 34: Tables moved and resized

9) Left-click the *ID* field in the *tblEntrants* table. Hold the mouse button down and move the pointer to the *ID_entrant* field in the *tblRel_entrant_sport* table. The cursor changes appearance to indicate a link (Figure 35). Release the mouse button.

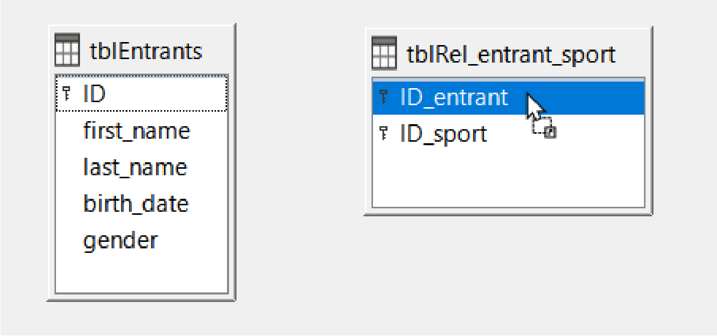
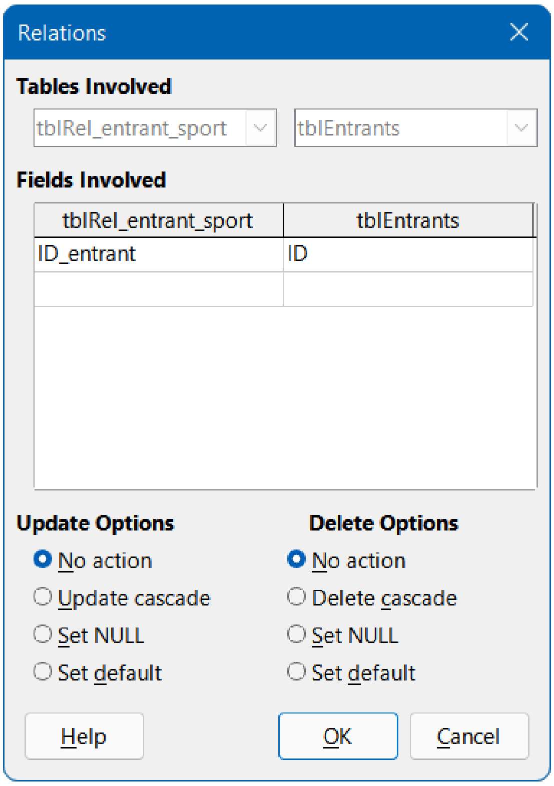


Figure 35: Creating a relationship

10) Base displays the Relations dialog to more fully define the nature of the relationship (Figure 36).



Tables Involved

tblRel_entrant_sport | tblEntrants

Fields Involved

tblRel_entrant_sport	tblEntrants
ID_entrant	ID

Update Options	**Delete Options**
● No action	● No action
○ Update cascade	○ Delete cascade
○ Set NULL	○ Set NULL
○ Set default	○ Set default

Figure 36: Relations dialog – tblRel_entrant_sport and tblEntrants

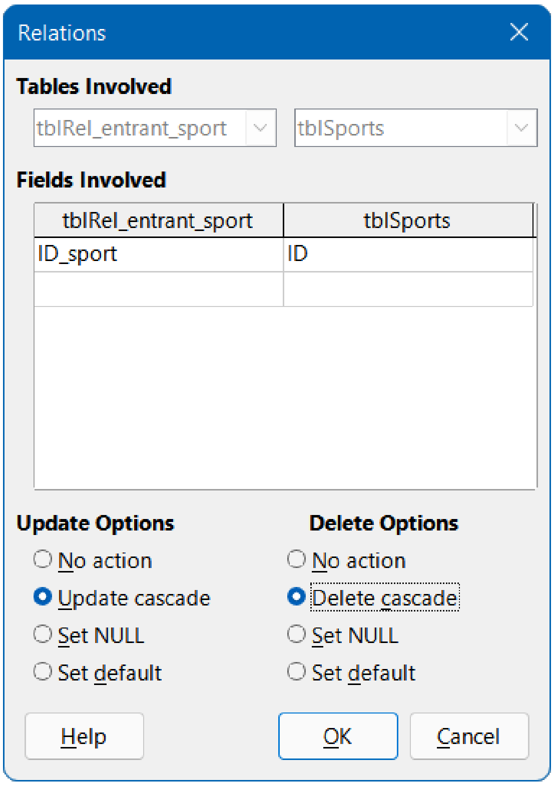
11) Make sure that the **No action** option is selected in the *Update Options* area (this is the default). The *ID* field for a record in the *tblEntrants* table is an automatically generated value.

12) Select the **Delete cascade** option in the *Delete Options* area. If this option is set on, when deleting an entry from the *tblEntrants* table, the database engine also deletes all records for that entrant from the *tblRel_entrant_sport* table, to maintain the referential integrity of the database without generating an error. With the option off, deleting an entrant could lead to orphaned records in the *tblRel_entrant_sport* table, which would be an error condition.

13) Click the **OK** button to close the Relations dialog. Base draws a line between the *ID* field of the *tblEntrants* table (labeled “1”) and the *ID_entrant* field in the *tblRel_entrant_sport* table (labelled “n”).

14) Left-click the *ID* field in the *tblSports* table. Hold the mouse button down and move the pointer to the *ID_sport* field in the *tblRel_entrant_sport* table. The cursor changes appearance to indicate a link. Release the mouse button.

15) Base displays the Relations dialog to more fully define the nature of the relationship.



Relations

Tables Involved

tblRel_entrant_sport | tblSports

Fields Involved

tblRel_entrant_sport	tblSports
ID_sport	ID

Update Options

○ No action
● Update cascade
○ Set NULL
○ Set default

Delete Options

○ No action
● Delete cascade
○ Set NULL
○ Set default

Figure 37: Relations dialog - tblRel_entrant_sport and tblSports

16) Select the **Update cascade** option in the *Update Options* area. If this option is set on and you change an *ID* field in the *tblSports* table to a different 5-character text string, the records in the *tblRel_entrant_sport* table that had *ID_sport* set to the *ID* that was being changed are also updated so that they continue to be linked to the correct sport. With the option off then changing the sport's *ID* field could lead to orphaned records in the *tblRel_entrant_sport* table, which would be an error condition.

17) Select the **Delete cascade** option in the *Delete Options* area. If this option is set on, when deleting a sport from the *tblSports* table, the database engine also deletes all records for that sport from the *tblRel_entrant_sport* table, to maintain the referential integrity of the database without generating an error. With the option off, deleting a sport could lead to orphaned records in the *tblRel_entrant_sport* table, which would be an error condition.

18) Click the **OK** button to close the Relations dialog. Base draws a line between the *ID* field of the *tblSports* table (labeled "1") and the *ID_sport* field in the *tblRel_entrant_sport* table (labelled "n").

19) Save the updated relationship design.

The required tables and fields are now completely connected (Figure 38). The relationships indicate that an entrant may appear repeatedly in the *tblRel_entrant_sport* table. A sport can also appear repeatedly in the same table. A given combination of starter and sport can appear in the table only once. From two *1:n* relationships, an *n:m* relationship now exists through the intervening *tblRel_entrant_sport* table.

Such a table design can be tiresome when the database is to be filled by typing content into tables. It requires all three tables to be opened when an entrant is to be assigned to a sport. The entrant's *ID* must be sought in the *tblEntrants* table and entered in the *ID_entrant* field in the *tblRel_entrant_sport* table. Similarly the sport's *ID* must be sought in the *tblSports* table and entered in the *tblRel_entrant_sport* table. This is overly complex and a form would solve the problem more elegantly.

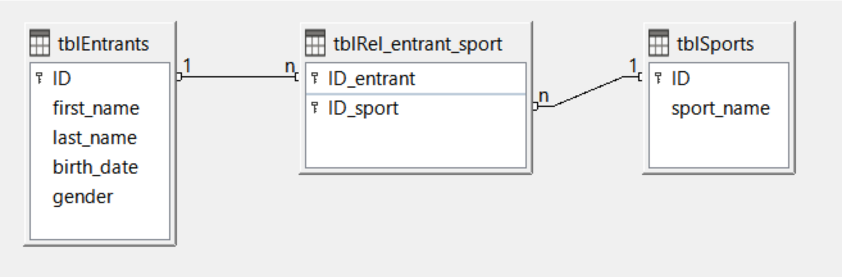


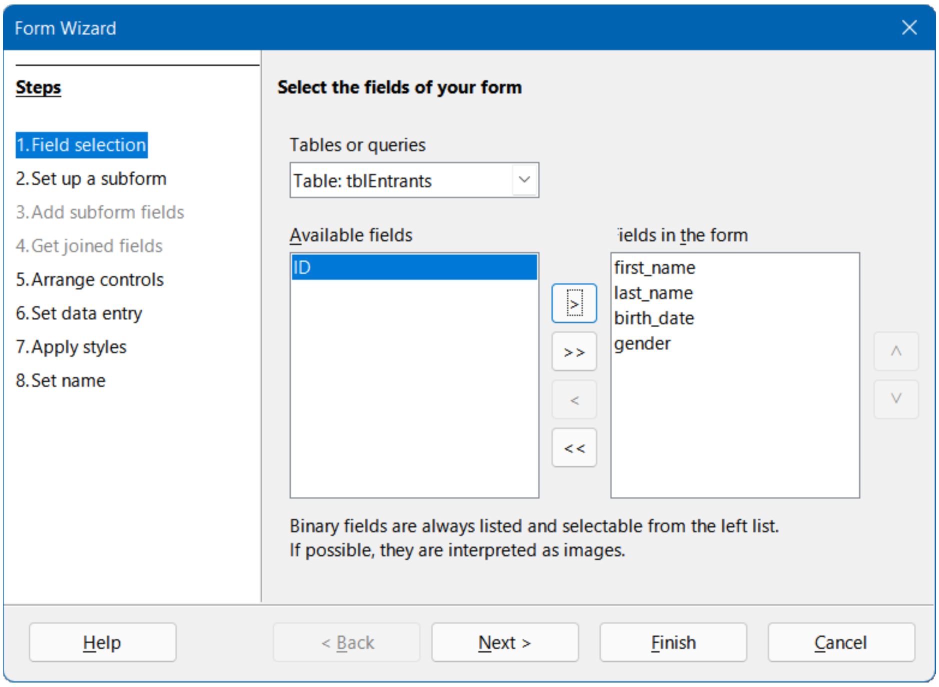
Figure 38: Relationships between three tables

Creating a data entry form

Forms can be created directly in the design view or by using a wizard. Even experienced users find that they can quickly use the wizard and then customize its output to produce a form that meets their needs. This approach can often save time.

To use the Form Wizard to create a form, carry out the following steps.

1) On the main Base window, click **Forms** in the *Database* area.
2) In the *Tasks* area, click **Use Wizard to Create Form**.
3) Base opens the Database Form window and the Form Wizard dialog in readiness for you to start creating the form. The Form Wizard opens at step 1 of 8. You can view components of the form building up on the underlying Database Form window as you complete the steps of the Form Wizard.
4) Select the entry for the *tblEntrants* table in the *Tables or queries* drop-down menu. The *tblEntrants* table data should be written in the main form. Data from the *tblSports* table is loaded directly with the few necessary sports and will be rarely updated.
5) Use the right arrow buttons to move all entries except *ID* in the *Available fields* area to the *Fields in the form* area (Figure 39). All fields except the primary key field (*ID*) are needed from the *tblEntrants* table. The primary key field is filled automatically with a unique value.



Select the fields of your form

Tables or queries: Table: tblEntrants

Available fields: ID

ields in the form: first_name, last_name, birth_date, gender

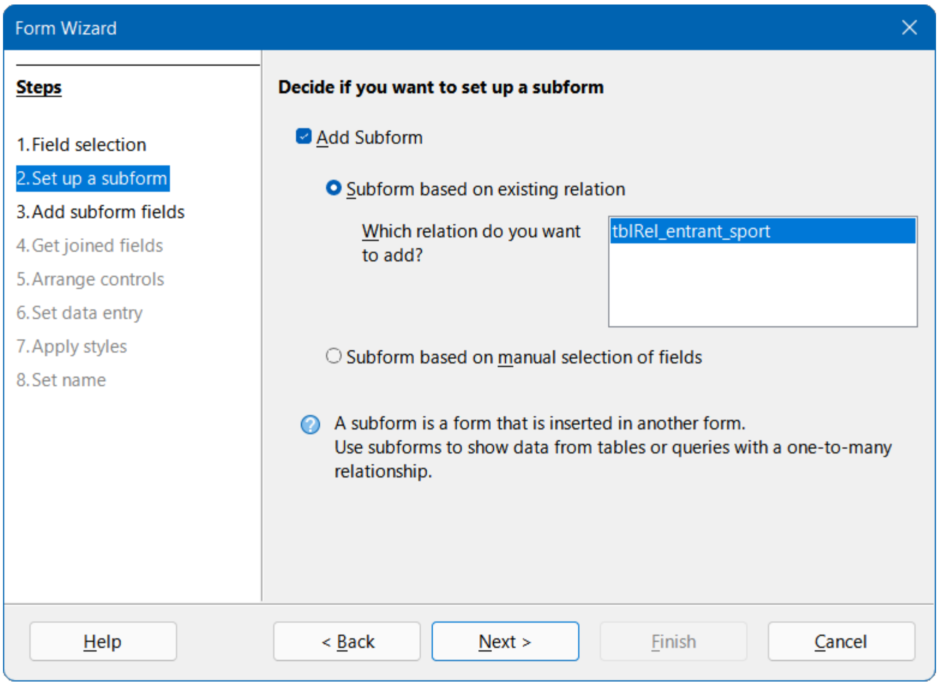
Binary fields are always listed and selectable from the left list.
If possible, they are interpreted as images.

Figure 39: Form Wizard step 1

6) Click the **Next >** button. Base displays step 2 of the Form Wizard.

7) Select **Add Subform** and **Subform based on existing relation**. A subform should be set up where a sport can be assigned to an entrant.

8) Select the *tblRel_entrant_sport* entry adjacent to the *Which relation do you want to add?* label (Figure 40). This confirms the previously defined relationship.



Form Wizard

Decide if you want to set up a subform

☑ Add Subform

● Subform based on existing relation

Which relation do you want to add? tblRel_entrant_sport

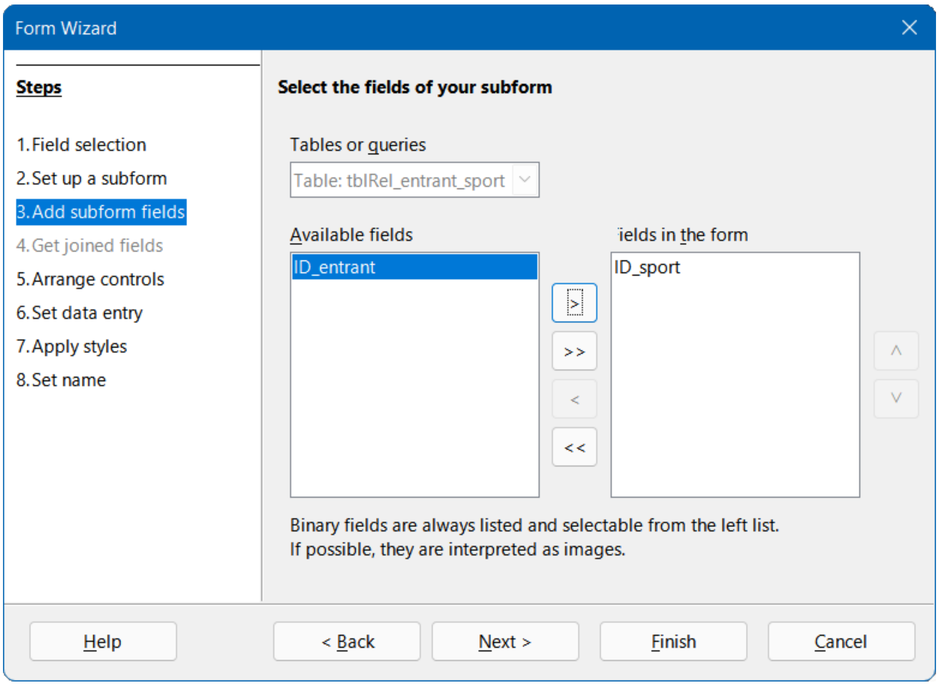
○ Subform based on manual selection of fields

A subform is a form that is inserted in another form.
Use subforms to show data from tables or queries with a one-to-many relationship.

Figure 40: Form Wizard step 2

9) Click the **Next >** button. Base displays step 3 of the Form Wizard.

10) Use the right arrow buttons to move the *ID_sport* field from the *Available fields* area to the *Fields in the form* area (Figure 41). The primary key in the *tblEntrants* table provides the value for the *ID_entrant* field for the current record by the connection of the main form to the subform.



Form Wizard

Select the fields of your subform

Tables or queries: Table: tblRel_entrant_sport

Available fields: ID_entrant

ields in the form: ID_sport

Binary fields are always listed and selectable from the left list.
If possible, they are interpreted as images.

Figure 41: Form Wizard step 3

11) Click the **Next >** button. Base displays step 5 of the Form Wizard. Step 4 (*Get joined fields*) is inactive because the linkage is already in place.

12) Click the second icon of the four under the label *Arrangement of the main form*. This arrangement is called *Columnar – Labels on Top*. How the items in the main form and subform are arranged is not of great importance for this example. However, the layout should always be clear and user-friendly to assist inexperienced users.

13) Click the third icon of the four under the label *Arrangement of the subform* (Figure 42). This arrangement is called *As Data Sheet*. The fields in the subform will later show all sports of the entrants, so the subform arrangement is best left at its default setting.

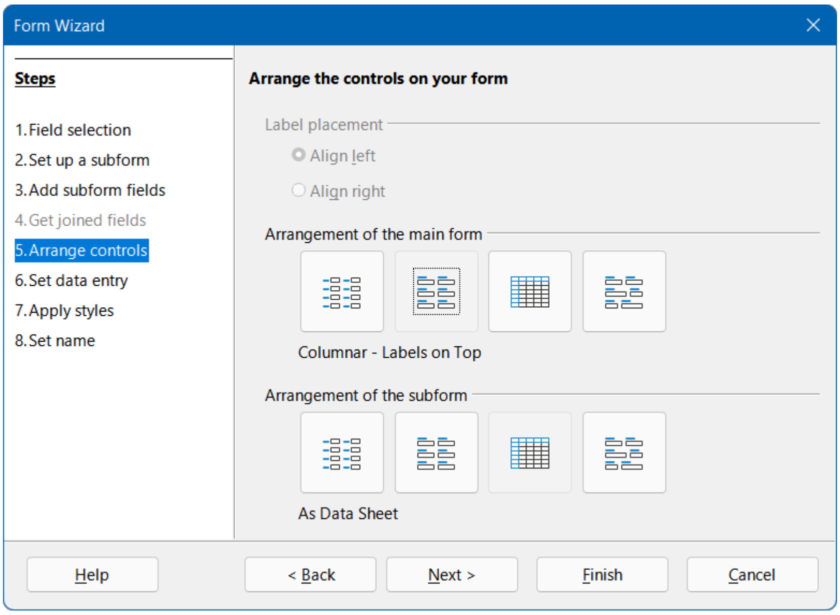
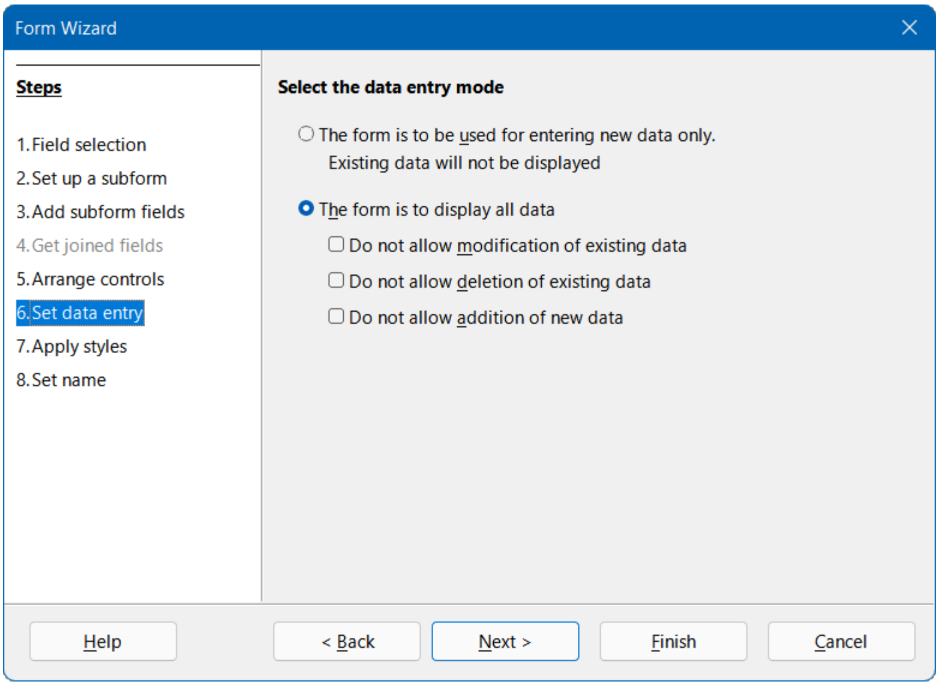


Figure 42: Form Wizard step 5

14) Click the **Next >** button. Base displays step 6 of the Form Wizard. Make sure that the default option **The form is to display all data** is selected (Figure 43). This means that new entries can be made and existing entries can be modified.



Select the data entry mode

○ The form is to be used for entering new data only. Existing data will not be displayed

● The form is to display all data

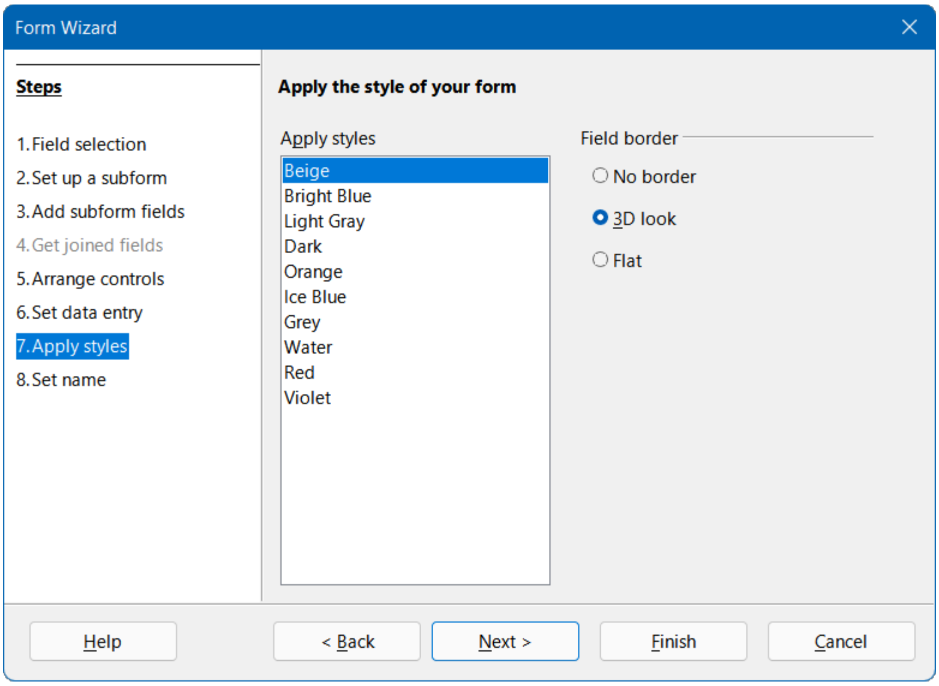
□ Do not allow modification of existing data

□ Do not allow deletion of existing data

□ Do not allow addition of new data

Figure 43: Form Wizard step 6

15) Click the **Next >** button. Base displays step 7 of the Form Wizard (Figure 44). Applying styles is a matter of taste. Just beware: some styles involve unexpectedly low-contrast images, especially in table control fields. In such cases, the font color of the data sheet fields may need to be readjusted.



Form Wizard

Steps

1. Field selection
2. Set up a subform
3. Add subform fields
4. Get joined fields
5. Arrange controls
6. Set data entry
7. Apply styles
8. Set name

Apply the style of your form

Apply styles

- Beige
- Bright Blue
- Light Gray
- Dark
- Orange
- Ice Blue
- Grey
- Water
- Red
- Violet

Field border

○ No border

● 3D look

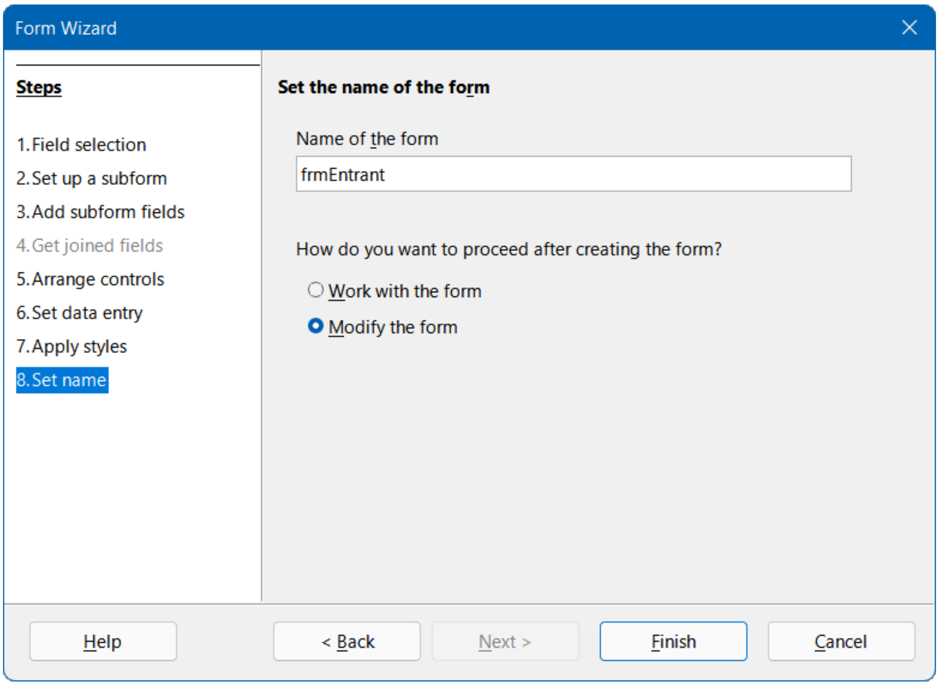
○ Flat

Figure 44: Form Wizard step 7

16) Click the **Next >** button. Base displays step 8 of the Form Wizard.

17) Type the text “frmEntrant” into the *Name of the form* field.

18) Select the **Modify the form** option (Figure 45).



Form Wizard

Steps

1. Field selection
2. Set up a subform
3. Add subform fields
4. Get joined fields
5. Arrange controls
6. Set data entry
7. Apply styles
8. Set name

Set the name of the form

Name of the form: frmEntrant

How do you want to proceed after creating the form?

○ Work with the form

● Modify the form

Figure 45: Form Wizard step 8

19) Click the **Finish** button to close the Form Wizard.

On exiting the Form Wizard, you can see the initial version of your form displayed in the Database Form window. Hopefully is resembles the version shown in Figure 46.

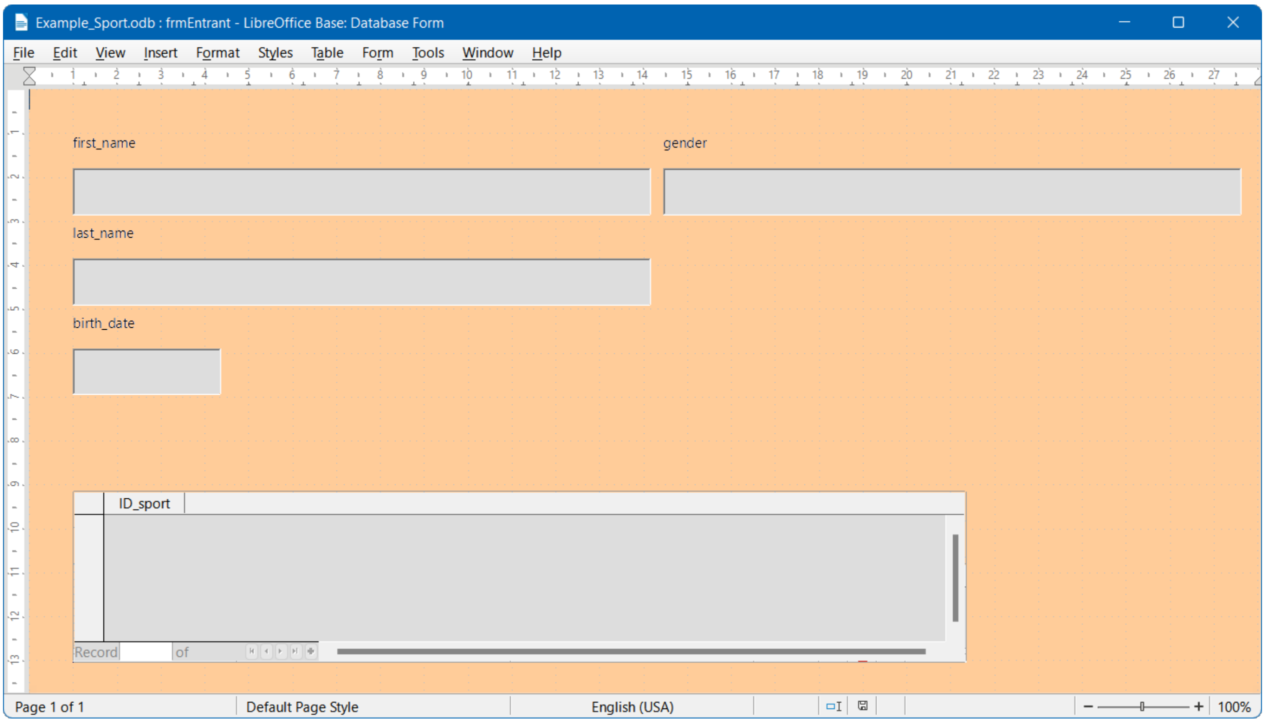

Example_Sport.odb : frmEntrant - LibreOffice Base: Database Form
File Edit View Insert Format Styles Table Form Tools Window Help
first_name
gender
last_name
birth_date
ID_sport
Record
of
Page 1 of 1
Default Page Style
English (USA)
100%

Figure 46: Initial version of frmEntrant form

In this initial version, the subform requires the user to enter the abbreviation for each sport which the entrant wishes to compete in. A significant improvement would be to provide a menu allowing selection of sports using their full names. This is achieved through the following actions.

1) On the subform area at the bottom of the form, right-click the *ID_sport* table header and select **Replace with > List Box** in the context menu.

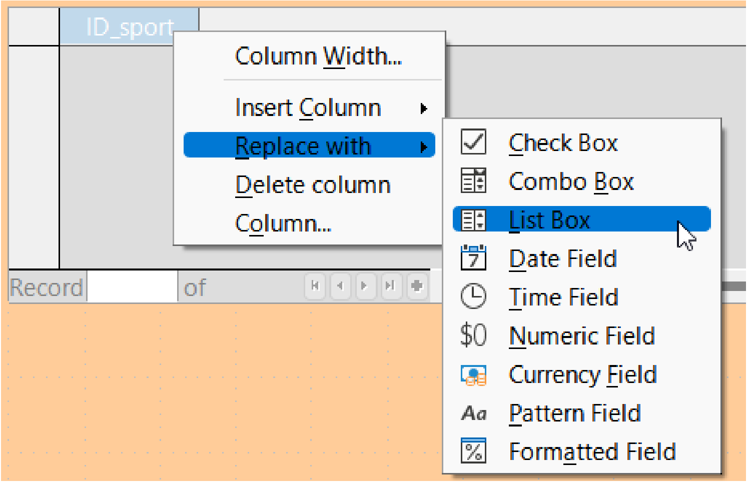
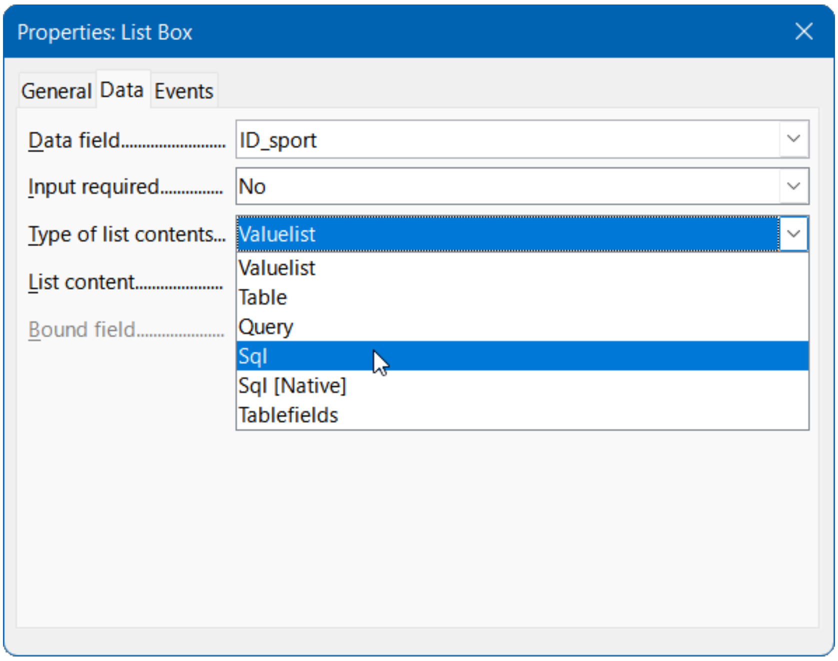


Figure 47: Convert ID_sport to List Box

2) Next the list should be modified so that it can display the intended sports data. Right-click again on the *ID_sport* table heading but this time select **Column** in the context menu. Base displays the Properties: List Box dialog.

3) On the *Data* tab of the Properties: List Box dialog, change the *Type of list contents* to **Sql** (Figure 48). With the help of SQL, the list will get its content from the *tblSports* table.



Properties: List Box

Data field: ID_sport

Input required: No

Type of list contents: Valuelist

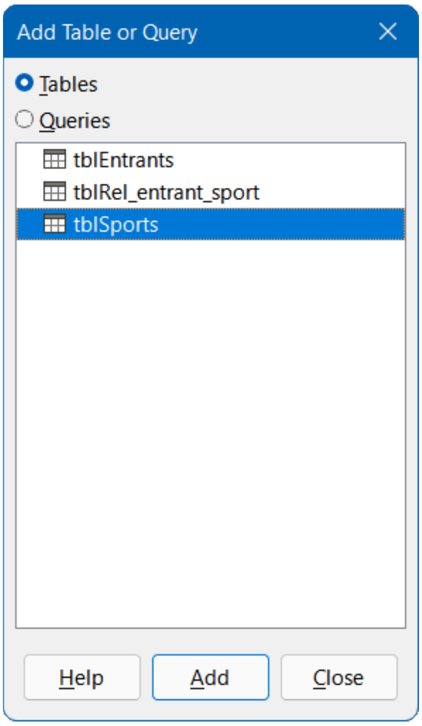
List content

Bound field

Figure 48: Setting List Box to use an SQL query

4) Click the ellipsis button (…) to the right of the *List content* field to open the editor for creating queries. A suitable query will be created and finally saved in the list box itself.

5) Base opens the Query Design window and the Add Table or Query dialog (Figure 49) in readiness for you to start creating a query.



Add Table or Query

● Tables

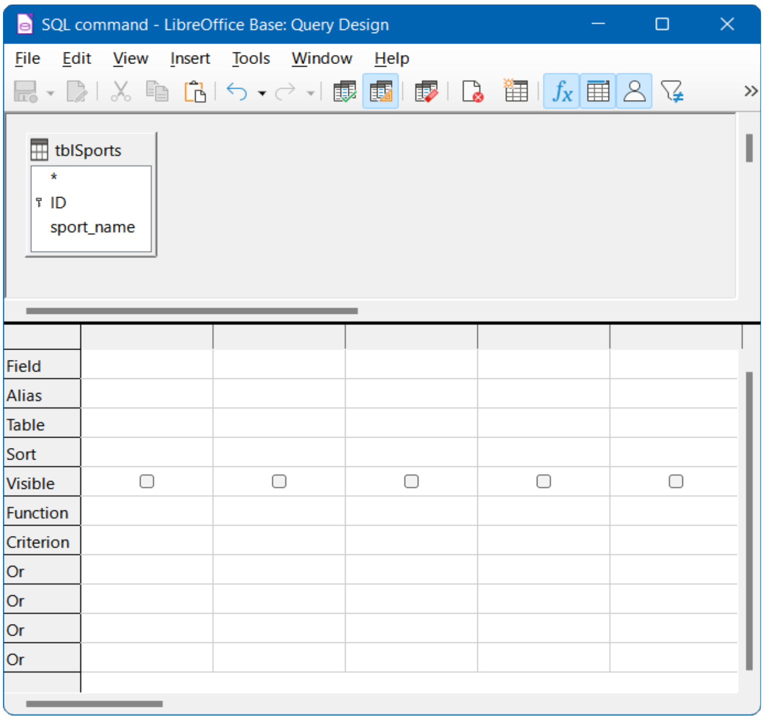
○ Queries

Figure 49: Add Table or Query dialog

6) In the Add Table or Query dialog, select the **Tables** radio button and then select the **tblSports** entry.

7) Click the **Add** button and then click the **Close** button.

8) Base removes the Add Table or Query dialog and displays a box representing the *tblSports* table in the upper part of the Query Design window (Figure 50).

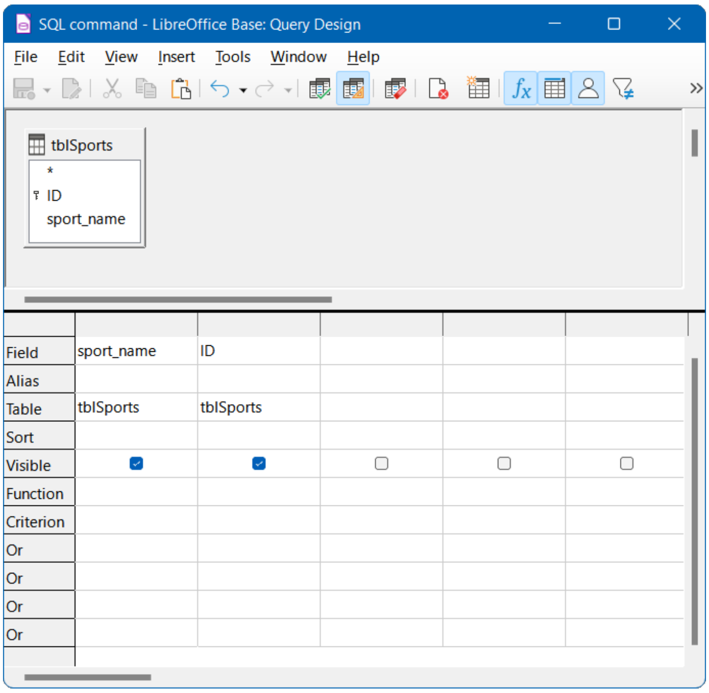

SQL command - LibreOffice Base: Query Design
File Edit View Insert Tools Window Help
tblSports
*
ID
sport_name

Field					
Alias					
Table					
Sort					
Visible	□	□	□	□	□
Function					
Criterion					
Or					
Or					
Or					
Or					

Figure 50: Query Design window showing the tblSports table

9) In the first column of the lower part of the window, click in the box next to *Field* and select *tblSports.sport_name* from the drop-down menu.

10) In the second column, select the *tblSports.ID* field (Figure 51). This field afterwards passes its value to the *tblSports* table, which is the data source of the subform. This allows the defined words to be displayed and the appropriate shortcuts to be stored.


SQL command - LibreOffice Base: Query Design
File Edit View Insert Tools Window Help
tblSports
*
ID
sport_name

Field	sport_name	ID			
Alias					
Table	tblSports	tblSports			
Sort					
Visible	☑	☑	□	□	□
Function					
Criterion					
Or					
Or					
Or					
Or					

Figure 51: Query design for populating ID_sport list box on frmEntrant form

11) Save the query, which is transferred to the properties of the list box. Close the query editor.

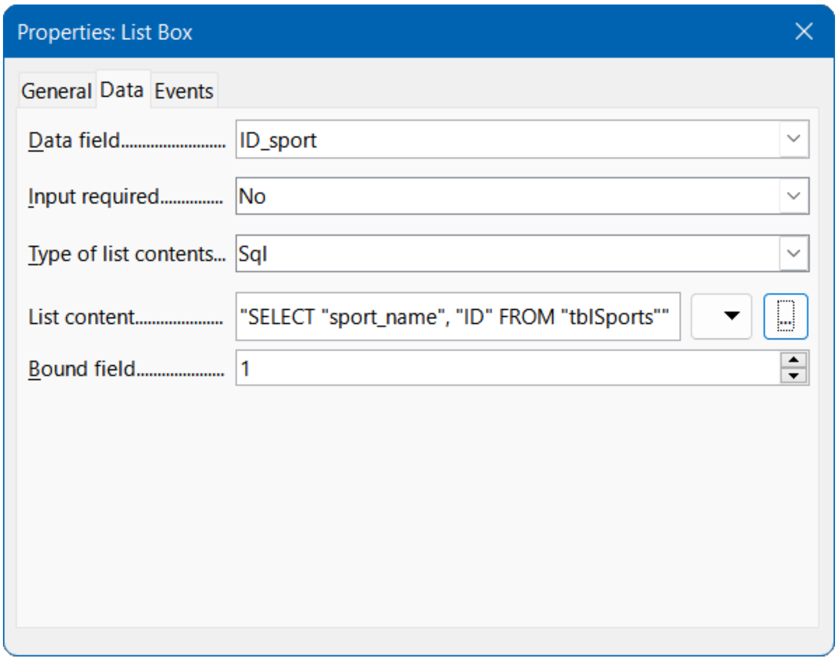
12) Now the *List content* field of the Properties: List Box dialog shows the SQL code that was created in the query editor (Figure 52):

```
SELECT "sport_name", "ID" FROM "tblSports"
```

This code says: From the *tblSports* table, select the *sport_name* field and the associated primary key value *ID*.

Tip

It is often useful to review the automatically generated SQL code so that you become more familiar with the query language.



Properties: List Box

Data field: ID_sport

Input required: No

Type of list contents: Sql

List content: "SELECT "sport_name", "ID" FROM "tblSports""

Bound field: 1

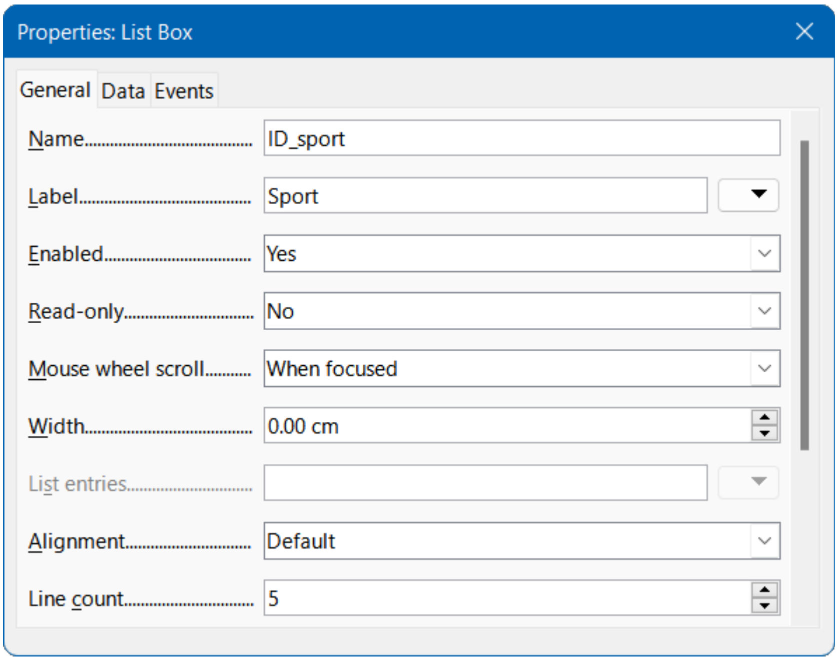
Figure 52: SQL code in List content field

This query illustrates the minimum that should be selected. Of course, sorting could be incorporated. Saving carefully selected abbreviations gives a useful list of sports stored in *ID*. If records are not sorted in a specified way, the sorting is always performed by the primary key field. In order to see sports later in the list box, this content must be entered in the *tblSports* table.

To change the label for the *ID_sport* field to be shown as *Sport*, carry out the following steps:

1) On the Properties: List Box dialog, click on the *General* tab.

2) Replace the contents of the *Label* field with “Sport” (Figure 53).



Properties: List Box

General | Data | Events

Name	ID_sport
Label	Sport
Enabled	Yes
Read-only	No
Mouse wheel scroll	When focused
Width	0.00 cm
List entries	
Alignment	Default
Line count	5

Figure 53: Changing the display label for ID_sport

3) Close the Properties: List Box dialog.

Changing the names of other fields in the form is best done through the Form Navigator. If fields are clicked, not only the fields, but also the associated labels are selected. Through the wizard, they were grouped together. This then requires further action from the selection's context menu.

The Form Navigator is accessed by clicking an icon on the Form Design toolbar, which when visible is normally located at the bottom of the Database Form window. Figure 54 shows the position of the **Form Navigator** icon on the Form Design toolbar, highlighted with a red circle.

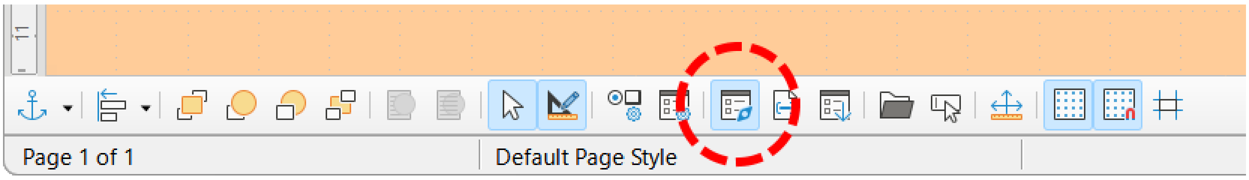


Figure 54: Location of Form Navigator icon on the Form Design toolbar

Tip

If the Form Design toolbar is not visible you can access it by selecting **View > Toolbars > Form Design** on the Menu bar.

Each field can be examined individually with the Form Navigator, and the field properties are then accessible in the context menu. A property entry is automatically saved after going to another property. It is possible to jump from one field to another even when the Properties dialog is open. Here also the respective intermediate level is stored.

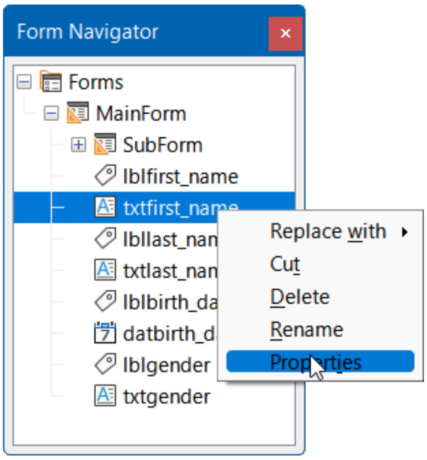


Figure 55: Accessing the Properties dialog from the Form Navigator

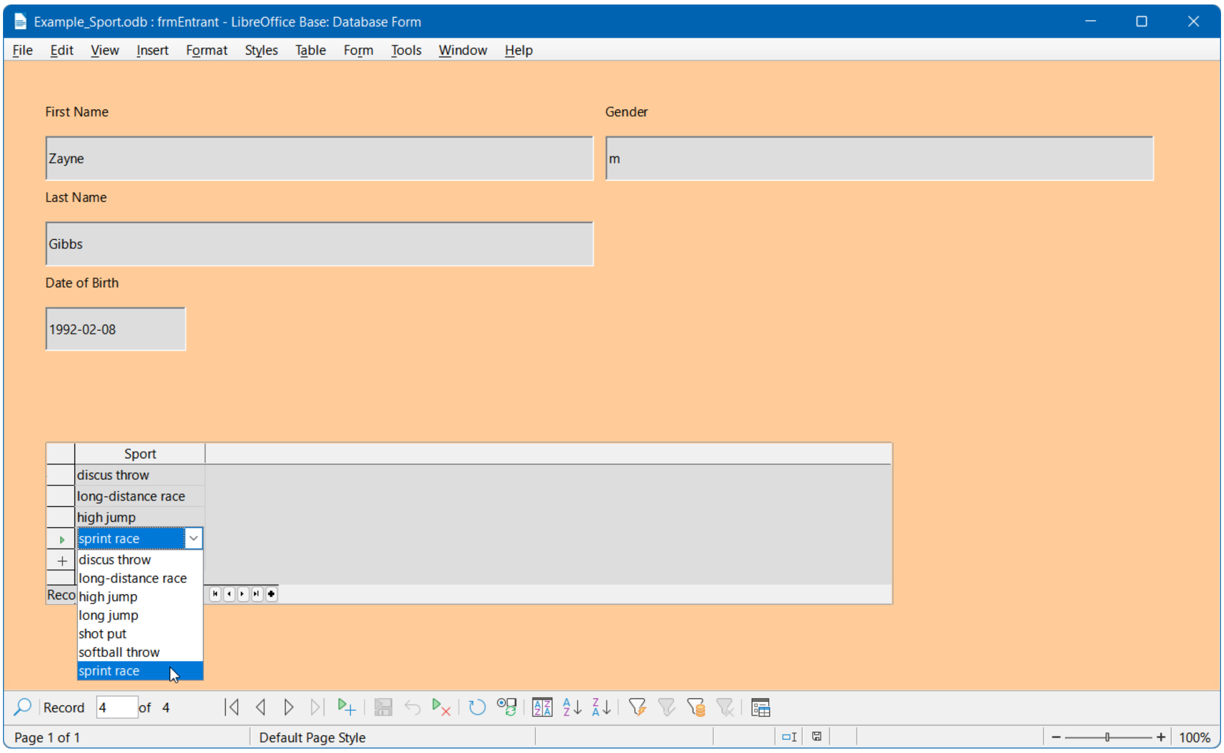
If the design has now been completed, save and close the form. Then save the Base file again.

For the purposes of testing the form, assume that the *tblSports* table has been populated with the data listed in Table 3.

Table 3: Data entered in the tblSports table

ID	***sport_name***
disc	discus throw
dist	long distance race
high	high jump
long	long jump
shot	shot put
soft	softball throw
sprt	sprint race

The first entrant to the competition is Zayne Gibbs and he would like to enter four events—discus throw, long-distance race, high jump, and sprint race. After entering data into the form, it might look something like that shown in Figure 56. Do not forget to save the record after entering the data.


Example_Sport.odb : frmEntrant - LibreOffice Base: Database Form
File Edit View Insert Format Styles Table Form Tools Window Help

First Name: Zayne

Gender: m

Last Name: Gibbs

Date of Birth: 1992-02-08

Sport
discus throw
long-distance race
high jump
sprint race

discus throw
long-distance race
high jump
long jump
shot put
softball throw
sprint race

Record 4 of 4

Page 1 of 1 Default Page Style 100%

Figure 56: Sample data in the frmEntrant form

Of course, sorting mechanisms could be incorporated in such a form.

When using the form, you may notice some inconveniences:

- The *Tab* key provides a convenient mechanism for quickly navigating through the fields of the form as data is entered. However, with the cursor in the *gender* field of the form, a further press of the *Tab* key jumps directly to the next *tblEntrants* record. The subform listing sports is not directly accessible using the *Tab* key.
- If the cursor lies within any of the four fields in the upper part of the from, the navigation bar within the Form Navigation toolbar at the bottom of the window shows the record number within the *tblEntrants* table. However, if working in the subform, then that navigation bar switches to showing the number of the *tblSports* record within the subform. This information is already present in the dedicated navigation bar within the subform area and so it would be preferable if the navigation bar within the Form Navigation toolbar always showed the record numbers of the entrant within the *tblEntrants* table.
- When entering an entrant’s gender, the user must remember the set of possible options. Currently the only limitation defined in the *tblEntrants* table is the limit of one character and a safer method of entry should be provided.

To solve each of these problems in turn, first open the form for editing rather than data entry (Figure 57).

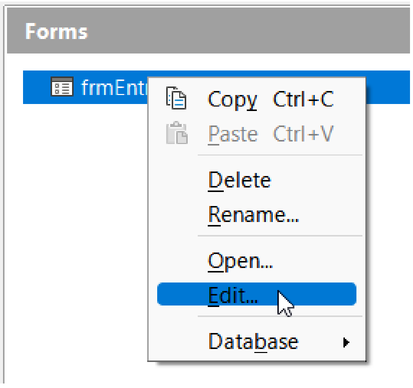


Figure 57: Open frmEntrant form for editing

Tabbing to the subform

In order not to go straight to the next entrant record using the *Tab* key after entering the entrant's gender, it is necessary to modify the activation sequence.

1) Click **Activation Order** on the Form Design toolbar, which when visible is normally located at the bottom of the Database Form window. Figure 58 shows the position of the **Activation Order** icon on the Form Design toolbar, highlighted with a red circle.

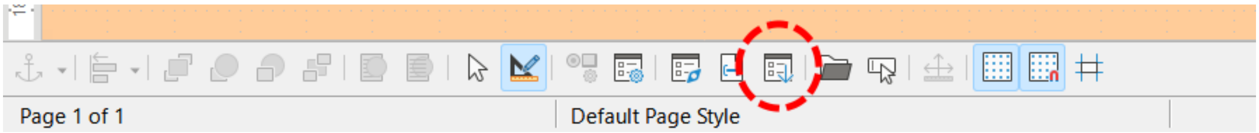


Figure 58: Location of Activation Order icon on the Form Design toolbar

2) Base displays the Tab Order dialog (Figure 59).

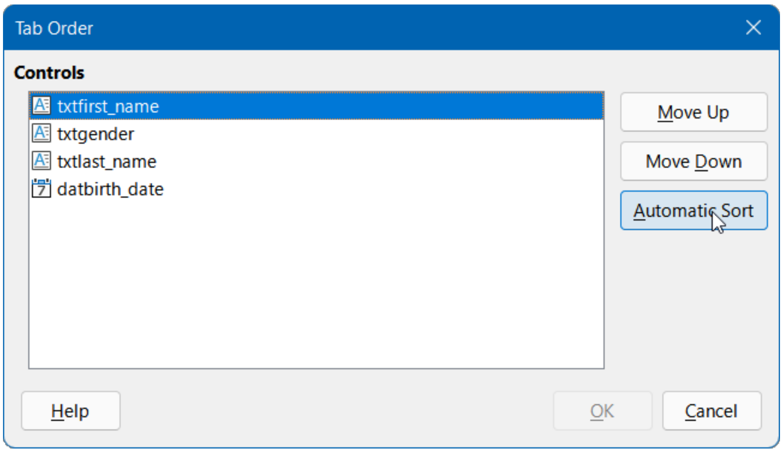


Figure 59: Tab Order dialog

3) Click the **Automatic Sort** option, which affects not only sorting of the displayed controls, but also the automatic redirection into the subform. Although not obvious from the dialog, it works this way in the background.
4) Click the **OK** button to close the Tab Order dialog.

Later, exit edit mode, open the form for data entry, make sure that the tab order for the *frmEntrant* form is now as expected, and that you can tab into the subform.

Managing the navigation bars of the main form and the subform

1) Click the **Form Navigator** icon on the Form Design toolbar (Figure 54) to open the Form Navigator.

2) Right-click the *SubForm* entry in the Form Navigator and select the **Properties** item in the context menu (Figure 60).

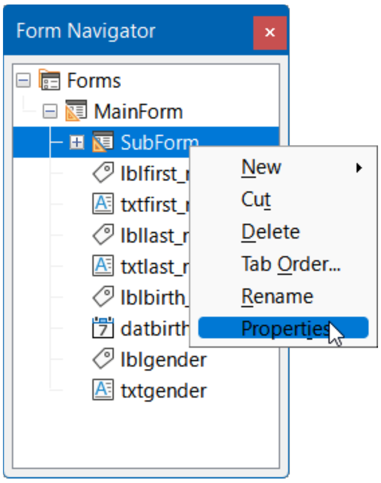
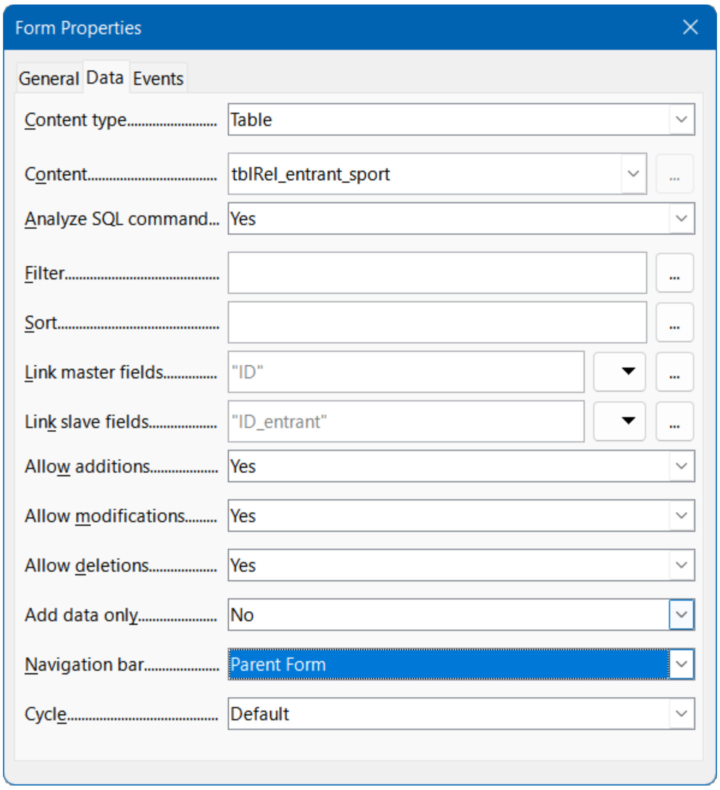


Figure 60: Access dialog to change subform properties

3) Base displays the Form Properties dialog.

4) On the *Data* tab, change the setting of the *Navigation bar* field to **Parent Form** (Figure 61).



Content type	Table
Content	tblRel_entrant_sport
Analyze SQL command	Yes
Filter	
Sort	
Link master fields	"ID"
Link slave fields	"ID_entrant"
Allow additions	Yes
Allow modifications	Yes
Allow deletions	Yes
Add data only	No
Navigation bar	Parent Form
Cycle	Default

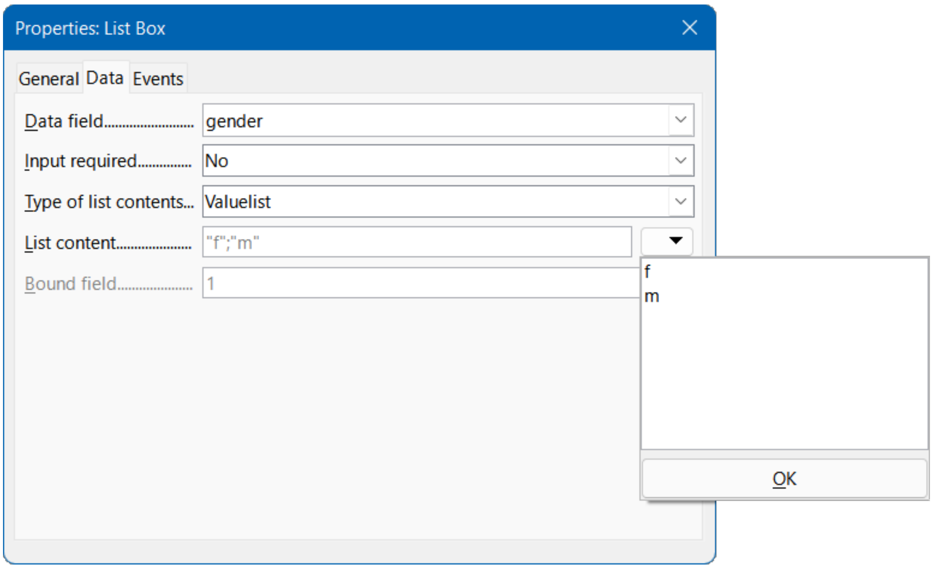
Figure 61: Change subform Navigation bar setting

Later, exit edit mode, open the form for data entry, and make sure that the navigation bar in the Form Navigation toolbar always shows the position of the current record within the *tblEntrants* table, even when working in the subform.

Restrict entry into a control

To limit input to specified values, the control cannot be a simple text box. One solution would be to use a Group Box, as described in Chapter 4, Forms. Another solution is to present the choices in a List Box, as follows:

1) Click the **Form Navigator** icon on the Form Design toolbar (Figure 54) to open the Form Navigator.
2) Right-click the Text Box field for entering an entrant’s gender and select **Replace > List Box** in the context menu.
3) Right-click the same field again but this time select the **Properties** option in the context menu.
4) Base displays the Properties: List Box dialog.
5) On the *Data* tab, click the down arrow at the right of the *List content* field. Enter the letter “f” on the first line of the text entry box that appears, type “m” on the next line, and then click the **OK** button (Figure 62). These abbreviations are the values that will be passed to the *tblEntrants* table.



Properties: List Box

General Data Events

Data field: gender

Input required: No

Type of list contents: Valuelist

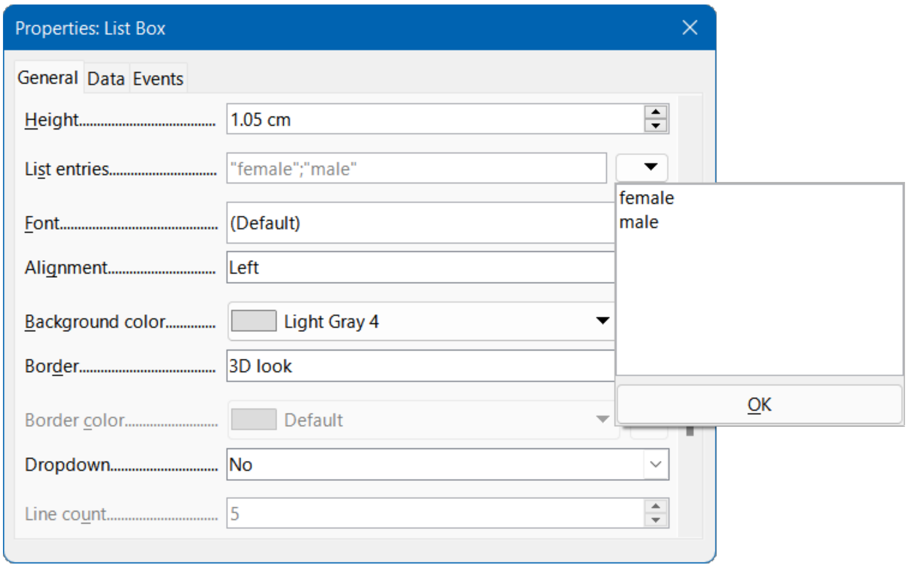
List content: "f";"m"

Bound field: 1

f
m

Figure 62: Setting values allowed in the gender field (Data tab)

6) Click on the *General* tab.
7) Scroll down this tab until you reach the *List entries* field. Click the down arrow at the right of the *List entries* field. Enter the word “female” on the first line of the text entry box that appears, type “male” on the next line, and then click the **OK** button (Figure 63). These are the values that the user will select from and must be in the same order as the values entered in the *List content* field on the *Data* tab.



Height 1.05 cm

List entries "female";"male"

female
male

Font (Default)

Alignment Left

Background color Light Gray 4

Border 3D look

Border color Default

Dropdown No

Line count 5

Figure 63: Setting values allowed in the gender field (General tab)

8) In the *Dropdown* field that is located a little lower down the tab, select the **Yes** option.
9) Make sure that you have saved all changes to the form and then saved the database.

Later, exit edit mode, open the form for data entry, and make sure that only valid inputs can be made to the *gender* field.

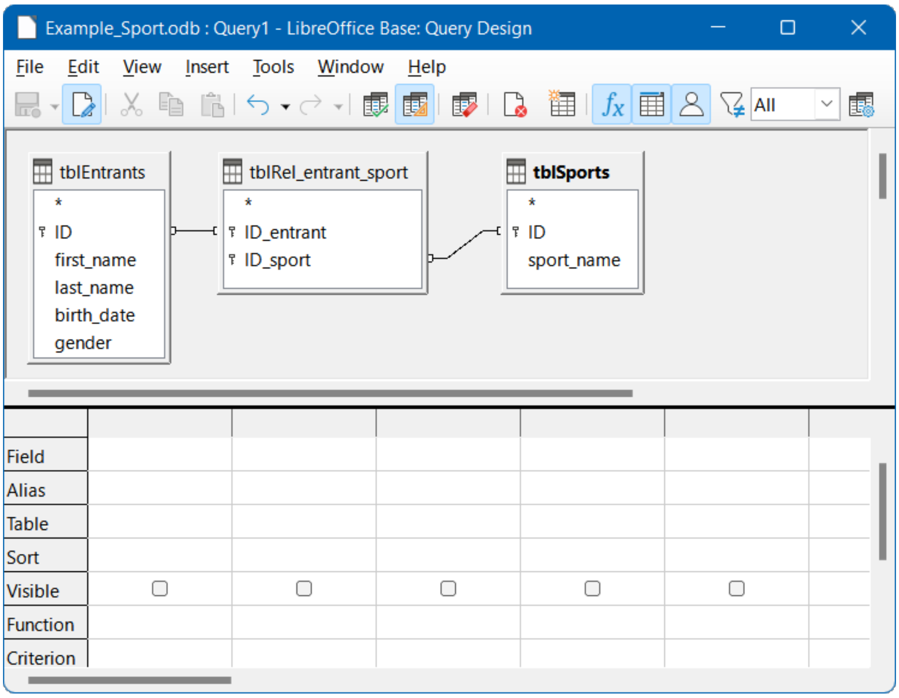
The most striking inconveniences have thus been eliminated. The input for male and female entrants may begin, as well as their assignment to the sports.

The following step is useful: records should be entered only once. Take care to ensure that the entrants can also compete with each other by age and by sport. Otherwise the subsequent queries and reports may not make sense.

Creating a query

In a query, the contents of various tables can be grouped together. In this section, a simple query is created that displays each of the entrants together with a list of the sports that they have entered.

1) Click **Queries** in the *Database* area of the main Base window.
2) Click **Create Query in Design View** in the *Tasks* area of the main Base window.
3) Base displays the Query Design window and opens the Add Table or Query dialog (Figure 49).
4) Select the *tblEntrants* table and click the **Add** button.
5) Select the *tblRel_entrant_sport* table and click the **Add** button.
6) Select the *tblSports* table and click the **Add** button.
7) Click the **Close** button to remove the Add Table or Query dialog from the display.
8) Note that the three tables, and the relationships between them, are represented in the upper area of the Query Design window (Figure 64).

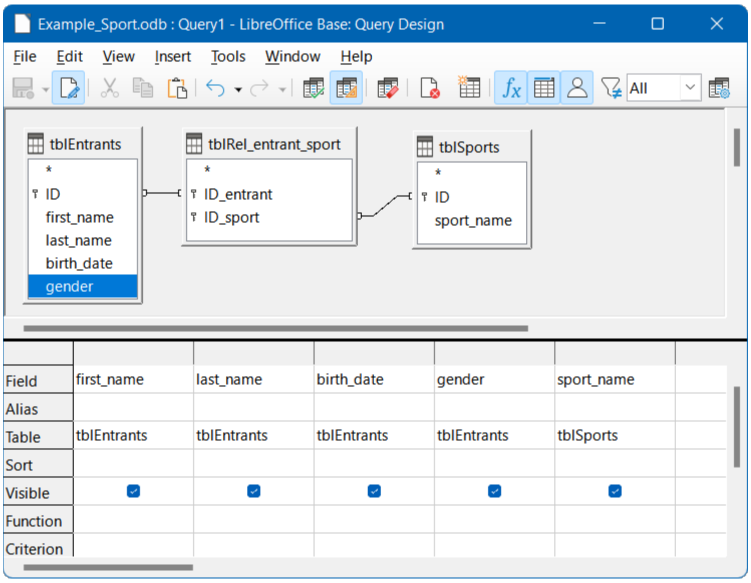


Field					
Alias					
Table					
Sort					
Visible	□	□	□	□	□
Function					
Criterion					

Figure 64: Query Design window with three tables added

9) Double-click the *first_name*, *last_name*, *birth_date*, and *gender* fields in the *tblEntrants* table and verify that these fields are added to the columns of the query in the lower part of the Query Design window. Double-clicking is perhaps the simplest method to add a field to a query.

10) In the lower part of the Query Design window, click in the blank *Field* entry located to the right of *gender* and then click the down arrow to access a drop-down list of fields that are available for inclusion in the query. The list consists of the field names and their corresponding table names indexed by table name. In order to match the fields of the tables correctly to their tables, they are labeled "table name"."field name" in queries. If "*" is used instead of the field name, it means that all the fields of the corresponding table are displayed. Select the *tblSports.sport_name* entry in the list—this provides an alternative to the method of double-clicking described in the previous step (Figure 65).



Field	first_name	last_name	birth_date	gender	sport_name
Alias					
Table	tblEntrants	tblEntrants	tblEntrants	tblEntrants	tblSports
Sort					
Visible	☑	☑	☑	☑	☑
Function					
Criterion					

Figure 65: Query Design window showing five fields added to the query

The query is complete and it is advisable to execute it before saving to see if it produces the desired results.

To run a query from the Query Design window, click the **Run Query** icon on the Query Design toolbar which is located by default at the top of the window (Figure 66).

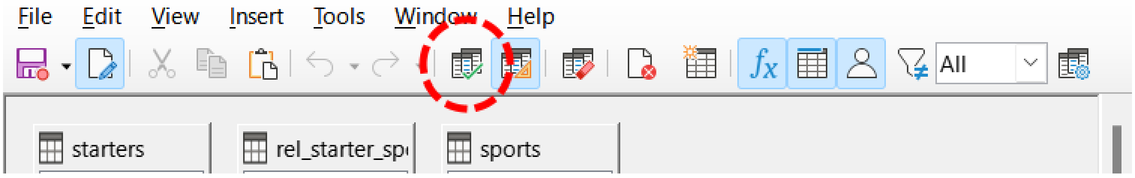
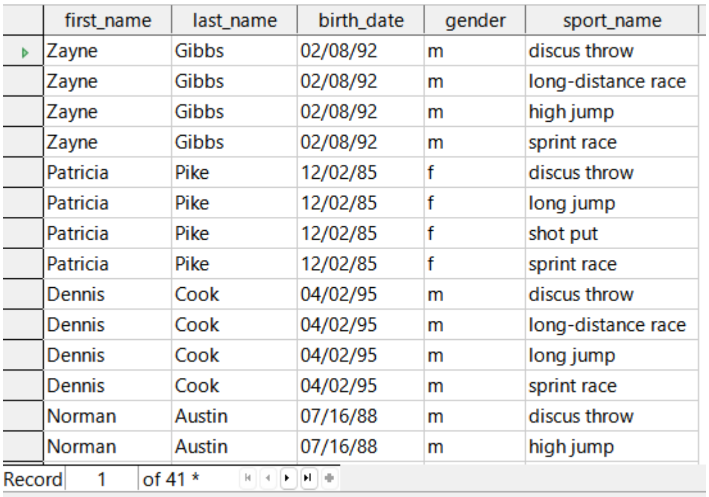


Figure 66: Location of Run Query icon

The outputs from running this simple query are displayed in a list positioned in the upper part of the Query Design window (Figure 67). The query displays all combinations of entrants and sports. If entrants have entered multiple sports, they have as many records. Entrants do not appear without sports.



first_name	last_name	birth_date	gender	sport_name
Zayne	Gibbs	02/08/92	m	discus throw
Zayne	Gibbs	02/08/92	m	long-distance race
Zayne	Gibbs	02/08/92	m	high jump
Zayne	Gibbs	02/08/92	m	sprint race
Patricia	Pike	12/02/85	f	discus throw
Patricia	Pike	12/02/85	f	long jump
Patricia	Pike	12/02/85	f	shot put
Patricia	Pike	12/02/85	f	sprint race
Dennis	Cook	04/02/95	m	discus throw
Dennis	Cook	04/02/95	m	long-distance race
Dennis	Cook	04/02/95	m	long jump
Dennis	Cook	04/02/95	m	sprint race
Norman	Austin	07/16/88	m	discus throw
Norman	Austin	07/16/88	m	high jump

Record 1 of 41 *

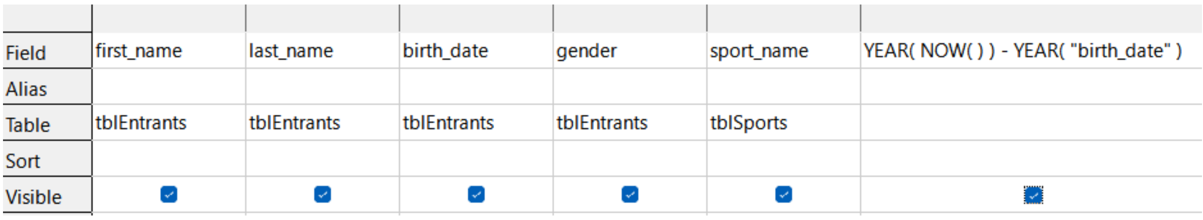
Figure 67: Output of query

To extend the query a little further, suppose that age limits are to be applied to some events. In that case, it will be important to know each entrant's age so that they can be placed in the correct age bracket for each of their events. A straightforward step to help with this would be to determine an entrant's age based on their date of birth (held in the *tblEntrants* table) and the current date, including the result of that calculation in the query.

A simple way to achieve this is to use two of HSQLDB's built-in functions:

- **NOW()**. The NOW function returns a date-time value representing the moment the function is called, that is the current date and time.
- **YEAR(dt)**. The YEAR function extracts the year component of its argument *dt*, which is a date-time value.

Type the formula `YEAR(NOW())-YEAR("birth_date")` into the next blank *Field* cell in the query, as shown in Figure 68.

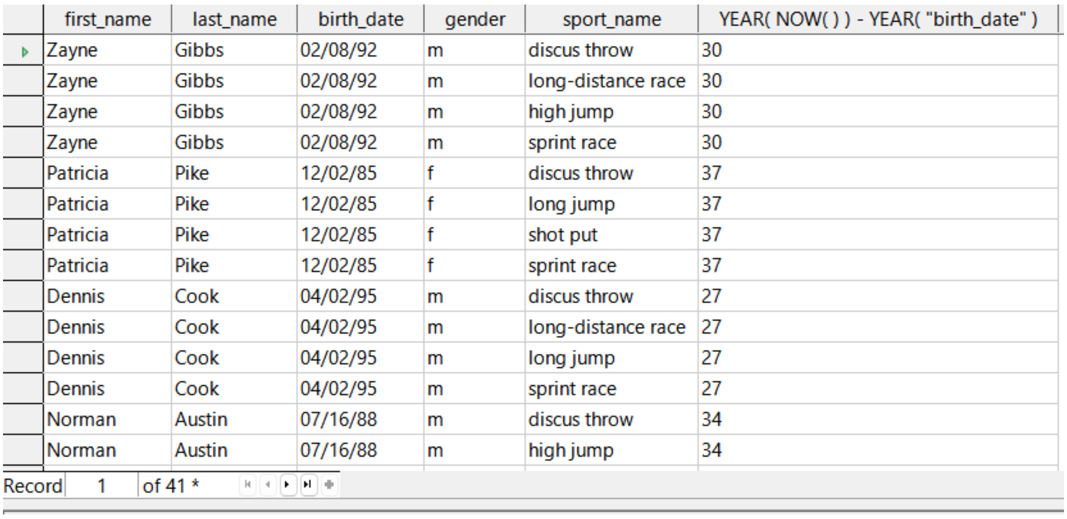


Field	first_name	last_name	birth_date	gender	sport_name	YEAR(NOW( )) - YEAR("birth_date")
Alias						
Table	tblEntrants	tblEntrants	tblEntrants	tblEntrants	tblSports	
Sort						
Visible	☑	☑	☑	☑	☑	☑

Figure 68: Age calculation included in query

The first part of this formula, `YEAR(NOW())`, picks out the current year. The second part, `YEAR("birth_date"),` picks out the year of the entrant's birth. A difference is formed between them, indicating (approximately) the person's current age. The results are shown in Figure 69.

These and many other functions that work with the embedded HSQLDB database engine are described in the remainder of this book and a fuller list can be found in Chapter 9 of the HyperSQL User Guide at http://hsqldb.org/doc/1.8/guide/index.html (see the section entitled Built-in Functions and Stored Procedures).

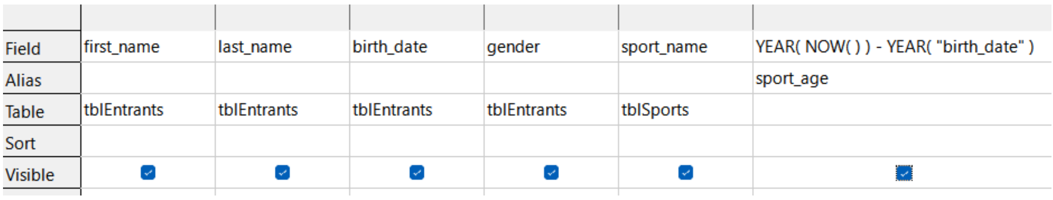


first_name	last_name	birth_date	gender	sport_name	YEAR(NOW( )) - YEAR("birth_date")
Zayne	Gibbs	02/08/92	m	discus throw	30
Zayne	Gibbs	02/08/92	m	long-distance race	30
Zayne	Gibbs	02/08/92	m	high jump	30
Zayne	Gibbs	02/08/92	m	sprint race	30
Patricia	Pike	12/02/85	f	discus throw	37
Patricia	Pike	12/02/85	f	long jump	37
Patricia	Pike	12/02/85	f	shot put	37
Patricia	Pike	12/02/85	f	sprint race	37
Dennis	Cook	04/02/95	m	discus throw	27
Dennis	Cook	04/02/95	m	long-distance race	27
Dennis	Cook	04/02/95	m	long jump	27
Dennis	Cook	04/02/95	m	sprint race	27
Norman	Austin	07/16/88	m	discus throw	34
Norman	Austin	07/16/88	m	high jump	34

Record 1 of 41 *

Figure 69: Query output showing calculated age

By default, the code entered in the field appears in the column heading of the query output and this isn't particularly user-friendly. This is improved by entering an alias to be used as the column heading instead of the code. In the *Alias* row under the field containing the code, enter the term *sport_age*.

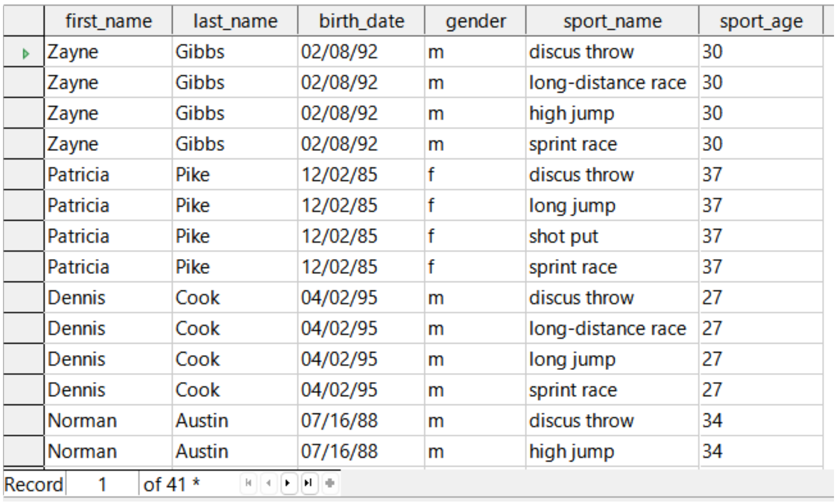


Field	first_name	last_name	birth_date	gender	sport_name	YEAR(NOW( )) - YEAR("birth_date")
Alias						sport_age
Table	tblEntrants	tblEntrants	tblEntrants	tblEntrants	tblSports	
Sort						
Visible	☑	☑	☑	☑	☑	☑

Figure 70: Using an alias to replace a formula

The alias *sport_age* is used to differentiate the value calculated from the person's true age. Generally speaking, the *sport_age* value will match a person's true age only when they have had a birthday in the current year.

If the query is run again, the column header no longer contains the code but instead the string *sport_age*.



first_name	last_name	birth_date	gender	sport_name	sport_age
Zayne	Gibbs	02/08/92	m	discus throw	30
Zayne	Gibbs	02/08/92	m	long-distance race	30
Zayne	Gibbs	02/08/92	m	high jump	30
Zayne	Gibbs	02/08/92	m	sprint race	30
Patricia	Pike	12/02/85	f	discus throw	37
Patricia	Pike	12/02/85	f	long jump	37
Patricia	Pike	12/02/85	f	shot put	37
Patricia	Pike	12/02/85	f	sprint race	37
Dennis	Cook	04/02/95	m	discus throw	27
Dennis	Cook	04/02/95	m	long-distance race	27
Dennis	Cook	04/02/95	m	long jump	27
Dennis	Cook	04/02/95	m	sprint race	27
Norman	Austin	07/16/88	m	discus throw	34
Norman	Austin	07/16/88	m	high jump	34

Record 1 of 41 *

Figure 71: Query output using an alias for the final column heading

Save this query under the name *qrySport_age*. It is then used as the basis for the next query, where slightly more complex formulas are used to assign people to age groups.

1) Click **Queries** in the *Database* area of the main Base window.
2) Click **Create Query in Design View** in the *Tasks* area of the main Base window.
3) Base displays the Query Design window and opens the Add Table or Query dialog (Figure 49).
4) Select the **Queries** option at the top of the Add Table or Query dialog.
5) Select the qry*Sport_age* query.
6) Click the **Add** button and then click the **Close** button to remove the Add Table or Query dialog (Figure 72). Note that the icon that appears in front of the title *qrySport_age* is not the same as that used for tables in this window (see Figure 64 for example). This different symbol is a visual indication that the basis of this query is another query rather than a table.

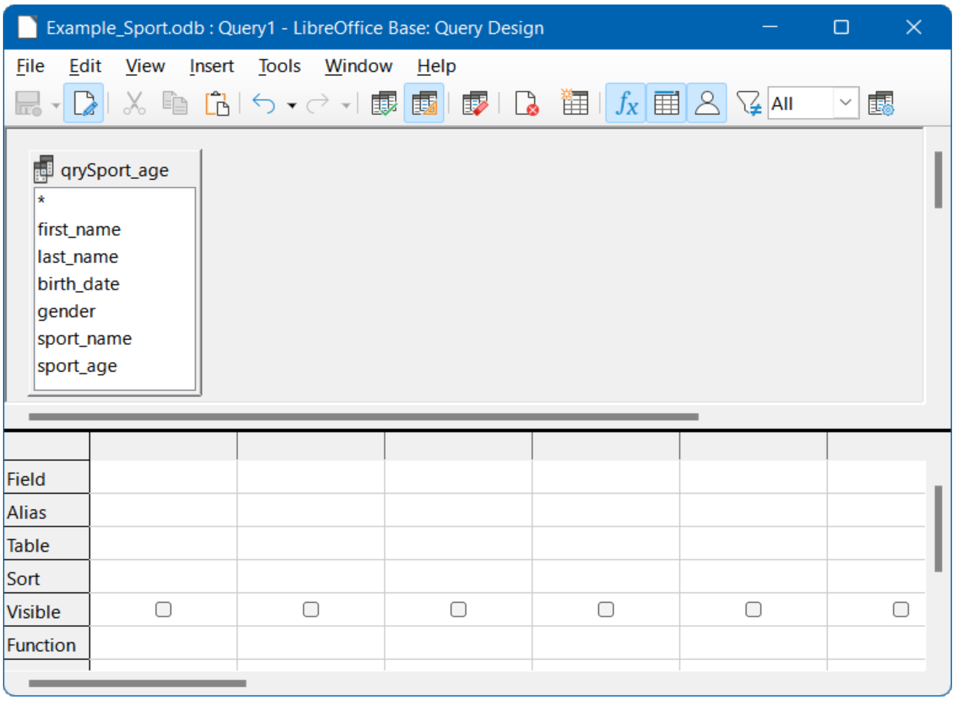


Figure 72: Query Design window with qrySport_age query added

7) In the *qrySport_age* query area, double-click on the entry comprising only an asterisk (*). This has the effect of selecting all fields in the *qrySport_age* query. An alternative method to achieve the same result would have been to click in the first *Field* entry and select *qrySport_age.** in the drop-down menu.

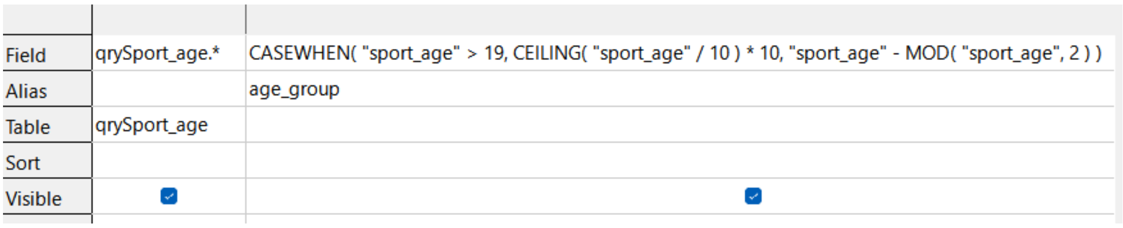
Note

If you run the new query at this point, you will see that it produces the same output as the *qrySport_age* query but without using a formula.

The *sport_age* field in the *qrySport_age* query will now be used to determine in which age group each person participates. In order that the calculation is not too complex, tor those under 20 years of age, the starters are divided into age groups containing two ages per group beginning with 0. From 20 years and older, groups containing 10 ages each are formed, for example, 20–29 years.

8) Enter the following formula into the first empty *Field* cell: `CASEWHEN( "sport_age" > 19, CEILING( "sport_age" / 10 ) * 10, "sport_age" - MOD( "sport_age", 2 ) )`.

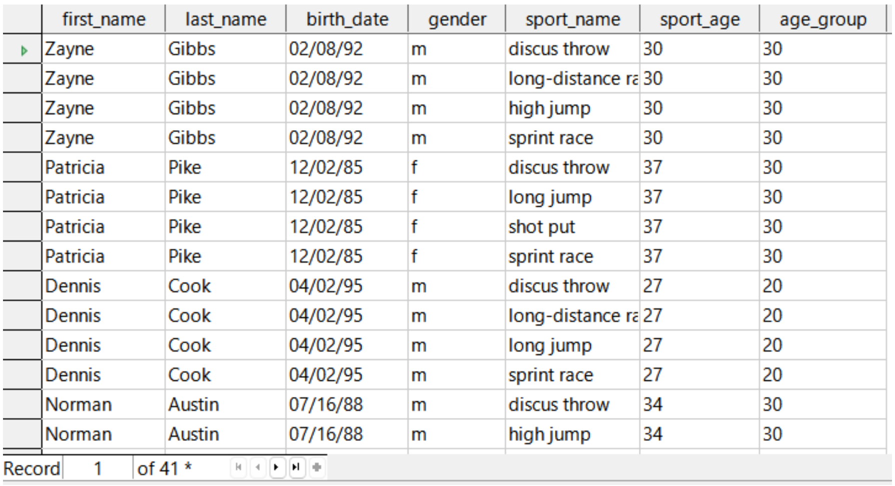
9) Enter the string “*age_group*” into the corresponding *Alias* cell (Figure 73).



Field	qrySport_age.*	CASEWHEN("sport_age" > 19, CEILING("sport_age" / 10) * 10, "sport_age" - MOD("sport_age", 2))
Alias		age_group
Table	qrySport_age	
Sort		
Visible	☑	☑

Figure 73: Formula to group starters into age groups

10) Click the **Run Query** icon in the Query Design toolbar. The output of the query is shown in Figure 74.



first_name	last_name	birth_date	gender	sport_name	sport_age	age_group
Zayne	Gibbs	02/08/92	m	discus throw	30	30
Zayne	Gibbs	02/08/92	m	long-distance ra	30	30
Zayne	Gibbs	02/08/92	m	high jump	30	30
Zayne	Gibbs	02/08/92	m	sprint race	30	30
Patricia	Pike	12/02/85	f	discus throw	37	30
Patricia	Pike	12/02/85	f	long jump	37	30
Patricia	Pike	12/02/85	f	shot put	37	30
Patricia	Pike	12/02/85	f	sprint race	37	30
Dennis	Cook	04/02/95	m	discus throw	27	20
Dennis	Cook	04/02/95	m	long-distance ra	27	20
Dennis	Cook	04/02/95	m	long jump	27	20
Dennis	Cook	04/02/95	m	sprint race	27	20
Norman	Austin	07/16/88	m	discus throw	34	30
Norman	Austin	07/16/88	m	high jump	34	30

Record 1 of 41 *

Figure 74: Query output showing age group for each starter

11) Save the query with the name *qryRegistration*.

Such formulas should not really be included in an entry-level introduction to Base and a simpler allocation of ages would be possible. However, although complex, the formula does show the power of the available facilities.

Looking at its components in a little more detail:

```
CASEWHEN( "sport_age" > 19, CEILING( "sport_age" / 10 ) * 10,
"sport_age" - MOD( "sport_age", 2 ) )
```

- CASEWHEN is another built-in function of HSQLDB and it takes three arguments. In a more abstract form, a call to CASEWHEN may be written as `CASEWHEN(expression, value1, value2)`. The function works by evaluating *expression* and if it is true, *value1* is returned; otherwise *value2* is returned.
- The CEILING function takes one argument. If the call to the function is written as `CEILING(d)`, then the function returns the smallest integer that is not less than *d*.
- The MOD function takes two arguments. If the call to the function is written as `MOD(a, b)` then the function returns *a* modulo *b* (that is, the remainder after dividing *a* by *b*).

Switching off the design view on the Query Design window is not really necessary, since all entries are possible in the design view without any major problems. However, from an educational perspective, it can be instructive to look more closely at the code underlying a query.

It is also worth remembering that there are SQL expressions that fit poorly in the design view, or are not even possible there. In this case you may in the future wish to enter SQL code directly (with the design view switched off).

Click the **Switch Design View On/Off** icon on the Design Query toolbar (Figure 75).

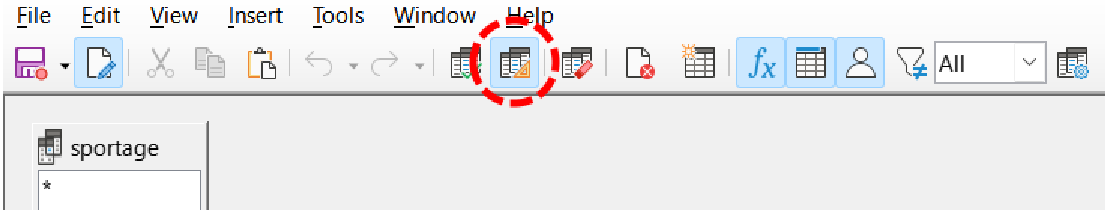
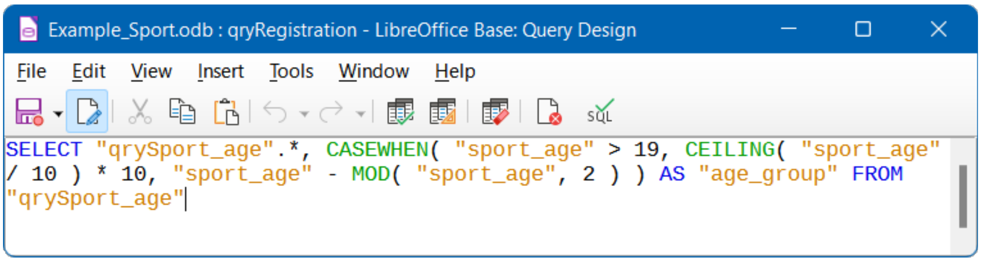


Figure 75: Location of Switch Design View On/Off on Query Design toolbar

With design view switched off, the Query Design window shows the SQL code as in Figure 76.


Example_Sport.odb : qryRegistration - LibreOffice Base: Query Design
File
Edit
View
Insert
Tools
Window
Help

```
SELECT "qrySport_age".*, CASEWHEN( "sport_age" > 19, CEILING( "sport_age"
/ 10 ) * 10, "sport_age" - MOD( "sport_age", 2 ) ) AS "age_group" FROM
"qrySport_age"
```

Figure 76: SQL code for qryRegistration query

The conventions used for displaying code are as follows:

- Field and table names are enclosed in double quotes and displayed in an ocher color.
- SQL terms are displayed in blue.
- Database function names are displayed in green.

Having already discussed the components of the formula, focus can switch more to the structure of the SQL:

```
SELECT "qrySport_age".*, ...  AS "age_group" FROM "qrySport_age"
```

Select from the *qrySport_age* query all the records from and in addition to those determined by the formula. Those that are determined by the formula, refer to as *age_group*.

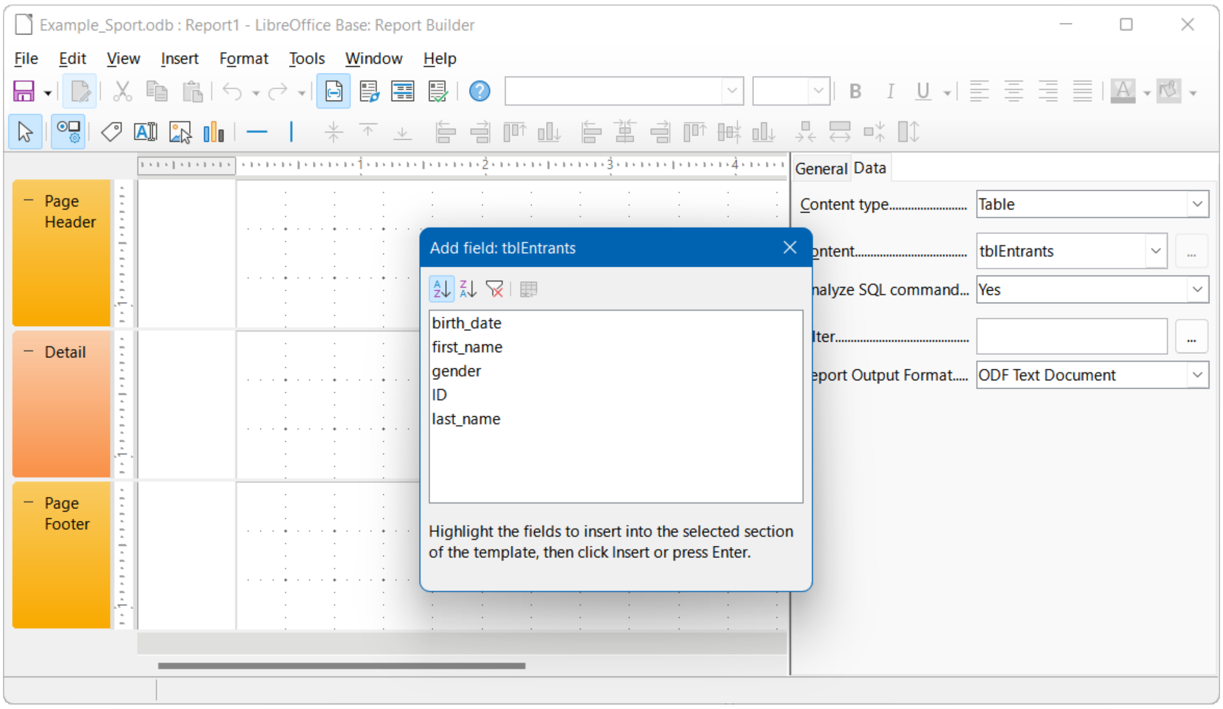
The code does not distinguish between tables and queries as the basis of the data. It therefore works only in the graphical user interface of Base. A query cannot have the same name as a table; a table cannot have the same name as a query.

When the **SQL** icon in the SQL toolbar is clicked (to run the SQL command directly) while in SQL view, the database responds with an error message. The embedded HSQLDB engine does not know the query *qrySport_age* to which the *qryRegistration* query refers.

Creating a report

A report will be created showing a list of entrants, sorted by sport, gender, and age group.

1) On the main Base window, click **Reports** in the *Database* area.
2) In the *Tasks* area, click **Create Report in Design View**.
3) Base displays the Report Builder window, with the Add field dialog in the foreground.
4) On the right side of the Report Builder window, there should be an area to view and edit properties of the currently active object. If this area is not visible, select **View > Properties** on the Menu bar. (Figure 77). The initial configuration of the *Properties* area for the *Example_Sport.odb* database is for the *Data* tab to be visible; the *Content type* field to be set to **Table**; and the *Content* field to be set to **tblEntrants** (the name of the first table alphabetically). These default settings define the initial content of the Add field dialog.



Example_Sport.odb : Report1 - LibreOffice Base: Report Builder

Content type: Table

Content: tblEntrants

Analyze SQL command: Yes

Filter:

Report Output Format: ODF Text Document

Add field: tblEntrants

birth_date
first_name
gender
ID
last_name

Highlight the fields to insert into the selected section of the template, then click Insert or press Enter.

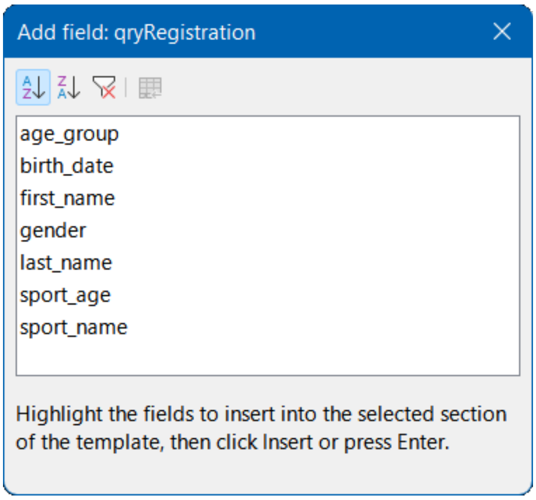
Figure 77: Initial display of Report Builder window

5) On the *Data* tab of the *Properties* area, for the *Content type* field select **Query**. The *qryRegistration* query is used as the basis for the new report.

Note

Making a selection in the *Content type* drop-down sometimes causes much of the Report Builder window to turn black. If this error occurs, a simple workaround is to resize the window using the control in the window’s title bar, and to then revert the window to its previous size. This action should cause Base to re-display the window in its normal color scheme.

6) On the *Data* tab of the *Properties* area, for the *Content* field select **qryRegistration**.
7) Base updates the content of the Add field dialog to list the fields of the selected query in alphanumeric order (Figure 78).



Add field: qryRegistration

age_group
birth_date
first_name
gender
last_name
sport_age
sport_name

Highlight the fields to insert into the selected section of the template, then click Insert or press Enter.

Figure 78: Add field dialog showing fields of qryRegistration query

8) Click the **Sorting and Grouping** icon on the Standard toolbar of the Report Builder window (Figure 79).

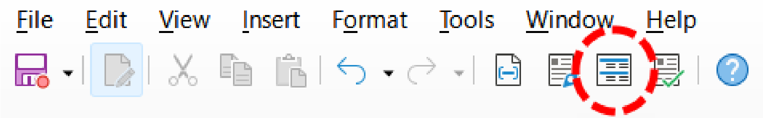
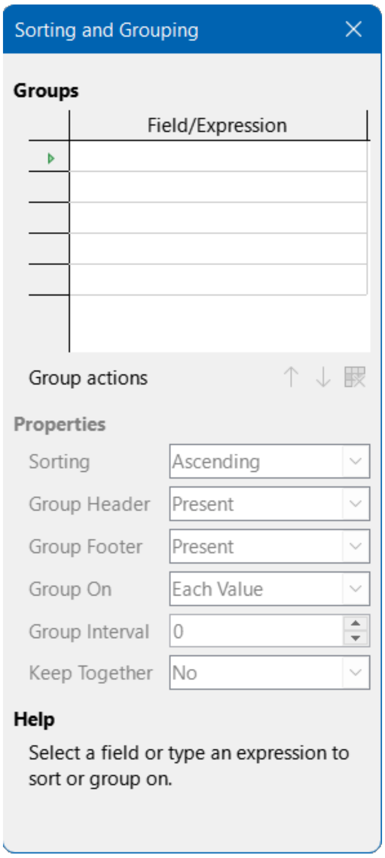


Figure 79: Location of Sorting and Grouping icon

9) Base displays the Sorting and Grouping dialog (Figure 80).



Sorting and Grouping

Groups

Field/Expression

Group actions

Properties

Sorting	Ascending
Group Header	Present
Group Footer	Present
Group On	Each Value
Group Interval	0
Keep Together	No

Help

Select a field or type an expression to sort or group on.

Figure 80: Sorting and Grouping dialog

10) In the Sorting and Grouping dialog, click in the first area below the *Field/Expression* label and select **sport_name** from the drop-down menu. Note that a *sport_name Header* area is added to the underlying window. Use the default settings in the Properties area of the Sorting and Grouping dialog.

11) Repeat step 10) but select *gender* from the drop-down menu in the next free area below the *Field/Expression* label.

12) Repeat step 10) but select *age_group* from the drop-down menu in the next free area below the *Field/Expression* label.

13) Close the Sorting and Grouping dialog.

14) On the Report Builder window, select the *sport_name Header* group section. Base highlights the selected area with a white border.

15) If the Add field dialog is not displayed, click the **Add Field** icon on the Standard toolbar of the Report Builder window (Figure 81).

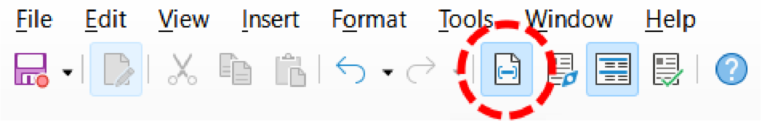


Figure 81: Location of Add Field icon

16) Base re-displays the Add Field dialog (Figure 78).

17) On the Add field dialog, select the *sport_name* entry and then click the **Insert** icon.

18) Base inserts a label field and a text field to display the contents of the *sport_name* field of the *qryRegistration* query in the highlighted area of the underlying report.

19) On the Report Builder window, select the *gender Header* group section. Base highlights the selected area with a white border.

20) On the Add field dialog, select the *gender* entry and then click the **Insert** icon.

21) Base inserts a label field and a text field to display the contents of the *gender* field of the *qryRegistration* query in the highlighted area of the underlying report.

22) On the Report Builder window, select the *age_group Header* group area. Base highlights the selected area with a white border.

23) On the Add field dialog, select the *age_group* entry and then click the **Insert** icon.

24) Base inserts a label field and a text field to display the contents of the *age_group* field of the *qryRegistration* query in the highlighted area of the underlying report.

25) On the Report Builder window, select the De*tail* group area. Base highlights the selected area with a white border.

26) On the Add field dialog, select the *first_name* entry and then click the **Insert** icon.

27) On the Add field dialog, select the *last_name* entry and then click the **Insert** icon.

28) On the Add field dialog, select the *birthday* entry and then click the **Insert** icon.

29) On the Add field dialog, select the *sport_age* entry and then click the **Insert** icon.

30) Base inserts pairs of label fields and text fields to display the contents of the *first_name*, *last_name, birth_date*, and *sport_age* fields of the *qryRegistration* query in the highlighted area of the underlying report.

31) Close the Add field dialog.

32) The draft report should now look similar to that shown in Figure 82.

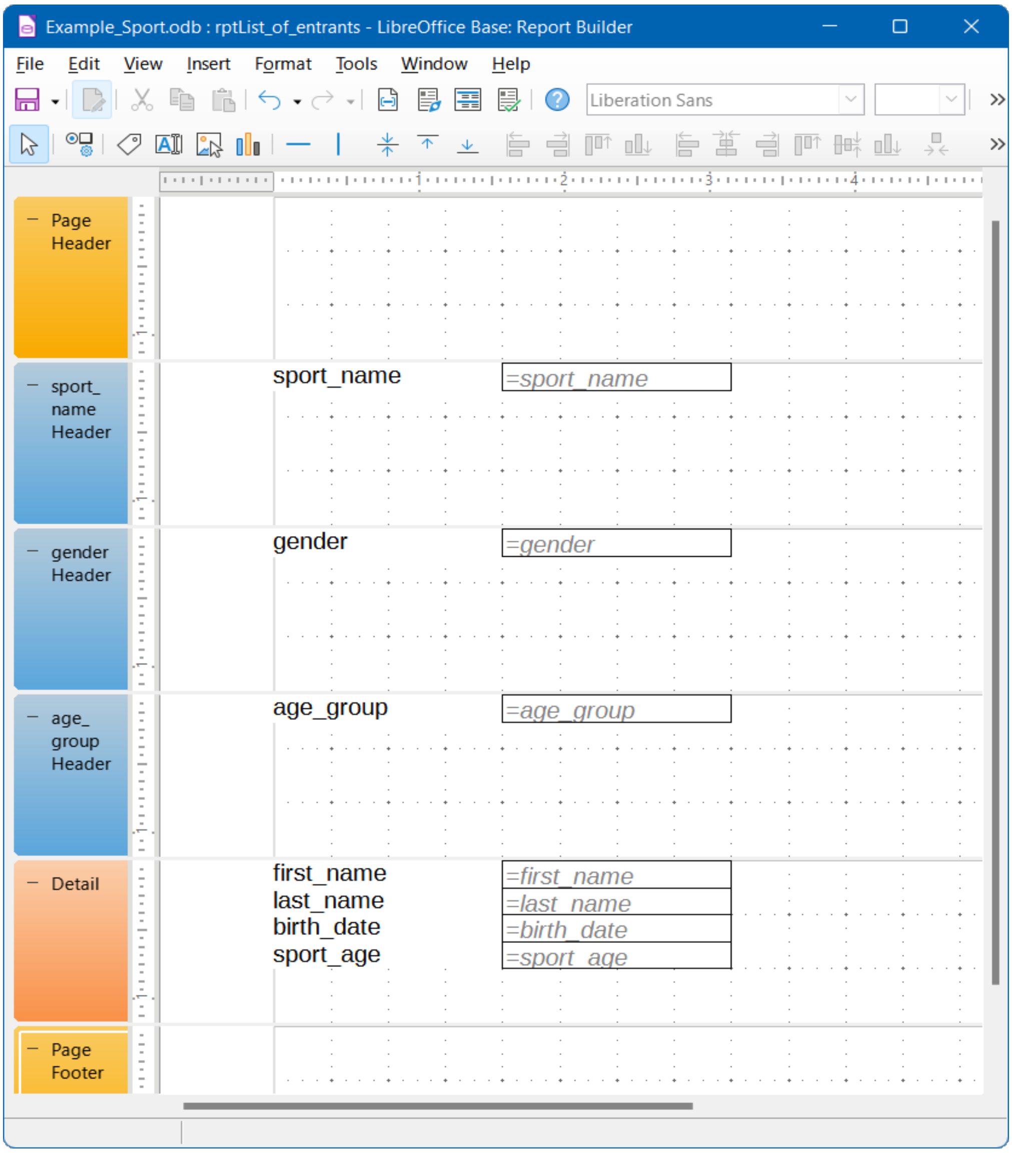


Figure 82: Draft report in Report Builder window

33) Save the report with the name *rptList_of_entrants*.

34) Save the database (otherwise, the report is stored only temporarily).

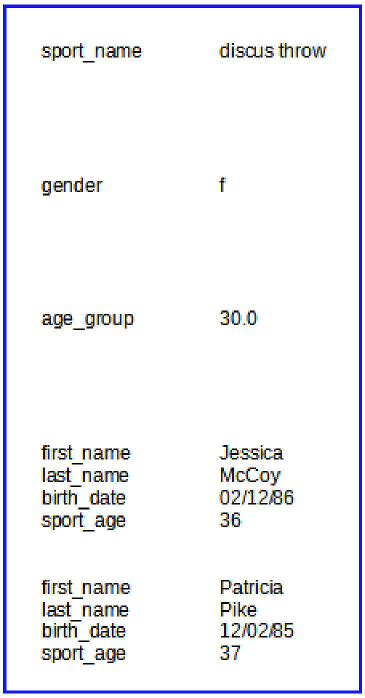
Note

When designing a report using the Report Builder, failures have been reported as instabilities of the program. Therefore it is important to save both the report and database file.

Fortunately, the later execution of a report is not usually affected by these issues.

To run this report, click **Reports** in the *Database* area of the main Base window and then double-click the name of the required report in the *Reports* area.

If this report is run with appropriate data, then the first part of the output should resemble that shown in Figure 83.



```
sport_name      discus throw


gender          f


age_group       30.0


first_name      Jessica
last_name       McCoy
birth_date      02/12/86
sport_age       36

first_name      Patricia
last_name       Pike
birth_date      12/02/85
sport_age       37
```

Figure 83: First part of draft of report output

The beginning of the report shows two female entrants, Jessica and Patricia, who have entered the discus throw event and are in the age group 30-39. Many other entrant / age group / sport combinations may follow depending on the data in the tables of the database.

When running the report for the first time, it is common to recognize some features that might be improved, such as:

- The distances between the group headers and before the content in the detail section are too large.
- The gender is indicated by the single letters *m* and *f*. Abbreviations aren’t necessary and full words would be better.
- The *age_group* field shows numbers to one decimal place and this granularity is unnecessary (presumably its appearance is the result of the mathematical operations carried out in the *qryRegistration* query).
- The data in the detail section for each sport / gender / age group combination (*first_name*, *last_name*, *birth_date*, and *age_group*) would be better laid out as a table.

The following subsections improve the report by addressing each of these issues in turn.

Setting the distances between the report fields

To adjust the height of a section of the report, use the mouse to drag the borders of that section up and down as required. You can also highlight a section header and specify the *Height* property in the *Properties* area on the right of the Report Builder window.

It is not possible for a section to be smaller than the labels and fields that it contains.

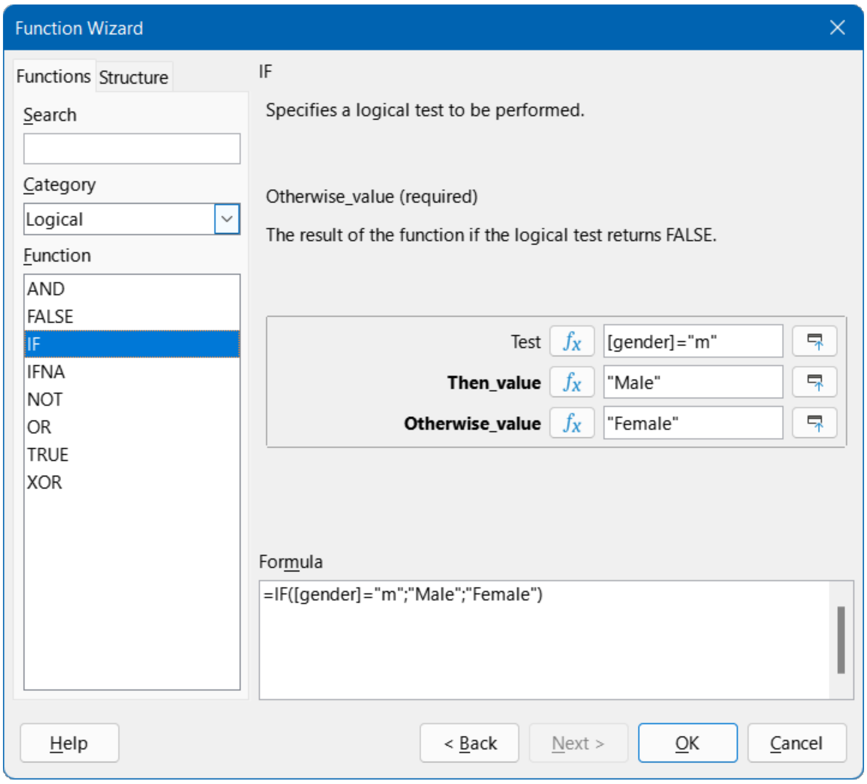
Of course, the detailed layout of any report is a matter of taste. Consider when selecting the distances that a following group does not appear too close below the previous group. The label and text boxes may need a distance to the top of the section containing them. If this distance to the top is not desired, group footers rather than headers can be displayed to provide some spacing distance. Such a preference is possible in the Sorting and Grouping dialog (Figure 80) for each group. For the *rptList_of_entrants* report, use the following suggestions:

- Set the height of the *Page Header* to 1 cm.
- Set the height of the *sport_name Header* to 1.5 cm.
- Set the height of the *gender Header* to 1 cm.
- Set the height of the *age_group Header* to 1.7 cm.
- Do not try to change the height of the *Detail* area at the moment.
- Set the height of the *Page Footer* to 1.3 cm.

Influencing a text field content by a formula

The designation of the gender in the table is not enough for the *rptList_of_entrants* report. Renaming the field could be done in the query. However, because the query has already been created, you can use functions in the report instead.

1) Highlight the text field = *gender* in the *gender Header* section of the report.
2) In the Properties area on the right side of the Report Builder window, click on the *Data* tab.
3) Make sure that the *Data Field Type* drop-down menu is set to **Field or Formula**.
4) Click the ellipsis (…) button to the right of the *Data* field.
5) Base displays the Function Wizard.
6) Select **Logical** in the *Category* drop-down menu.
7) Double-click the **IF** entry in the *Function* area. Note that once details of the IF function are selected on the right side of the Function Wizard, the contents of the *Category* and *Function* areas may change – this is a predictive behavior that should be ignored at this stage.
8) Enter `[gender]=“m”` in the *Test* field. This defines the test to be performed. The name of a field of the query from which the data is read is surrounded by square brackets. Text strings are enclosed in double quotes.
9) Enter `“Male”` in the T*hen_value* field. If the field is set to “m”, then replace it with the string “Male”.
10) Enter `“Female”` in the *Otherwise_value* field (Figure 84). If the field is not set to “m”, then replace it with the string “Female”.



Category: Logical

IF

Specifies a logical test to be performed.

Otherwise_value (required)

The result of the function if the logical test returns FALSE.

Test	[gender]="m"
Then_value	"Male"
Otherwise_value	"Female"

```
=IF([gender]="m";"Male";"Female")
```

Figure 84: Using the Function Wizard to change the gender string

11) Click **OK** to confirm the entry.

Next time you run the report, the wording of the *gender* field is changed accordingly.

Change the formatting of a text field

Fields that display the contents of the database are initially identified in the report as text boxes, but they can be formatted just like the fields in tables in Writer or Calc.

1) Highlight the field *=age_group*.
2) In the *Properties* area on the right of the Report Builder window, click on the *General* tab.
3) Notice that the *Formatting* field is initially set to **Text**. Click the button with the ellipsis (…) to the right of the *Formatting* field
4) Base opens the Format Number dialog. Similar dialogs are used in Calc, Writer, or when creating forms.
5) In the *Category* area select the **Number** option.
6) Make sure that the **General** option is selected in the *Format* area (Figure 85).

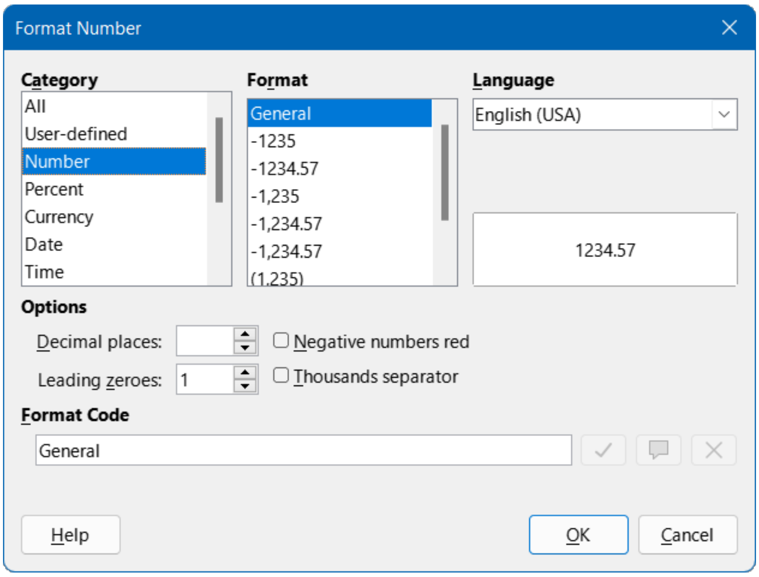


Figure 85: Format Number dialog

7) Click the **OK** button.

Next time you run the report, the format of the age group field is now displayed without decimal places.

Moving boxes in the Report Builder window

This subsection describes how to tidy up the report a little, moving fields to more convenient positions.

Report fields can be moved beyond the boundaries of a section into another section. However, sufficient space for the field must be present in the destination section. No part of one field can exist in the same location as any part of another field.

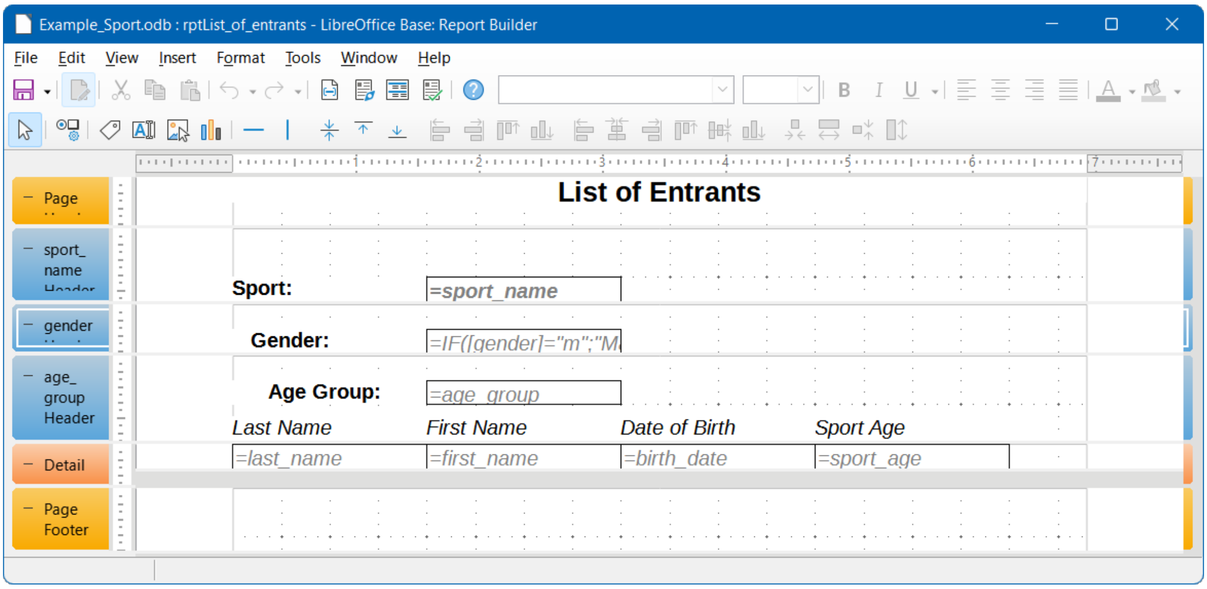
Positioning of fields using a mouse is inaccurate, so you need to provide ample space in each section for the fields you intend to place there. After dragging a field to a new position using the mouse, more precise positioning can be achieved using the arrow keys. Alternatively, to position fields using the keyboard, modify the *Position X* and *Position Y* values on the *General* tab of the *Properties* area.

A label is added to the *Page Header* area by selecting **Insert > Report Controls > Label** on the Menu bar. The text displayed on the label is defined in the *Label* field of the *Properties* area and the font can be chosen in the *Font* field of the same area.

The four labels initially located in the *Detail* area can be moved to form a row at the bottom of the *age_group Header* area. The four text areas in the *Detail* area may now be rearranged in a row, each underneath its corresponding label. On completion, the height of the *Detail* area can be reduced to 0.5 cm.

To hide the *Page Footer* section completely, set *Visible* to **No** in the **Properties** area. The bottom margin of the document contains too much space. Remember that the available amount is already reduced by the size of the page margins.

Your improved report might be structured in a similar way to that shown in Figure 86

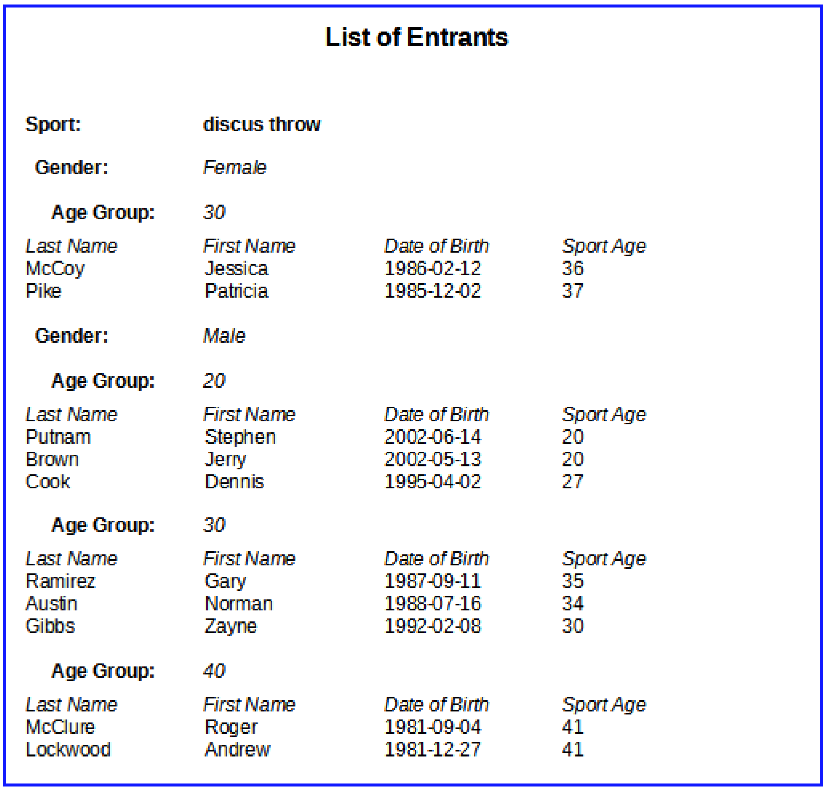


```
                         List of Entrants

Sport:            =sport_name
 Gender:          =IF([gender]="m";"M
  Age Group:      =age_group
Last Name         First Name        Date of Birth     Sport Age
=last_name        =first_name       =birth_date       =sport_age
```

Figure 86: Updated rptList_of_entrants report

As can be seen in Figure 87, these changes have improved the layout of the report’s output.



```
                    List of Entrants


Sport:          discus throw

 Gender:        Female

   Age Group:   30

Last Name       First Name      Date of Birth   Sport Age
McCoy           Jessica         1986-02-12      36
Pike            Patricia        1985-12-02      37

 Gender:        Male

   Age Group:   20

Last Name       First Name      Date of Birth   Sport Age
Putnam          Stephen         2002-06-14      20
Brown           Jerry           2002-05-13      20
Cook            Dennis          1995-04-02      27

   Age Group:   30

Last Name       First Name      Date of Birth   Sport Age
Ramirez         Gary            1987-09-11      35
Austin          Norman          1988-07-16      34
Gibbs           Zayne           1992-02-08      30

   Age Group:   40

Last Name       First Name      Date of Birth   Sport Age
McClure         Roger           1981-09-04      41
Lockwood        Andrew          1981-12-27      41
```

Figure 87: First few lines of updated output from rptList_of_entrants

For more information about formatting options for reports, see Chapter 6, Reports.

Extensions to the Example_Sport.odb database

The example database presented here is only the first step for a database to be used in the sports sector. Many useful extension could be suggested for the database:

- Extra fields could be added for items of extra information that might be needed. A good place to add such fields is in the *tblRel_entrant_sport* table.

- If running several competitions for the same sport, you might include in the *tblRel_entrant_sport* table a date field or another field that can be assigned to a specific competition. The field then becomes part of the table's primary key.
- Perhaps the club that an entrant is a member of could also be added. Adding a field in the *tblEntrants* table would be sufficient. If multiple clubs could have the same name, this suggests adding a separate *tblClub* table and also the corresponding foreign key in the *tblEntrants* table.
- As with all competitions, it must be determined who should be placed in which age group and sport. Sorting is required here, which might again end up as a report with a results list.
- It would be useful to design a report that could be used to give each entrant an attractively designed certificate with their personal performance and placement.

These and many other extensions are possible, using the methods described in later chapters of this guide.

Base Guide 7.3

Chapter 2
Creating a Database

Introduction

Chapter 1, Introducing Base, describes how Base provides a graphical user interface for databases.

Integrated into Base are two embedded relational database engines, HSQLDB (HyperSQL DataBase) version 1.8.0 and Firebird version 3.0.7, with the latter currently accessible only as an experimental feature. Databases created using an embedded engine are accessible by only one user at a time, with all the data for the database stored in an ODB file. The basics of creating an embedded database are described in Chapter 8 of the *Getting Started Guide*, Getting Started with Base, and in Chapter 1 of this book. The database examples in this book refer to HSQLDB but are customized so that most functions are directly transferable to Firebird. Where appropriate, alternatives in Firebird are identified.

Base also supports connection to many types of external databases. These may be hosted either locally or at a remote network location and may support multiple concurrent users. Before Base can make a connection to an external database, the database must exist and, where appropriate, be configured to allow network connection with specific login credentials.

This chapter describes how to create a new (embedded) database using the embedded HSQLDB engine, how to create connections to several different types of external databases, and how to later modify those connections when necessary.

Note

The embedded HSQLDB engine is written in Java. When an embedded HSQLDB database contains a large amount of data, the use of Java may sometimes lead to a loss of speed. This factor is one of the motivations for the on-going work to convert the default embedded database engine from HSQLDB to Firebird.

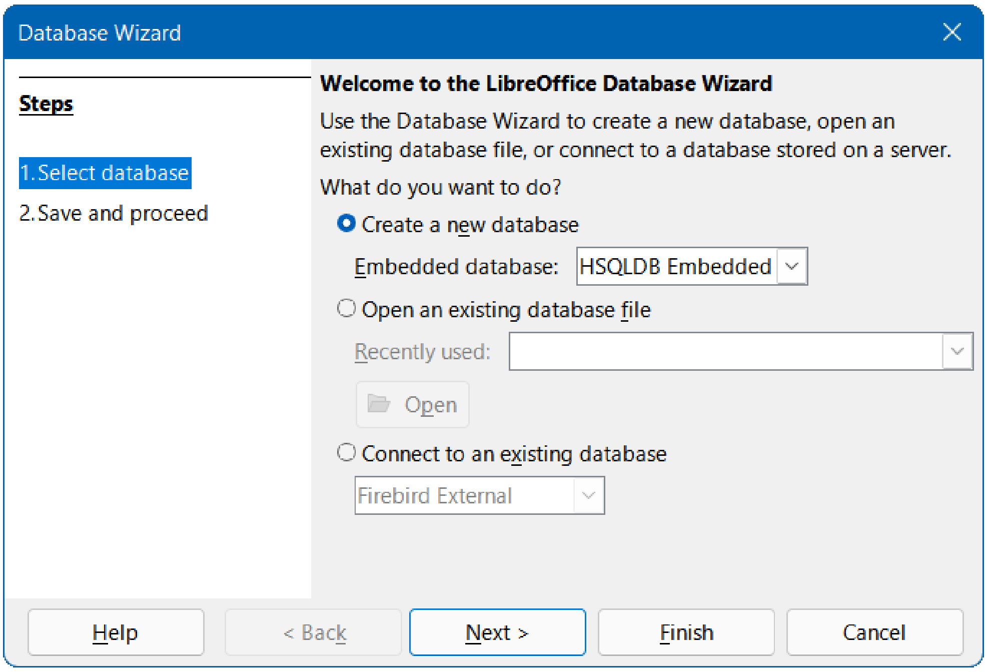
Similar effects may be observed when accessing external databases that do not use Java, depending on the driver used. When the connection is via JDBC, which is a Java-based technology, performance can be adversely affected.

Creating a database using the embedded database engine

If you are planning a single-user database, or if you simply intend to gain some initial experience with a database, the embedded HSQLDB engine will meet your needs. If later your objectives change, then it is possible to transfer the database to an external server, where performance may be better and multiple users can have concurrent access to the database.

To create an embedded database, carry out the following steps.

1) Access the Database Wizard by performing one of the following actions:
   a) On the LibreOffice Start Center, click the **Base Database** option in the *Create* area.
   b) In any LibreOffice component, select **File > New > Database** on the Menu bar.
   c) In Base, click the **New** icon on the Standard toolbar or use the equivalent keyboard shortcut *Ctrl+N*.
   d) In any LibreOffice component, click the down arrow to the right of the **New** icon on the Standard toolbar and select **Database** in the drop-down menu.
2) Base displays Step 1 (*Select database*) of the Database Wizard (Figure 88).



Welcome to the LibreOffice Database Wizard

Use the Database Wizard to create a new database, open an existing database file, or connect to a database stored on a server.

What do you want to do?

● Create a new database

Embedded database: HSQLDB Embedded

○ Open an existing database file

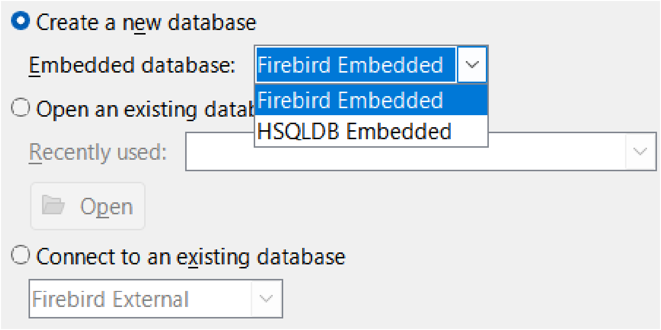
Recently used:

○ Connect to an existing database

Firebird External

Figure 88: Step 1 of Database Wizard: Select database

3) Click the **Create a new database** option. By default (with experimental features disabled), Base will create an HSQLDB database and this is reflected in the setting of the *Embedded database* drop-down menu, which contains no other options. However, if the **Enable experimental features (may be unstable)** option is selected on the **Tools > Options > LibreOffice > Advanced** dialog, the behavior of the Database Wizard changes so that the *Embedded database* menu contains two options (Figure 89), with the default being **Firebird Embedded**.



● Create a new database

Embedded database: Firebird Embedded

○ Open an existing datal

Recently used:

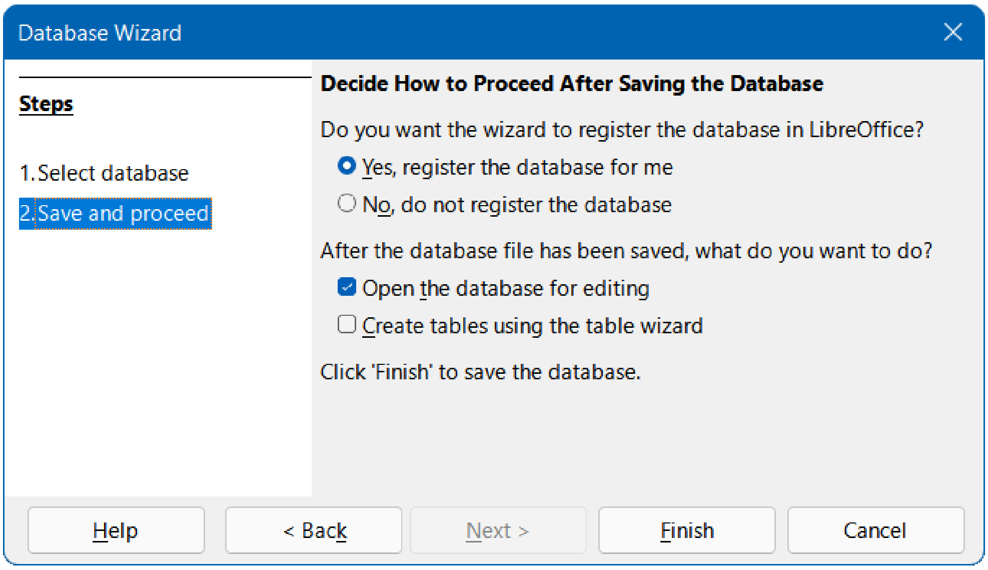
○ Connect to an existing database

Firebird External

Figure 89: Embedded database menu with experimental features enabled

4) Click the **Next >** button.

5) Base displays Step 2 (*Save and proceed*) of the Database Wizard (Figure 90).



Database Wizard

Steps

1. Select database

2. Save and proceed

Decide How to Proceed After Saving the Database

Do you want the wizard to register the database in LibreOffice?

● Yes, register the database for me

○ No, do not register the database

After the database file has been saved, what do you want to do?

☑ Open the database for editing

□ Create tables using the table wizard

Click 'Finish' to save the database.

Figure 90: Step 2 of Database Wizard: Save and proceed

6) By default, the **Yes, register the database for me** option is selected. A registered database can be used by other components of LibreOffice, for example for mail merge letters in Writer. It is recommended that all databases be registered on creation, but the final choice belongs to the user.

7) By default, the **Open the database for editing** option is selected, as normally this will be appropriate for a new database. You may deselect the option if you do not wish to edit the database at this time.

8) By default, the **Create tables using the table wizard** option is deselected. Use of the Table Wizard was described in Chapter 8 of the *Getting Started Guide*, Getting Started with Base. The remainder of this book intentionally avoids describing the use of wizards for creating tables, queries, and so on. However, the user can choose to select this option as required.

9) Click the **Finish** button to save the new database.

10) Base displays the Save As dialog, where you should specify a file name and location for the ODB file.

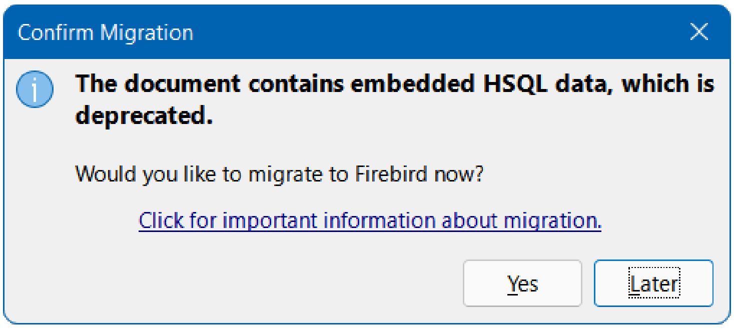
11) Click the **Save** button.

The embedded database is ready for the entry of records, and the storage of views, queries, forms, and reports. Note that the file is saved before you have made any data entries, which is a different approach to that taken in other LibreOffice components.

Caution

If experimental features have been enabled by selecting **Enable experimental features (may be unstable)** on the **Tools > Options > LibreOffice > Advanced** dialog and you later open an existing embedded HSQLDB database, then Base may display the Confirm Migration dialog (Figure 91) asking if you wish to migrate the database to Firebird. The link on this dialog points to the following page on The Document Foundation’s wiki: https://wiki.documentfoundation.org/Documentation/HowTo/MigrateFromHSQLDB.

This experimental automatic migration function is started whenever an embedded HSQLDB database is opened and the first connection to the data schema is created. If you are faced with this dialog, extreme caution is advised because work continues on the development of this feature. Do not simply confirm by clicking **Yes**; be sure to click **Later**. Make sure that you carry out the preparations described on the referenced wiki page before migration is attempted.



Confirm Migration

The document contains embedded HSQL data, which is deprecated.

Would you like to migrate to Firebird now?

Click for important information about migration.

Figure 91: Confirm Migration dialog

Accessing external databases

Note

When using Base to make a connection over the Internet, make sure that you are aware of how that connection is structured. Any database accessed over the Internet should be protected by a specific user name and password. Ask questions such as: Is the connection encrypted / secure? How is the password transferred? Is my data safe?

Introduction

An external database must exist before it can be accessed using Base. If access to an external database is desired, it must be set up to allow network connections with specific login credentials before any other software (such as Base) can connect to it.

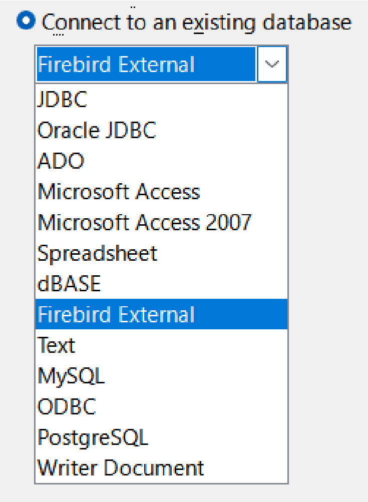
When an external database is appropriately set up, you may, depending on the type of database and the available driver software, create tables, input data, and query data. Some external data sources (for example, a connection to a spreadsheet or a text database) do not allow new data to be entered and these are used only to search for or report on existing data.

To connect to an external database, start with the following steps.

1) Access the Database Wizard by performing one of the actions listed in step 1) of the procedure for creating a new database using the embedded database engine (page 81).
2) At Step 1 (*Select database*) of the Database Wizard (Figure 88), click the **Connect to an existing database** option. The database types listed in the adjacent drop-down menu vary according to operating system and other factors, but the following should always be available:
   - dBASE
   - Firebird
   - JDBC
   - MySQL
   - ODBC
   - Oracle JDBC

- PostgreSQL
- Spreadsheet
- Text
- Writer document.

Figure 92 is an example showing the options available for a Windows-based system.



● Connect to an existing database

Firebird External

JDBC
Oracle JDBC
ADO
Microsoft Access
Microsoft Access 2007
Spreadsheet
dBASE
Firebird External
Text
MySQL
ODBC
PostgreSQL
Writer Document

Figure 92: External database connection options on a Windows platform

3) Select the entry for the connection type required and then click the **Next >** button.

The number of steps remaining and the choices provided at each step depend on the nature of the connection type selected. You can change the connection settings after the associated ODB file has been created.

The descriptions in subsequent chapters of this book primarily describe how Base is used in conjunction with an embedded HSQLDB database. Most of the database design work can be extended to databases that use Firebird, MySQL, PostgreSQL, and so on.

The remainder of this chapter provides examples of how to establish connections to typical external databases. Some users may wish to use Base to connect to an existing multi-user database and should consult their organization’s database administrator in order to gain access to the RDBMS. Other users may wish to install a RDBMS for single-user access on their own computer and then use Base as the front-end. It is beyond the scope of this document to cover all possible use cases.

✓ **Note**

Depending on the driver used, it may be possible to access several databases simultaneously on a connected server. For example, if a connection to a MySQL/MariaDB database is established, other databases on the server for which the specified user has access rights may be displayed in Base at the same time. It is therefore possible to copy data from one database to another, perform related queries, and so on.

MySQL/MariaDB databases

Overview

MySQL is an open-source relational database management system (RDBMS) that is provided by Oracle Corporation. More information can be found on the MySQL website at https://www.mysql.com/. The current version of the MySQL Reference Manual is accessed through the *Documentation* link on the MySQL website.

MariaDB is an open-source RDBMS that is compatible with MySQL. The development of MariaDB is overseen by the MariaDB Foundation and more information can be found at https://mariadb.com/. The MariaDB Server Documentation can be found at https://mariadb.com/kb/en/documentation/.

Base can connect to MySQL and MariaDB databases using one of three methods:

- Direct connection using a built-in connector. This is the simplest and fastest method.
- Connection using ODBC (Open Database Connectivity).
- Connection using JDBC (Java Database Connectivity).

Initial Preparation

Refer to the MySQL Reference Manual or MariaDB Server Documentation as appropriate for detailed instructions about downloading and installing the database server software for your computer configuration, and for details of how to access a SQL console. After installation, follow any setup instructions specified in the documentation. These may include:

1) In both MySQL and MariaDB the user name of the administrator account is *root* (note that the name root in this context is not related to the root superuser account that is provided for some operating systems). Make sure that the password for the *root* account of the database server has been set to a string that is appropriate to maintain the security of your data.
2) Some user accounts may be provided by default, for test purposes, and some test databases may allow anonymous access. As an extra security measure, make sure that all anonymous user accounts are deleted.

Before making a connection, the database of interest must exist on the server. If your database does not yet exist, carry out the following steps to create it.

1) To create an (empty) database called *libretest*, type the following command at a SQL console:

```
CREATE DATABASE libretest;
```

2) To grant all rights on the new database to a new user called *lotest* with the password *libre*, enter the following commands at a SQL console:

```
CREATE USER 'lotest' IDENTIFIED BY 'libre';
GRANT ALL PRIVILEGES ON libretest.* TO 'lotest';
FLUSH PRIVILEGES;
```

On successful completion of these steps, the new database is available and Base can connect to it, as described in the following subsections.

Connecting directly

Note

In earlier versions of Base (prior to LibreOffice 6.2), direct connection to MySQL or MariaDB required installation of an extension. Since LibreOffice 6.2, the software needed to make a direct connection is integrated within Base.

The first time that you want to make a direct connection to a particular MySQL or MariaDB database, carry out the following steps.

1) Access the Database Wizard by performing one of the actions listed in step 1) of the procedure for creating a new database using the embedded database engine (page 81).

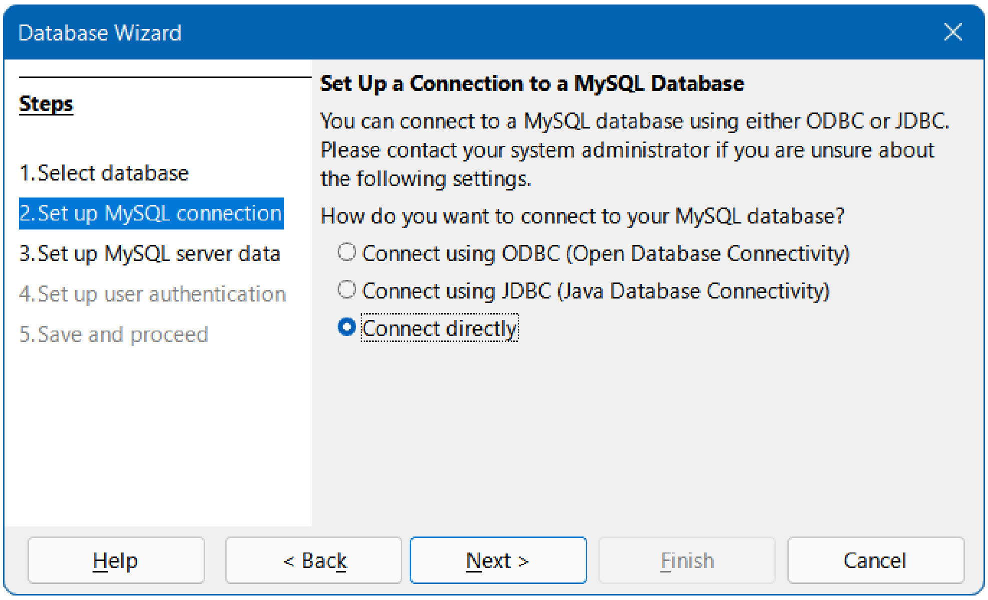
2) Base displays Step 1 (*Select database*) of the Database Wizard (Figure 88).

3) Click the **Connect to an existing database** option. Base enables the adjacent drop-down menu listing the available options (Figure 92).

4) Select **MySQL** in the drop-down menu.

5) Click the **Next >** button.

6) Base displays Step 2 (*Set up MySQL connection*) of the Database Wizard (Figure 93).



Set Up a Connection to a MySQL Database

You can connect to a MySQL database using either ODBC or JDBC. Please contact your system administrator if you are unsure about the following settings.

How do you want to connect to your MySQL database?

○ Connect using ODBC (Open Database Connectivity)

○ Connect using JDBC (Java Database Connectivity)

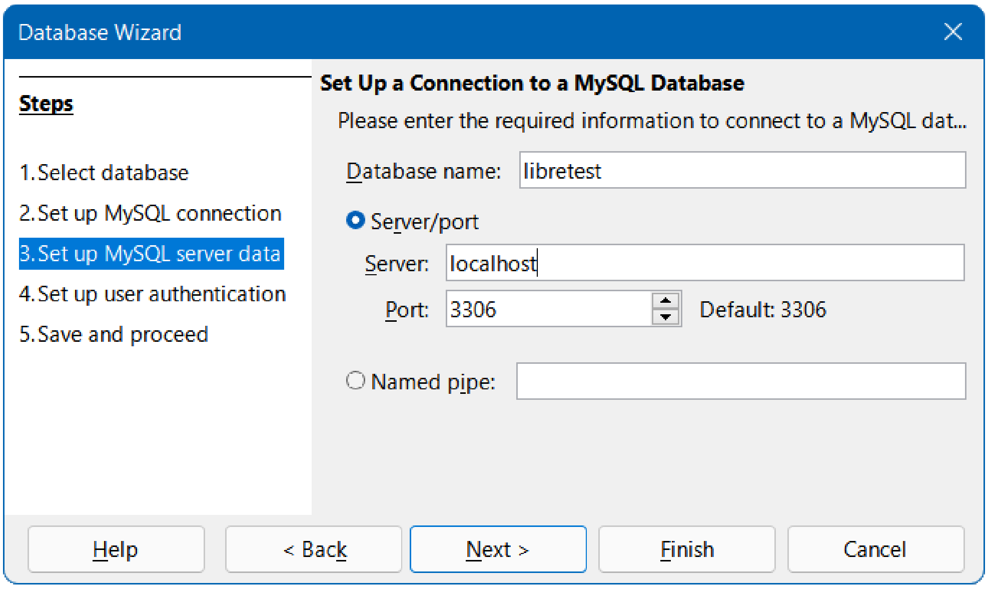
● Connect directly

Figure 93: Step 2 of Database Wizard: Set up MySQL connection

7) Click the **Connect directly** option.

8) Click the **Next >** button.

9) Base displays Step 3 (*Set up MySQL server data*) of the Database Wizard (Figure 94).



Set Up a Connection to a MySQL Database

Please enter the required information to connect to a MySQL dat...

Database name: libretest

● Server/port

Server: localhost

Port: 3306 Default: 3306

○ Named pipe:

Figure 94: Step 3 of Database Wizard: Set up MySQL server data

10) Enter the name of your database in the *Database name* field. Our example database is called *libretest*.

11) Click the **Server/port** option.

12) Enter the name of the server that runs the MySQL or MariaDB database in the *Server* field. If you are connecting to a database on the same computer, the server name should be *localhost*. If you are connecting to a remote server, then it may be an IP address, a network computer name, or an Internet address.

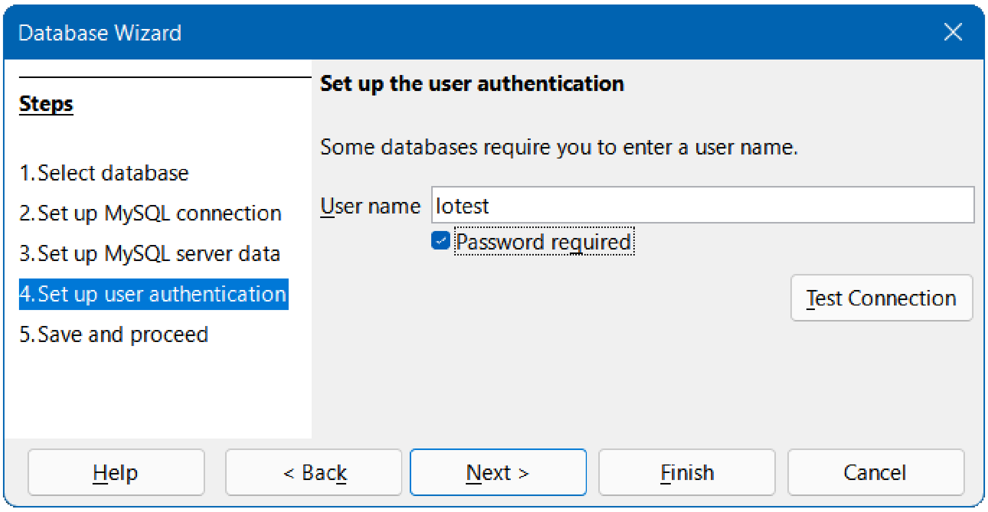
13) Enter the port number for the database server in the *Port* field. The default port number for MySQL and MariaDB databases is 3306.

Tip

At steps 10) to 13), if you are unsure about the correct information to be entered, seek guidance from your database administrator.

14) Click the **Next >** button.

15) Base displays Step 4 (*Set up user authentication*) of the Database Wizard (Figure 95).



Set up the user authentication

Some databases require you to enter a user name.

User name: lotest

☑ Password required

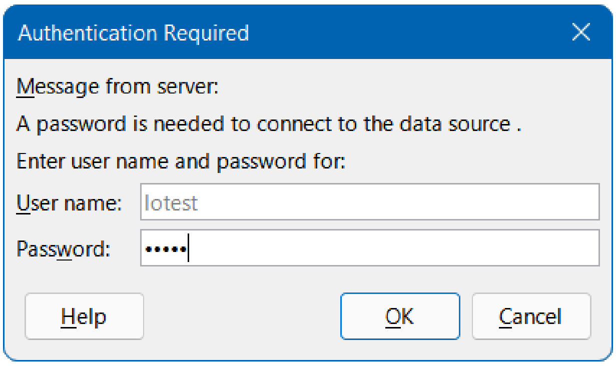
Figure 95: Step 4 of Database Wizard: Set up user authentication

16) Enter a valid user name in the *User name* field. Our example user is named *lotest*.

17) Click the **Password required** option.

18) Click the **Test Connection** button.

19) Base displays the Authentication Required dialog (Figure 96).



Message from server:

A password is needed to connect to the data source .

Enter user name and password for:

User name: lotest

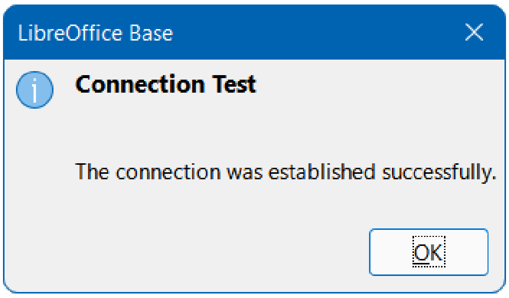
Password: •••••

Figure 96: Authentication Required dialog

20) Enter the user name (*lotest*) and password (*libre*) required to access the database.

21) Click the **OK** button.

22) Base displays a dialog to inform that the connection test was successful (Figure 97). If the test was unsuccessful (for example, the database server is not currently running or you typed incorrect login credentials), an appropriate error message is displayed instead.



LibreOffice Base

Connection Test

The connection was established successfully.

Figure 97: Connection test successful dialog

23) Press the **OK** button to remove the connection test successful dialog from the screen.

24) Click the **Next >** button.

25) Base displays Step 5 (*Save and proceed*) of the Database Wizard (with the same options as are shown in Figure 90).

26) Click the **No, do not register the database** option, as this is a test database.

27) Click the **Open the database for editing** option, as normally this will be appropriate for a new database. You may deselect the option if you do not wish to edit the database at this time.

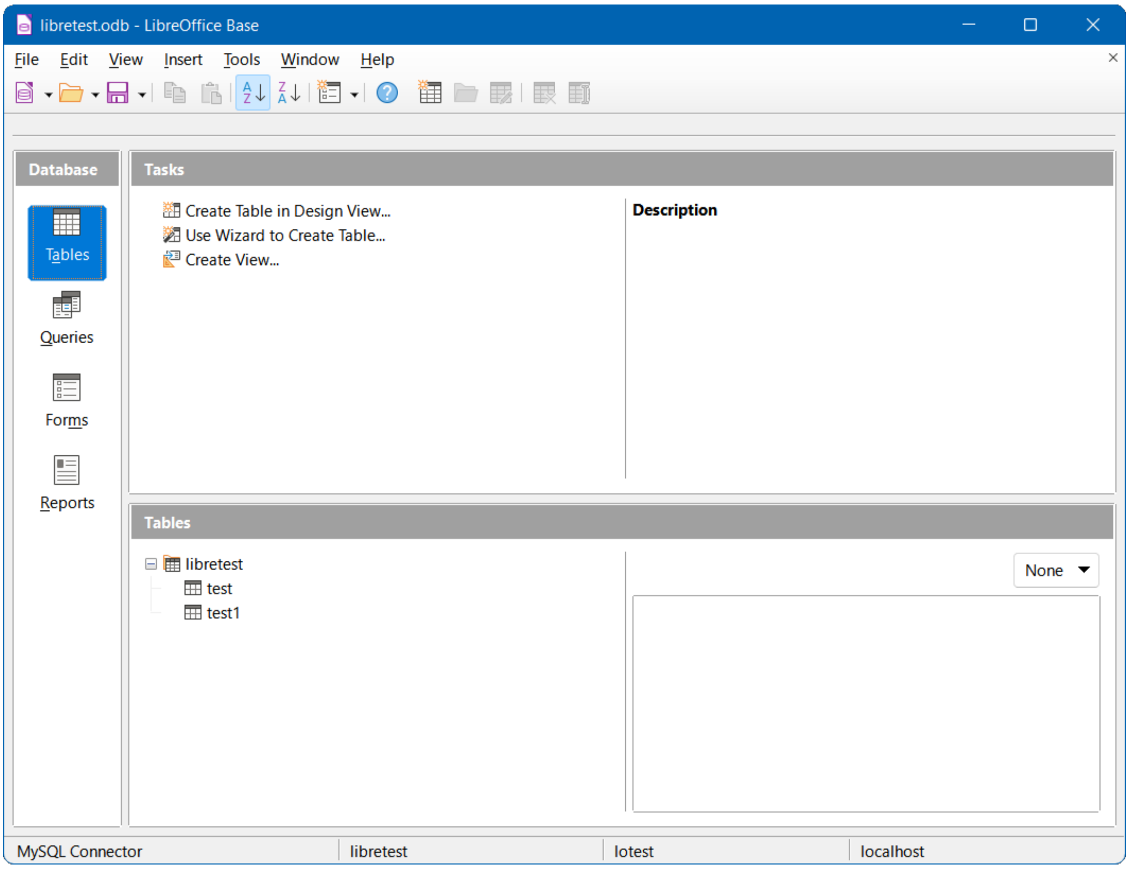
28) Unless you wish to use the Table Wizard, leave the **Create tables using the table wizard** option in its default deselected state.

29) Click the **Finish** button.

30) Base displays the Save As dialog, where you should specify a file name and location for the ODB file.

31) Click the **Save** button.

32) The Database Wizard is removed from the screen, the ODB file containing the database connection information is created, and Base updates its main window to show information about the database tables under the name of the database itself (Figure 98).


libretest.odb - LibreOffice Base
File Edit View Insert Tools Window Help
Database
Tables
Queries
Forms
Reports
Tasks
Description
Tables
None

Create Table in Design View...
Use Wizard to Create Table...
Create View...

libretest
test
test1

MySQL Connector | libretest | lotest | localhost

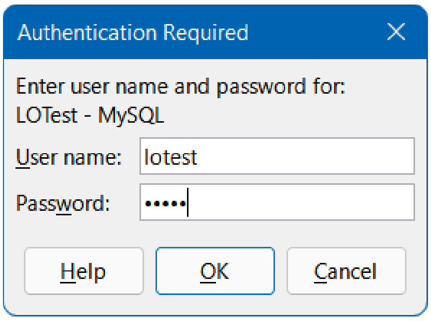
Figure 98: Main Base window showing tables within libretest database

Tables can be created, edited, and deleted.

The Status bar at the bottom of Figure 98 shows:

- The name of the connector used to connect to the MySQL or MariaDB database (*MySQL Connector* in this case).
- The name of the open MySQL or MariaDB database. In this case the ODB file and the MySQL database share a common name (*libretest*), but this need not be the case.
- The user name that is being used to connect to the MySQL or MariaDB database (*lotest* in this case).
- The name of the server on which the MySQL or MariaDB database is running (*localhost* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying MySQL or MariaDB database, the Authentication Required dialog (Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.



Enter user name and password for:
LOTest - MySQL

User name: lotest

Password: •••••

Figure 99: Authentication Required dialog

Connecting with ODBC

Initial preparation

To use Base to connect to a MySQL or MariaDB database server through the Open Database Connectivity (ODBC) Application Programming Interface (API), there should be a properly configured ODBC driver installed on your computer. If the database server is already installed on your computer, then the ODBC connector may already be installed as well. If the connector is not currently available on your computer, then you will need administration rights to install the new software.

To connect to a MySQL database through ODBC, it is recommended that you use the MySQL Connector/ODBC. The correct version of the ODBC connector for your computer can be downloaded from the MySQL Community Downloads website at https://dev.mysql.com/downloads/connector/odbc/. Platform-dependent installation instructions can be found in the MySQL Connector/ODBC Developer Guide at https://dev.mysql.com/doc/connector-odbc/en/.

To connect to a MariaDB database through ODBC, it is recommended that you use the MariaDB Connector/ODBC. The correct version of the ODBC connector for your computer can be downloaded from the Downloads area of the MariaDB website at https://mariadb.com/downloads/connectors/connectors-data-access/odbc-connector. Platform-dependent installation instructions can be found in the MariaDB Knowledge Base at https://mariadb.com/kb/en/about-mariadb-connector-odbc/.

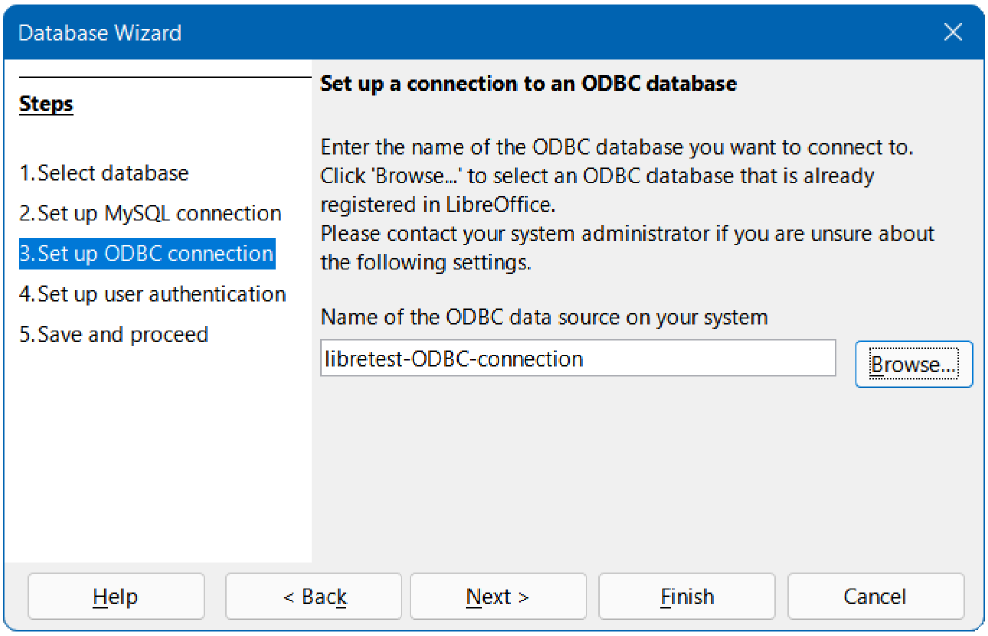
Make sure that you follow the instructions given in the connector documentation to create an ODBC Data Source Name (DSN) for your database. For the purposes of the example in the next subsection, assume that a data source has been configured as follows:

- The MySQL ODBC Unicode driver is selected.
- The name of the data source is set to *libretest-ODBC-connection*.
- The database name (*libretest*) is included within the definition of the data source.
- Login credentials for the database server are not defined within the data source.

Using the Database Wizard

The first time that you want to make an ODBC connection to a particular MySQL or MariaDB database, carry out the following steps.

1) Access the Database Wizard by performing one of the actions listed in step 1) of the procedure for creating a new database using the embedded database engine (page 81).
2) Base displays Step 1 (*Select database*) of the Database Wizard (Figure 88).
3) Click the **Connect to an existing database** option. Base enables the adjacent drop-down menu listing the available options (Figure 92).
4) Select **MySQL** in the drop-down menu.
5) Click the **Next >** button.
6) Base displays Step 2 (*Set up MySQL connection*) of the Database Wizard (Figure 93).
7) Click the **Connect using ODBC (Open Database Connectivity)** option.
8) Click the **Next >** button.
9) Base displays Step 3 (*Set up ODBC connection*) of the Database Wizard (Figure 100).



Database Wizard

Steps

1. Select database
2. Set up MySQL connection
3. Set up ODBC connection
4. Set up user authentication
5. Save and proceed

Set up a connection to an ODBC database

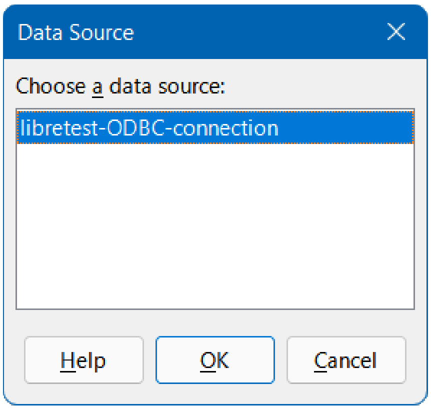
Enter the name of the ODBC database you want to connect to. Click 'Browse...' to select an ODBC database that is already registered in LibreOffice.
Please contact your system administrator if you are unsure about the following settings.

Name of the ODBC data source on your system

libretest-ODBC-connection

Figure 100: Step 3 of Database Wizard: Set up ODBC connection

10) Enter the name of the previously defined ODBC data source that is to be used to access the MySQL or MariaDB database (*libretest-ODBC-connection* in our example). You can click the **Browse** button to access the Data Source dialog (Figure 101), where you can select the data source of interest from a list of those available and then click the **OK** button.



Data Source

Choose a data source:

libretest-ODBC-connection

Figure 101: Data Source dialog

11) Click the **Next >** button.

12) Base displays Step 4 (*Set up user authentication*) of the Database Wizard (similar to Figure 95).

13) Enter a valid user name in the *User name* field. Our example user is name *lotest*.

14) Click the **Password required** option.

15) If required, test the connection by carrying out steps 18) to 23) of the direct connection procedure (page 88).

16) Click the **Next >** button.

17) Base displays Step 5 (*Save and proceed*) of the Database Wizard (with the same options as are shown in Figure 90).

18) Click the **No, do not register the database** option, as this is a test database.

19) Click the **Open the database for editing** option, as normally this will be appropriate for a new database. You may deselect the option if you do not wish to edit the database at this time.

20) Unless you wish to use the Table Wizard, leave the **Create tables using the table wizard** option in its default deselected state.

21) Click the **Finish** button.

22) Base displays the Save As dialog, where you should specify a file name and location for the ODB file.

23) Click the **Save** button.

24) The Database Wizard is removed from the screen, the ODB file containing the database connection information is created, and Base updates its main window to show information about the database tables under the name of the database itself (similar to Figure 98).

Tables can be created, edited, and deleted.

In this case, the Status bar shows the following information:

- The name of the connector used to connect to the MySQL or MariaDB database (for example *MySQL (ODBC)*).
- The name of the ODBC data source, in this case *libretest-ODBC-connection*.
- The user name that is being used to connect to the MySQL or MariaDB database (*lotest* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying MySQL or MariaDB database, the Authentication Required dialog (Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.

Connecting with JDBC

Initial preparation

To use Base to connect to a MySQL or MariaDB database server through the Java Database Connectivity (JDBC) Application Programming Interface (API), there should be an appropriate JDBC connector located on your computer. If the database server is already installed on your computer, then the JDBC connector may already be present as well. If the connector is not currently available on your computer, then you will need to download it and place a specific JAR (Java ARchive) file in an accessible folder.

To connect to a MySQL database through JDBC, it is recommended that you use the MySQL Connector/J. The correct version of the JDBC connector for your computer can be downloaded from the MySQL Community Downloads website at https://dev.mysql.com/downloads/connector/j/. Installation instructions can be found in the MySQL Connector/J Developer Guide, which is accessible from https://dev.mysql.com/doc/index-connectors.html.

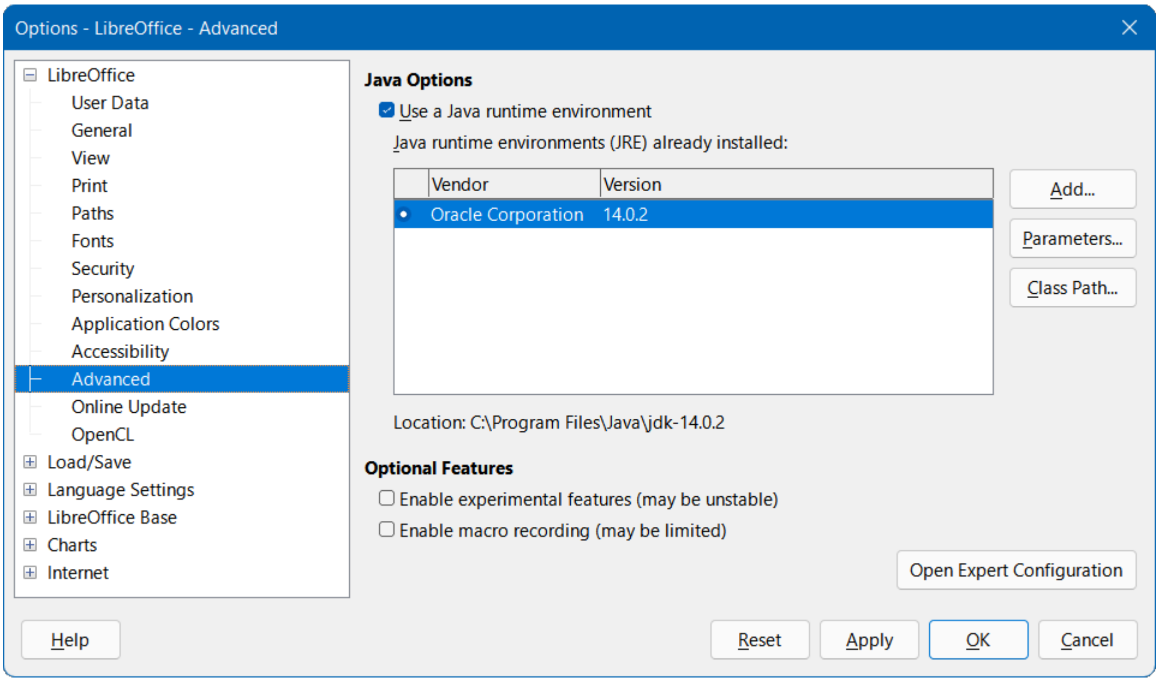
To connect to a MariaDB database through JDBC, it is recommended that you use the MariaDB Connector/J. The correct version of the JDBC connector for your computer can be downloaded from the Downloads area of the MariaDB website at https://mariadb.org/connector-java/all-releases/. Installation instructions can be found in the MariaDB Knowledge Base at https://mariadb.com/kb/en/mariadb-connector-j/.

When downloading a JDBC connector, make sure that you choose the correct version for your operating system, database version, and Java version. Compatibility is addressed in the MySQL and MariaDB documentation specified in the previous paragraphs. A JDBC connector cannot be used if your system does not support Java.

Take the following steps to inform Base where it can find the JDBC connector.

1) Select **Tools > Options > LibreOffice > Advanced** on the Menu bar.

2) LibreOffice displays the Options – LibreOffice – Advanced dialog (Figure 102).



Java Options

☑ Use a Java runtime environment

Java runtime environments (JRE) already installed:

	Vendor	Version
●	Oracle Corporation	14.0.2

Location: C:\Program Files\Java\jdk-14.0.2

Optional Features

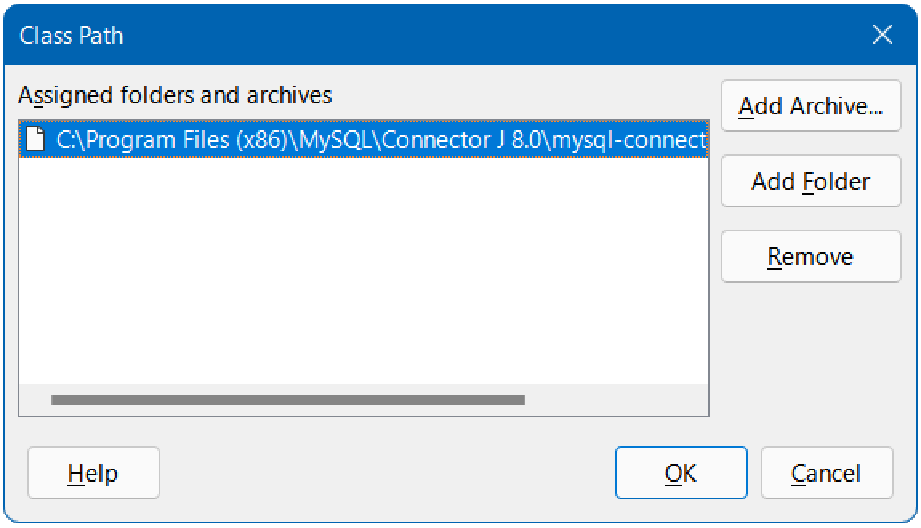
☐ Enable experimental features (may be unstable)

☐ Enable macro recording (may be limited)

Figure 102: Options - LibreOffice - Advanced dialog

3) Click the **Class Path** button.

4) LibreOffice displays the Class Path dialog (Figure 103)



Assigned folders and archives

C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connect

Figure 103: Class Path dialog

5) Click the **Add Archive** button.

6) LibreOffice displays the Select Archives dialog (not shown).

7) Navigate to the folder containing the JAR file for the JDBC connector, select the file, and click the **Open** button.

8) Make sure that the required JAR file is selected and click the **OK** button on the Class Path dialog.

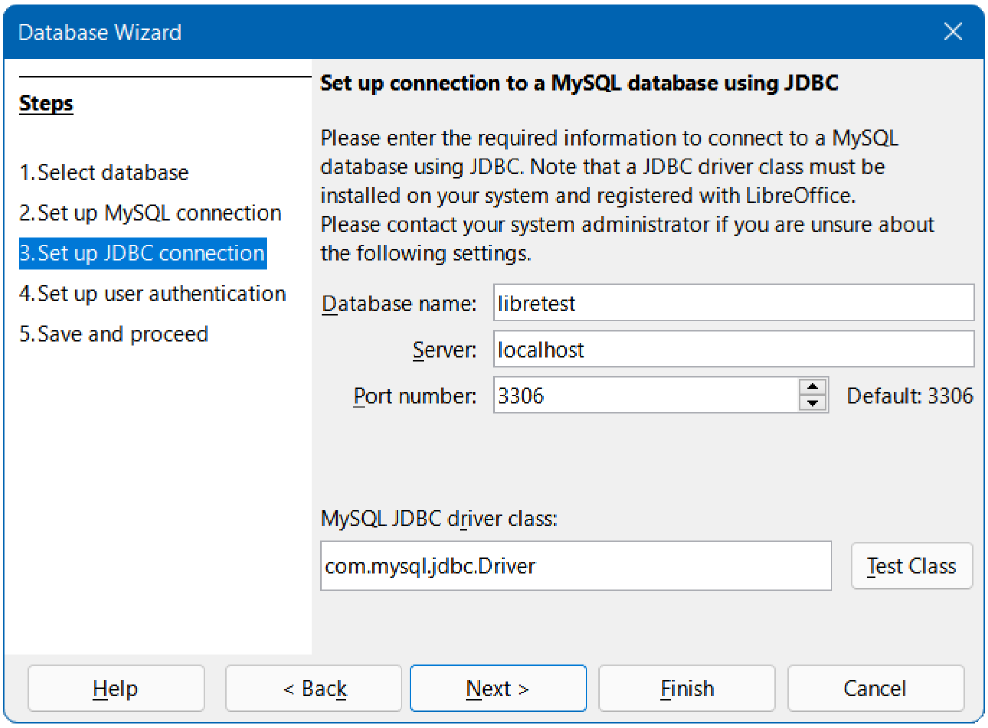
9) Click the **OK** button on the Options – LibreOffice – Advanced dialog.

When you later attempt to create a JDBC connection to your database as described in the next subsection, Base should be able to find the JDBC connector.

Using the Database Wizard

The first time that you want to make a JDBC connection to a particular MySQL or MariaDB database, carry out the following steps.

1) Access the Database Wizard by performing one of the actions listed in step 1) of the procedure for creating a new database using the embedded database engine (page 81).
2) Base displays Step 1 (*Select database*) of the Database Wizard (Figure 88).
3) Click the **Connect to an existing database** option. Base enables the adjacent drop-down menu listing the available options (Figure 92).
4) Select **MySQL** in the drop-down menu.
5) Click the **Next >** button.
6) Base displays Step 2 (*Set up MySQL connection*) of the Database Wizard (Figure 93).
7) If necessary, click the **Connect using JDBC (Java Database Connectivity)** option. This is the default choice for this set of three radio buttons.
8) Click the **Next >** button.
9) Base displays Step 3 (*Set up JDBC connection*) of the Database Wizard (Figure 104).



Set up connection to a MySQL database using JDBC

Please enter the required information to connect to a MySQL database using JDBC. Note that a JDBC driver class must be installed on your system and registered with LibreOffice.
Please contact your system administrator if you are unsure about the following settings.

Database name: libretest

Server: localhost

Port number: 3306 Default: 3306

MySQL JDBC driver class: com.mysql.jdbc.Driver

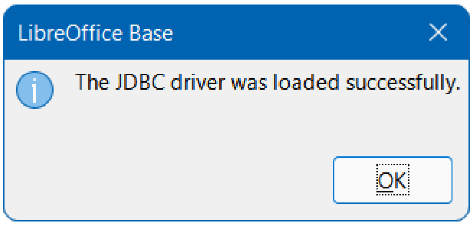
Figure 104: Step 3 of Database Wizard: Set up JDBC connection

10) Enter the name of your database in the *Database name* field. Our example database is called *libretest*.
11) Enter the name of the server that runs the MySQL or MariaDB database in the *Server* field. If you are connecting to a database on the same computer, the server name should be *localhost*. If you are connecting to a remote server, then it may be an IP address, a network computer name, or an Internet address.
12) Enter the port number for the database server in the *Port number* field. The default port number for MySQL and MariaDB databases is 3306.

Tip

At steps 10) to 12), if you are unsure about the correct information to be entered, seek guidance from your database administrator.

13) Providing that Base is aware of the JDBC connector, it automatically populates the *MySQL JDBC driver class* field with the name of the connector. This is either *com.mysql.jbc.Driver* (for MySQL Connector/J) or *org.mariadb.jdbc.Driver* (for MariaDB Connector/J).
14) Click the **Test Class** button to check that the connector's JAR file is accessible to Base. If successful, Base displays a confirmation dialog (Figure 105). Click the **OK** button to remove this dialog from the screen.



The JDBC driver was loaded successfully.

Figure 105: JDBC connector successfully loaded dialog

15) Click the **Next >** button.
16) Base displays Step 4 (*Set up user authentication*) of the Database Wizard (similar to Figure 95).
17) Enter a valid user name in the *User name* field. Our example user is name *lotest*.
18) Click the **Password required** option.
19) If required, test the connection by carrying out steps 18) to 23) of the direct connection procedure (page 88).
20) Click the **Next >** button.
21) Base displays Step 5 (*Save and proceed*) of the Database Wizard (with the same options as are shown in Figure 90).
22) Click the **No, do not register the database** option, as this is a test database.
23) Click the **Open the database for editing** option, as normally this will be appropriate for a new database. You may deselect the option if you do not wish to edit the database at this time.
24) Unless you wish to use the Table Wizard, leave the **Create tables using the table wizard** option in its default deselected state.
25) Click the **Finish** button.
26) Base displays the Save As dialog, where you should specify a file name and location for the ODB file.
27) Click the **Save** button.
28) The Database Wizard is removed from the screen, the ODB file containing the database connection information is created, and Base updates its main window to show information about the database tables under the name of the database itself (similar to Figure 98).

Tables can be created, edited, and deleted.

In this case, the Status bar shows the following information:

- The name of the connector used to connect to the MySQL or MariaDB database (*MySQL (JDBC)* in this case).
- The name of the open MySQL or MariaDB database (*libretest* in this case).
- The user name that is being used to connect to the MySQL or MariaDB database (*lotest* in this case).
- The name of the server on which the MySQL or MariaDB database is running (*localhost* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying MySQL or MariaDB database, the Authentication Required dialog (Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.

Tip

If the *Tables* area of the main Base window shows INFORMATION_SCHEMA tables, these should normally be ignored by most users. The simplest approach is to collapse the *information_schema* node in the table tree structure. Alternatively, select **Tools > Table Filter** on the Menu bar, deselect the *information_schema* node on the Tables Filter dialog, and click the **OK** button. Then select **View > Refresh Tables** on the Menu bar, after which the *information_schema* node and its children should no longer be visible in the *Tables* area.

Possible limitations and workarounds

Tables without primary keys

In MySQL and MariaDB, it is possible to enter and change data in tables without a primary key field. The Base user interface can display such tables, but offers no input or modification options.

If you want to use tables without a primary key, you can instead use the **Tools > SQL** option, or include macros within forms, to supply the tables with data.

Connecting via the Internet

Connections to MySQL or MariaDB databases may be to a local server (“localhost”), to a networked server, or to an Internet server. MySQL and MariaDB provide a *wait_timeout* system variable that defines the time that the server waits for an inactive connection to become active before closing it. In some cases you may feel that the current wait timeout is inappropriate for your tasks, for example when a low *wait_timeout* has been set to avoid exceeding the maximum number of connections.

Depending on the server settings, it may be possible to modify the value of *wait_timeout*. Try the following command using the **Tools > SQL** option:

```
SET SESSION wait_timeout = 600;
```

If this is successful, *wait_timeout* is reset to 600 seconds or 10 minutes. As soon as any query is executed, the wait time until the next interruption is reset to 10 minutes.

If the *wait_timeout* cannot be reset to a suitable value, then you could consider writing a macro that makes a simple query (e.g. `SELECT NOW()`) at a regular rate, with the frequency selected to keep your connection open.

With a JDBC connection, you can try setting the *autoReconnect* configuration parameter to *true*. In this case, a first interruption may occur, but the connection will be restored next time contact is attempted.

With a direct connection, the database file must be closed and reopened after an interruption to re-establish contact with the server. However, connection data such as login credentials do not need to be reentered.

Accessing stored routines

Both MySQL and MariaDB support stored routines (procedures and functions). A stored routine is a set of SQL commands that are stored on the database server and invoked using a `CALL` statement. A stored routine can have parameters so that you can pass values to it and get results back.

In Base, stored routines can be called using the **Tools > SQL** option. However, there is no direct way to display the output from a stored routine in Base.

A straightforward workaround to access the output of a stored routine is to write the routine's output to a temporary table in the database, which will be visible only to the current database user. That user can then query the temporary table to access the results from the stored routine. A more automated approach to solving this issue is discussed in Chapter 9, Macros.

PostgreSQL databases

Overview

PostgreSQL (often simply referred to as "Postgres") is an open-source RDBMS provided by The PostgreSQL Global Development Group. More information can be found on the PostgreSQL website at https://www.postgresql.org/. Links to current and past versions of the PostgreSQL Documentation can be found at https://www.postgresql.org/docs/.

Base can connect to a PostgreSQL database using one of three methods:

- Direct connection using a built-in PostgreSQL connector.
- Connection using ODBC.
- Connection using JDBC. This may be a good option if a secure connection to a remote PostgreSQL database is to be established via the SSH protocol (Secure Shell).

The user interactions necessary to connect to a PostgreSQL database have some commonality with the interactions for connecting to a MySQL or MariaDB database. However, there are also significant differences. The aim of the following subsections is to highlight and explain those differences, without repeating details that are identical to those already described for MySQL/MariaDB.

Introduction to PostgreSQL schemas

Schemas are important components of a PostgreSQL database, but new users should be aware that the term *schema* is not always used with the same meaning in other relational database management systems. More detailed information can be found in the Schemas chapter of the PostgreSQL Documentation (https://www.postgresql.org/docs/current/ddl-schemas.html).

The following notes highlight some of the features of PostgreSQL schemas that may be reflected in the Base user interface when connected to a PostgreSQL database.

- A schema is a named collection of database objects, including (but not limited to) tables and views.
- A database contains one or more schemas. Two schemas within a database may contain different objects, which share a similar name.
- For every new database, PostgreSQL generates a schema called *public*. When you create a database object without specifying the schema name, it is placed in the *public* schema.

- For every new database, PostgreSQL generates additional schemas intended for system purposes. Some of these may be visible in Base depending on the type of connection (for example, the schemas *information_schema* and *pg_catalog*). Users should not save their database objects in schemas that are intended for system use.
- Users can create their own schemas via the **Tools > SQL** option, but names beginning with the prefix "pg_" should be avoided. To access an object in a specific schema, qualify the object using the syntax: `schema_name.object_name`.

User-created schemas do not appear in the *Tables* area of the main Base window until they contain at least one saved database object.

If you find that that the *Tables* area of the main Base window becomes cluttered with too many schemas and / or tables, select **Tools > Table Filter** on the Menu bar and use the Tables Filter dialog to select items to be displayed or hidden. Note that you may need to select **View > Refresh Tables** to see the effect of any changes made through this dialog.

The first time you save a new table created in Design View, Base displays a Save As dialog that includes a *Schema* menu from which the required schema should be selected (Figure 106). A similar menu is provided on the dialog displayed when you select a query and then select **Edit > Create as View** on the Menu bar.

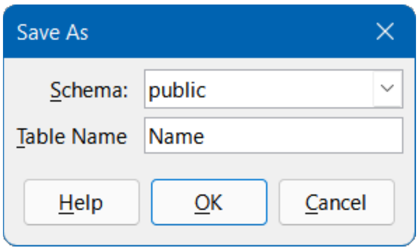

Schema: public

Table Name: Name

Figure 106: Save As dialog with Schema drop-down menu

Initial preparations before connecting

Refer to the PostgreSQL Documentation for detailed instructions about downloading and installing the database server software for your computer configuration, and for details of how to access any tool that you require to enter SQL commands prior to connecting with Base. After installation, follow any setup instructions specified in the documentation. These may include making sure that the password for the default superuser account of the database server (user name *postgres*) has been set to a string that is appropriate to maintain the security of your data.

Before making a connection, the database of interest must exist on the server and you must have a user name and password. If your database and user name do not yet exist, the *postgres* user should carry out the following steps to create them. Commands may be entered through a terminal-based front-end (such as *psql*) or a graphical interface (such as *pgAdmin*).

1) To create an (empty) database called *libretest*, enter the following command:

```
CREATE DATABASE libretest;
```

2) To grant all rights on the new database to a new user called *lotest* with the password *libre*, enter the following commands:

```
CREATE USER lotest WITH PASSWORD 'libre';
GRANT ALL PRIVILEGES ON DATABASE libretest TO lotest;
```

3) If needed, to create a schema called *user_area* for the *lotest* user to save database objects, enter the following commands:

```
CREATE SCHEMA user_area;
ALTER SCHEMA user_area OWNER TO lotest;
```

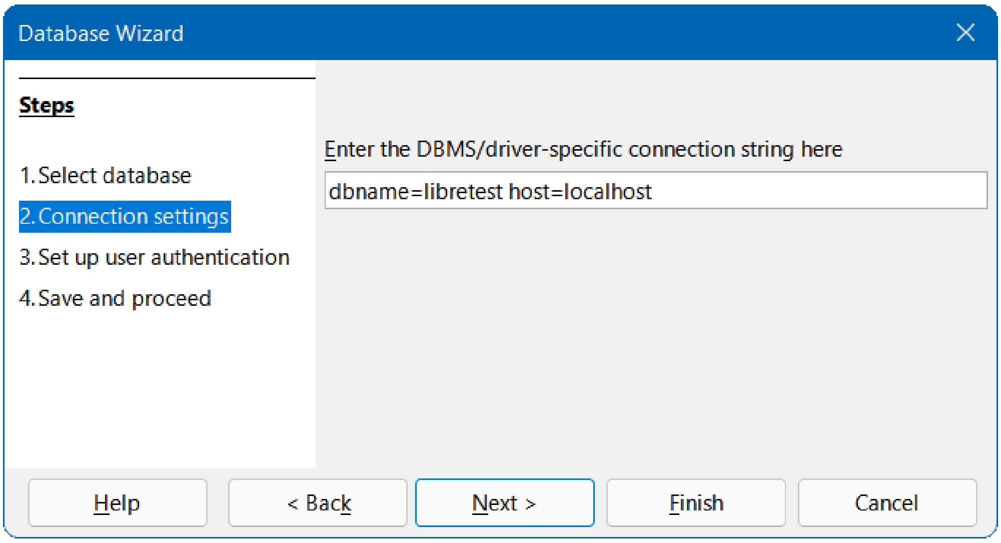
On successful completion of these steps, the new database is available and Base can connect to it, as described in the following subsections.

Connecting directly

The first time that you make a direct connection to a particular PostgreSQL database, carry out steps similar to those for a direct connection to a MySQL/MariaDB database (see Connecting directly subsection on page 86), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **PostgreSQL** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) and Step 3 (*Set up MySQL server data*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to PostgreSQL. Instead, Base displays Step 2 (*Connection settings*) of the Database Wizard (Figure 107).

  Enter the name of your database and the host name, as shown in Figure 107 (*dbname=libretest host=localhost*). This utilizes the plain keyword/value string format for specifying connection parameters, which is described further in the Connection Strings section of the PostgreSQL Documentation (https://www.postgresql.org/docs/current/libpq-connect.html). Under some circumstances, it is necessary to give the fully qualified host name, including the domain name.

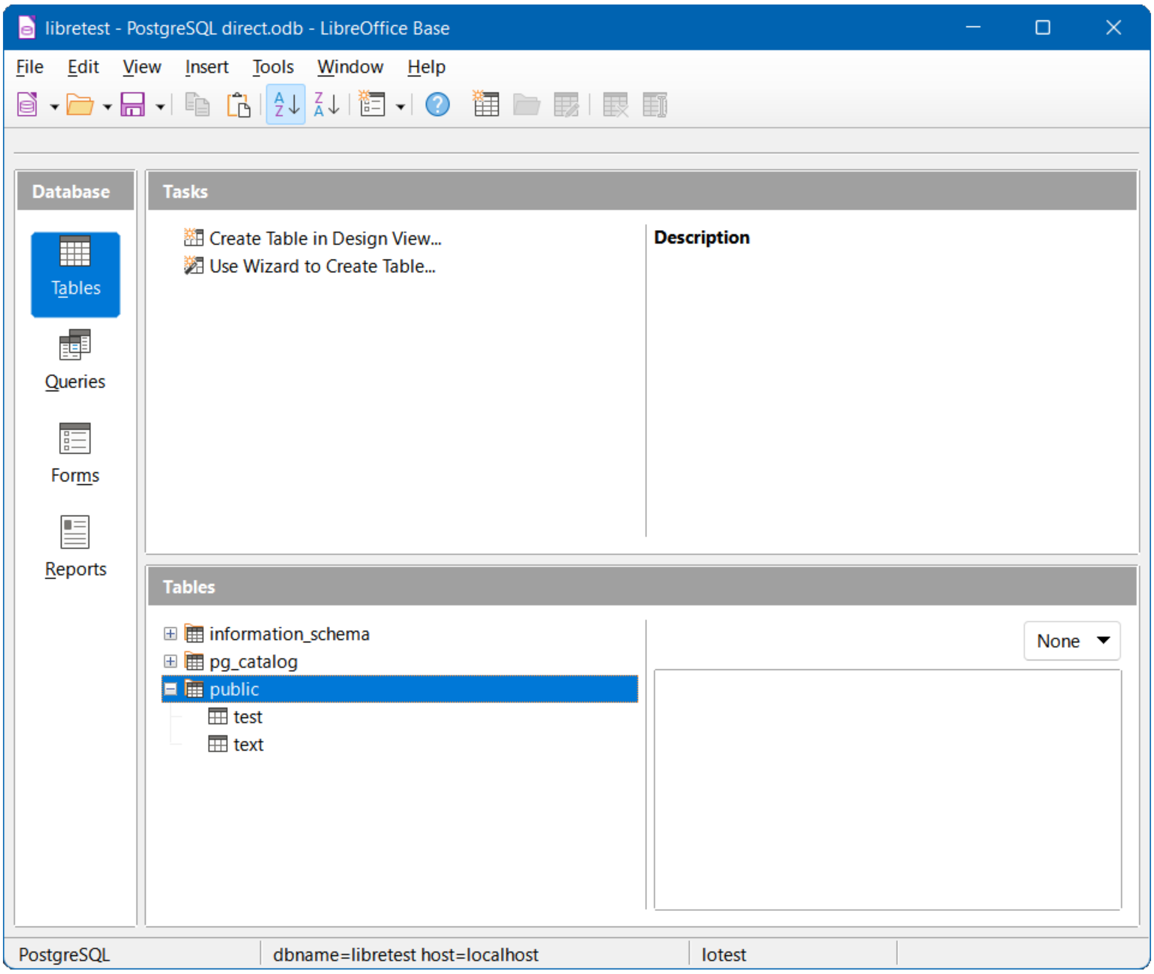


Enter the DBMS/driver-specific connection string here

dbname=libretest host=localhost

Figure 107: Step 2 of Database Wizard: Connection settings

When the connection is established and the ODB file has been created, Base updates its main window to show information about the PostgreSQL database (Figure 108).



PostgreSQL | dbname=libretest host=localhost | lotest

Figure 108: Main Base window when connected directly to PostgreSQL

The *Tables* area shows three schemas, named *information_schema*, *pg_catalog*, and *public*. The *public* schema has been expanded to show its two constituent tables, *test* and *text*.

The Status bar at the bottom of Figure 108 shows:

- The name of the connector used (*PostgreSQL*).
- Details of the connection string entered at Step 2 of the Database Wizard (*dbname=libretest host=localhost* in this case).
- The user name that was used to connect to the PostgreSQL database (*lotest* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying PostgreSQL database, the Authentication Required dialog (similar to that shown in Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.

Connecting with ODBC

Initial preparation

To use Base to connect to a PostgreSQL database server through the ODBC API, there should be a properly configured ODBC driver installed on your computer. If the driver is not currently available on your computer, then you will need administration rights to install the new software.

To connect to a PostgreSQL database through ODBC, it is recommended that you use the official PostgreSQL ODBC driver (psqlODBC). The correct version of the ODBC driver for your computer can be downloaded from the psqlODBC website at https://odbc.postgresql.org/, where relevant documentation can also be found. For Windows, the driver is available as a standard, easy-to-use installer package. For Unix environments, you must compile the source code yourself, using instructions given on the website. No instructions are given for using psqlODBC on macOS.

Make sure that you follow the instructions given in the driver documentation to create an ODBC Data Source Name (DSN) for your database. For the purposes of the example in the next subsection, assume that a data source has been configured as follows:

- The PostgreSQL Unicode driver is selected.
- The name of the data source is set to *libretest-PostgreSQL-ODBC*.
- The database name (*libretest*) is included within the definition of the data source.
- Login credentials for the database server are not specified within the data source.

Using the Database Wizard

The first time that you make an ODBC connection to a particular PostgreSQL database, carry out steps similar to those for an ODBC connection to a MySQL/MariaDB database (see Using the Database Wizard subsection on page 91), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **ODBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to PostgreSQL.
- At Step 2 (*Set up ODBC connection*), enter the name of the previously defined ODBC data source that is to be used to access the PostgreSQL database (*libretest-PostgreSQL-ODBC* in our example). You can click the **Browse** button to access the Data Source dialog (Figure 101), where you can select the data source of interest from a list of those available and then click the **OK** button.

When the connection is established and the ODB file has been created, Base updates its main window to show information about the PostgreSQL database.

By default for an ODBC connection to PostgreSQL, the *Tables* area only displays objects in the public schema, even when other user-defined schemas are present.

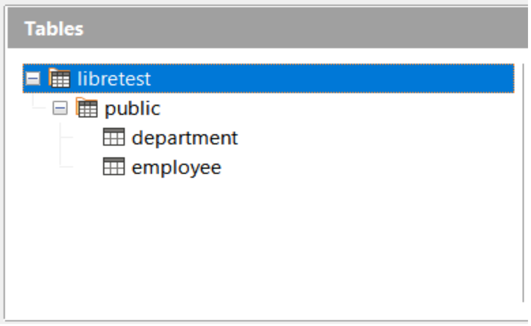


Figure 109: Initial layout of Tables area for ODBC connection to PostgreSQL

To see other defined schemas in the *Tables* area, it can be helpful to reset the *search_path* variable to the database level. One possible way to achieve this is to configure the ODBC driver to issue an appropriate command when it connects to the database. For example, on a Windows system this may be achieved using the ODBC Data Source Administrator dialog – select the driver, click **Configure**, click **Datasource**, click **Page 2**, and then type the command `SET SEARCH_PATH TO libretest;` in the *Connect Settings* text box. On Unix, make sure that the file “odbc.ini” contains the entry `ConnSettings = SET SEARCH_PATH TO libretest;`. The impact of changing *search_path* can be seen in Figure 110, which represents exactly the same underlying database but now the additional *lotest_data* schema is visible.

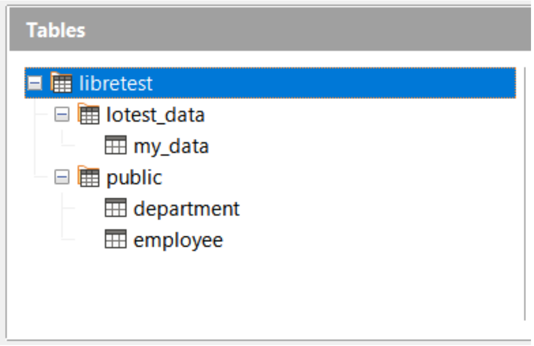


Figure 110: Updated layout of Tables area for ODBC connection to PostgreSQL

With an ODBC connection to PostgreSQL, the Status bar of the main Base window shows the following information:

- The name of the connector used (*ODBC*).
- The name of the ODBC data source (*libretest-PostgreSQL-ODBC* in this case).
- The user name that was used to connect to the PostgreSQL database (*lotest* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying PostgreSQL database, the Authentication Required dialog (similar to that shown in Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.

Connecting with JDBC

Initial preparation

To use Base to connect to a PostgreSQL database server through the JDBC API, there should be an appropriate JDBC driver located on your computer. If the driver is not currently available on your computer, then you will need to download it and place a specific JAR (Java ARchive) file in an accessible folder.

To connect to a PostgreSQL database through JDBC, it is recommended that you use the official PostgreSQL JDBC Driver. The correct version of the JDBC driver for your computer can be downloaded from the PostgreSQL JDBC Driver website at https://jdbc.postgresql.org/. Installation instructions can be found in the PostgreSQL JDBC Driver Documentation, which is accessible from https://jdbc.postgresql.org/documentation/documentation.html.

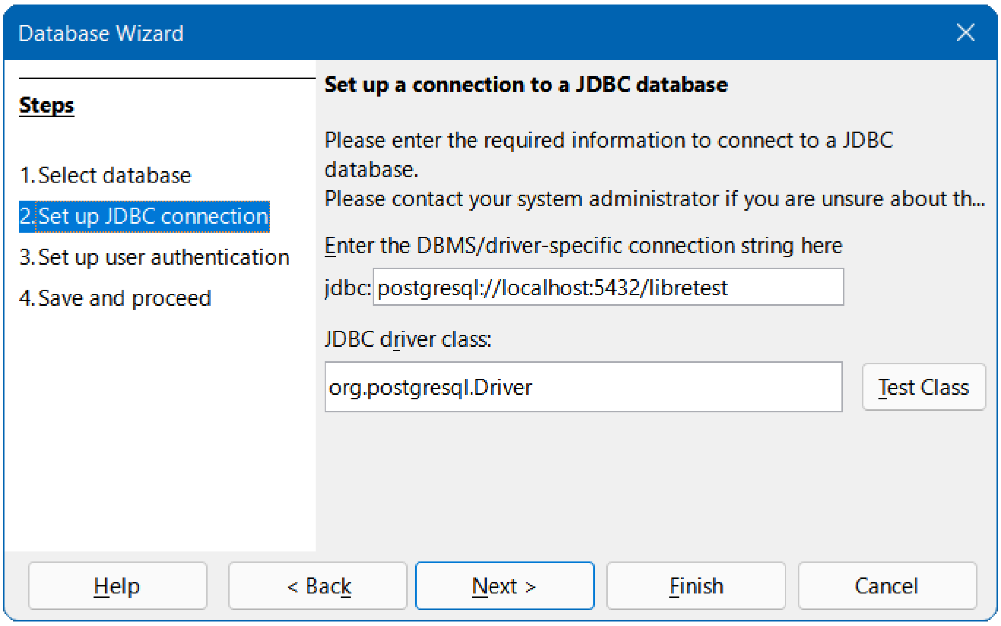
When downloading a JDBC driver, make sure that you choose the correct version for your computer configuration. A JDBC driver cannot be used if your system does not support Java.

Carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database to inform Base where it can find the JAR file for the JDBC driver (see Initial preparation subsection on page 93).

Using the Database Wizard

The first time that you make a JDBC connection to a particular PostgreSQL database, carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database (see Using the Database Wizard subsection on page 95), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **JDBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to PostgreSQL.
- For a JDBC connection, Base displays Step 2 (*Set up JDBC connection*) of the Database Wizard (Figure 111).



Set up a connection to a JDBC database

Please enter the required information to connect to a JDBC database.
Please contact your system administrator if you are unsure about th...

Enter the DBMS/driver-specific connection string here

jdbc: postgresql://localhost:5432/libretest

JDBC driver class:

org.postgresql.Driver

Figure 111: Step 2 of Database Wizard: Set up JDBC connection

- – Enter the connection string using one of the forms given in the PostgreSQL JDBC Driver Documentation. In Figure 111, the *jdbc:postgresql://host:port/database* form is used.
- – Enter the name of the driver as *org.postgresql.Driver* in the *JDBC driver class* field.
- – Click the **Test Class** button to check that the driver’s JAR file is accessible to Base. If successful, Base displays a confirmation dialog (Figure 105). Click the **OK** button to remove this dialog from the screen.

By default for a JDBC connection to PostgreSQL, the *Tables* area displays objects in the *public* schema and other user-defined schemas (such as *lotest_data* in Figure 112).

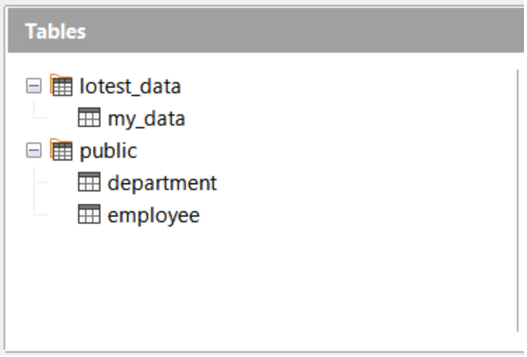


Figure 112: Initial layout of Tables area for JDBC connection to PostgreSQL

With a JDBC connection to PostgreSQL, the Status bar of the main Base window shows the following information:

- The name of the connector used (*JDBC*).
- The connection string entered at Step 2 of the Database Wizard (*postgresql://localhost:5432/libretest* in this case).
- The user name that was used to connect to the PostgreSQL database (*lotest* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying PostgreSQL database, the Authentication Required dialog (similar to that shown in Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.

Possible limitations and workarounds

Copying a table from another database

In some scenarios a user may want to use Base to copy a table from some database to a PostgreSQL database. With both databases open in Base, a simple way to achieve this is to right-click the table name in the source database and select **Copy** from the context menu. Moving to the PostgreSQL database, make sure that the **Tables** option is selected in the *Database* area of the main Base window and click the **Paste** button in the Standard toolbar. Base displays the Copy table dialog (Figure 113), in which you should add the required prefix in the *Table name* field (for example, “public.”). For the ODBC driver (for which we set *search_path* to *libretest*), a typical prefix would be “libretest.public”.

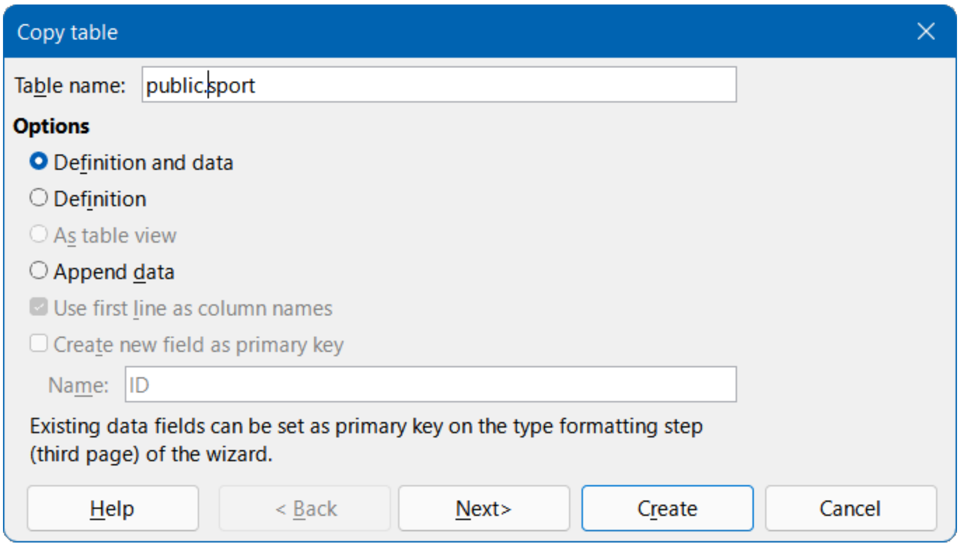

Copy table

Table name: public.sport

Options

● Definition and data

○ Definition

○ As table view

○ Append data

☑ Use first line as column names

□ Create new field as primary key

Name: ID

Existing data fields can be set as primary key on the type formatting step (third page) of the wizard.

Figure 113: Copy table dialog

PostgreSQL auto-increment values

When using an embedded HSQLDB database, users may create a primary key for a table by defining a column named *id* (or similar), of type **Integer**, with its *AutoValue* property set to **Yes**. If the same approach is adopted for a table in a PostgreSQL database, it will not have the desired effect and the column values will not auto-increment – the user must enter those values.

A long-established and common method for adding an auto-incrementing primary key column in a PostgresSQL table is to use one of the non-standard BIGSERIAL, SERIAL, or SMALLSERIAL data types. For example, the following SQL code creates a simple table using the SERIAL data type:

```
CREATE TABLE public.employee (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);
```

When creating a table in Design View, with either the direct or the ODBC connection, the SERIAL data types are unavailable for selection in the Field Type column. However, they are available with the JDBC connection.

In later versions of PostgreSQL, an alternative approach is available, using a standard integer data type. In this case, the above SQL code could be refactored as follows:

```
CREATE TABLE public.employee (
  id INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
  name VARCHAR(100) NOT NULL
);
```

In cases where you cannot select the desired type of primary key in Design View, it is recommended that you select **Tools > SQL** on the Menu bar, enter the SQL code necessary to create a table comprising just the required primary key. You can then switch to Design View and add the remaining columns. In the first instance, the new table may not appear in the *Tables* area of the main Base window, but this can be rectified by selecting **View > Refresh Tables** on the Menu bar.

Note

If using either a direct connection or a JDBC connection, you may encounter a problem when using the graphical user interface to run a query (Bug 130376). This problem could result in data loss, but should only arise with queries that involve an auto-incrementing field and a table alias (in the FROM clause of the SQL command). The issue can be avoided either by removing the table alias or by running any such query directly via the **Tools > SQL** option. If using an ODBC connection, difficulties should only arise if the option **Respect the result set type from the database driver** is ticked on the *Special Settings* tab of the **Edit > Database > Advanced Settings** dialog.

PostgreSQL views

The normal mechanism for saving a view through Base is to select a query and then select **Edit > Create as View** on the Menu bar. However, the three methods of connecting to a PostgreSQL database lead to different behaviors with respect to this interaction as follows:

- **Direct connection**. The direct driver does not allow creation of views through this interaction.
- **ODBC connection**. The OBDC driver does not allow creation of views through this interaction as long as no view already exists. However, this can be remedied by creating a dummy view via the **Tools > SQL** option (for example, the following code from the PostgreSQL documentation could be used: `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`).
- **JDBC connection**. The JBDC driver always allows creation of views through this interaction.

For all connection types, views can be created through the **Tools > SQL** option.

Firebird databases

Overview

Firebird is an open-source RDBMS provided by the Firebird Foundation. More information can be found on the Firebird website at https://firebirdsql.org/. Links to current and past versions of the Firebird documentation can be found at https://firebirdsql.org/en/documentation/.

Base can connect to an external database via a Firebird server using one of two methods:

- Connection using ODBC.
- Connection using JDBC.

The user interactions necessary to establish such a connection have some commonality with the interactions for connecting to a MySQL or MariaDB database. However, there are also significant differences. The aim of the following subsections is to highlight and explain those differences, without repeating details that are identical to those already described for MySQL/MariaDB.

If access to a server database is not required, then it is possible for Base to directly open an external Firebird database file. In this case the opened database will be managed using the Firebird engine embedded within Base. It is not necessary to enable experimental features within Base to utilize this facility.

Note

The embedded engine is based on Firebird 3.0.7. The Firebird 3 Quick Start Guide states that Firebird 3 will not open databases created using earlier versions. Similarly, Firebird 4 creates databases with different on-disk structure and such databases cannot be opened directly using the embedded Firebird 3 engine.

Initial preparations before connecting to a Firebird server

Refer to the Firebird documentation for detailed instructions about downloading and installing the database server software for your computer configuration, and for details of how to access any tool that you require to enter SQL commands prior to connecting with Base. After installation, follow any setup instructions specified in the documentation. These may include making sure that the password for the default superuser account of the database server (user name *SYSDBA*) has been set to a string that is appropriate to maintain the security of your data. The default password for user *SYSDBA* is set to *masterkey*.

Note

At the time of drafting this section, no download of Firebird 4 was available for macOS. In addition, no Firebird 4 Quick Start Guide was available.

Before making a connection, the database of interest must exist and you must have a user account that gives you permissions to work with the database and the objects within it. If your database and user name do not yet exist, the *SYSDBA* superuser should carry out the following steps to create them. Commands may be entered using the Firebird Interactive SQL Utility (*isql*) – early versions of Firebird provided the *gsec* utility for user management, but this is now deprecated and such actions are performed using SQL commands.

Note

In some Firebird installations, *isql* may have been renamed to *isql-fb* to avoid a potential name clash.

1) To create an (empty) database in the file *libretest.fdb*, located in the folder *c:\my_databases\*, enter the following command (replacing the string “masterkey” with the password selected for user *SYSDBA*):

```
CREATE DATABASE 'c:\my_databases\libretest.fdb'
USER SYSDBA
PASSWORD 'masterkey';
```

2) To create a new user called *lotest* with the password *libre*, enter the following command:

```
CREATE USER lotest PASSWORD 'libre';
```

Information about users authorized to access a specific Firebird server is stored in a security database named *security3.fdb* or *security4.fdb*, depending on the version of Firebird. These files are normally found in the installation directory.

3) To allow the new user to create and alter tables or views in the *libretest* database, enter the following commands:

```
GRANT ALL PRIVILEGES TABLE TO USER lotest;
GRANT ALL PRIVILEGES VIEW TO USER lotest;
COMMIT;
```

4) Finally, it is useful to create an alias for the new database. Edit the file named *databases.conf* that is normally located in the installation directory so that it contains the line:

```
libretest=c:\my_databases\libretest.fdb
```

This allows the database to be referenced by its alias *libretest*, thus hiding the physical location of the database and reducing the possibility of typing errors.

Tip

At steps 2) and 4) above, you may need to adjust file permissions on the *security3.fdb*, *security4.fdb*, or *databases.conf* files before the specified actions can be carried out.

On successful completion of these steps, the new database is available and Base can connect to it, as described in the following subsections.

Note

If some user other than *lotest* creates tables in the *libretest* database, then *lotest* may not be able to fully access those tables. To enable *lotest* to fully access a table named *Table1*, enter the following command:

```
GRANT ALL ON TABLE "Table1" TO USER lotest;
```

Connecting with ODBC

Initial preparation

To use Base to connect to a Firebird server through the ODBC API, there should be a properly configured ODBC driver installed on your computer. If the driver is not currently available on your computer, then you will need administration rights to install the new software.

To connect to a Firebird server database through ODBC, it is recommended that you use the official Firebird ODBC driver. The correct version of the Firebird ODBC driver for your computer can be downloaded from the Firebird website at https://dev.mysql.com/downloads/connector/odbc/. Platform-dependent installation instructions can be found in the Firebird ODBC/JDBC Driver 2.0 Manual, accessed through the Firebird Drivers Documentation page at https://firebirdsql.org/en/drivers-documentation/.

Note

At the time of drafting this section, the Firebird ODBC/JDBC Driver 2.0 Manual provided no installation instructions for macOS.

Make sure that you follow the instructions given in the driver documentation to create an ODBC Data Source Name (DSN) for your database. For the purposes of the example in the next subsection, assume that a data source has been configured as follows:

- The name of the data source is set to *libretest-firebird*.
- The alias for the database of interest is included within the definition of the data source (for example, *localhost:libretest*).
- Login credentials for the database server are not defined within the data source.
- The *Sensitive Identifier* extended identifier property is enabled.

Using the Database Wizard

The first time that you make an ODBC connection to a particular Firebird database, carry out steps similar to those for an ODBC connection to a MySQL/MariaDB database (see Using the Database Wizard subsection on page 91), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **ODBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to Firebird.
- At Step 2 (*Set up ODBC connection*), enter the name of the previously defined ODBC data source that is to be used to access the Firebird database (*libretest-firebird* in our example). You can click the **Browse** button to access the Data Source dialog (Figure 101), where you can select the data source of interest from a list of those available and then click the **OK** button.

When the connection is established and the ODB file has been created, Base updates its main window to show information about the Firebird database.

With an ODBC connection to Firebird, the Status bar of the main Base window shows the following information:

- The name of the connector used (*ODBC*).
- The name of the ODBC data source (*libretest-firebird* in this case).
- The user name that was used to connect to the Firebird database (*lotest* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying Firebird database, the Authentication Required dialog (similar to that shown in Figure 99) is displayed. You do not need to reconfigure the connection but you do need to reenter your correct login credentials.

Tip

When creating tables in a Firebird database via ODBC, take care to ensure that the correct type is assigned to each field from the start. It is not possible to alter the field type once the table has been saved, although it is possible to delete unwanted fields and add new fields.

Connecting with JDBC

Initial preparation

To use Base to connect to a Firebird server through the JDBC API, there should be an appropriate JDBC driver located on your computer. If the driver is not currently available on your computer, then you will need to download it and place a specific JAR (Java ARchive) file in an accessible folder.

To connect to a Firebird database through JDBC, it is recommended that you use the official Firebird JDBC driver which is called Jaybird. The correct version of Jaybird for your computer can be downloaded from the Firebird website at https://firebirdsql.org/en/jdbc-driver/. Instructions for use can be found on the Firebird Drivers Documentation page at https://firebirdsql.org/en/drivers-documentation/.

When downloading Jaybird, make sure that you choose the correct version for your computer configuration. A JDBC driver cannot be used if your system does not support Java.

Carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database to inform Base where it can find the JAR file for Jaybird (see Initial preparation subsection on page 93). Use the JAR file with a name of the form *jaybird-full-4.0.5.java11.jar* (the digits in this name vary depending on the versions of Jaybird and Java selected on the download page).

Using the Database Wizard

The first time that you make a JDBC connection to a particular Firebird database, carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database (see Using the Database Wizard subsection on page 95), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **JDBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to Firebird.
- For a JDBC connection, Base displays Step 2 (*Set up JDBC connection*) of the Database Wizard (Figure 111).
- Enter details of the database as *jdbc:firebirdsql:oo://localhost/libretest?charSet=utf-8*. If the character set is not specified here, an error will be displayed.
- Enter the name of the driver class as *org.firebirdsql.jdbc.FBDriver*.
- Click the **Test Class** button to check that the driver’s JAR file is accessible to Base. If successful, Base displays a confirmation dialog (Figure 105). Click the **OK** button to remove this dialog from the screen.

With a JDBC connection to Firebird, the Status bar of the main Base window shows the following information:

- The name of the connector used (*JDBC*).
- The name of the JDBC data source as entered at Step 2 of the Database Wizard (*firebirdsql:oo://localhost/libretest?charSet=utf-8*).
- The user name that was used to connect to the Firebird database (*lotest* in this case).

At some later time when you open this ODB file again and then use Base to access the underlying Firebird database, the Authentication Required dialog (similar to that shown in Figure 99) is displayed. You do not need to reconfigure the connection, but you do need to reenter your correct login credentials.

Tip

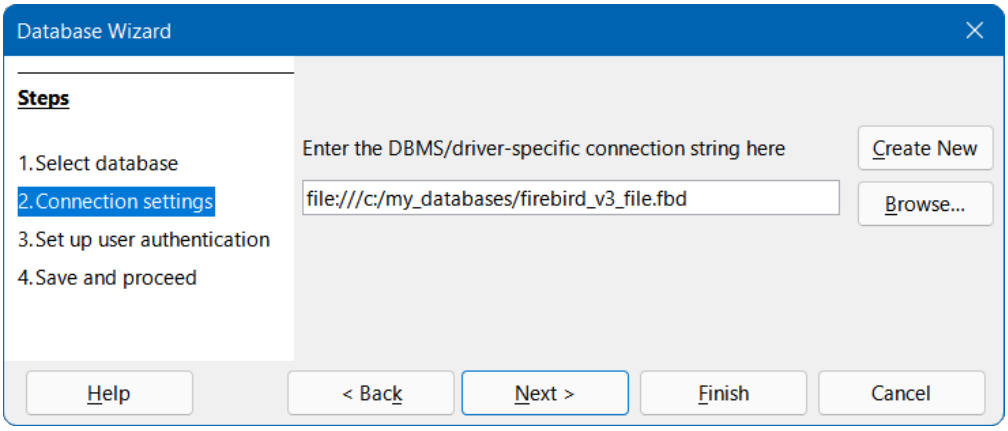
When creating tables in a Firebird database via JDBC, take care to ensure that the correct type is assigned to each field from the start. It is not possible to alter the field type once the table has been saved, although it is possible to delete unwanted fields and add new fields.

Direct connection to a Firebird file

To connect to a (non-server) Firebird 3 database file located in a path for which you have read and write access, carry out steps similar to those for setting up a direct connection to a MySQL/MariaDB database (see Connecting directly subsection on page 86), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **Firebird External** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) and Step 3 (*Set up MySQL server data*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to the **Firebird External** option. Instead, Base displays Step 2 (*Connection settings*) of the Database Wizard as shown in Figure 114.

  Enter the name and location of your Firebird 3 file in the text field provided or use the **Create New** or **Browse** functions as required.



Enter the DBMS/driver-specific connection string here

file:///c:/my_databases/firebird_v3_file.fbd

Figure 114: Step 2 of Database Wizard: Connection Settings (Firebird External)

- At Step 3 (*Set up user authentication*) of the Database Wizard, click the **Next >** button without entering any user authentications details – these are not required to connect to a non-server Firebird 3 database.

When the connection is established and the ODB file has been created, Base updates its main window to show information about the Firebird database.

The Status bar at the bottom of screen shows:

- The connection type used (*Firebird External*).
- The file name and location entered at Step 2 of Database Wizard (for example *file:///C:/my_databases/firebird_v3_file.fdb*).

Note

It is possible to establish a connection to a server database via a Firebird server using the **Firebird External** option on Step 1 (*Select database*) of the Database Wizard. However, the facilities available through such a connection require further investigation and may be described in a future version of this book.

External HSQLDB databases

Overview

HyperSQL Database (HSQLDB) is an open-source RDBMS written in Java and provided by The HSQL Development Group. More information can be found on the HSQLDB website at https://hsqldb.org/. Links to the latest HSQLDB documentation can be found at https://hsqldb.org/web/hsqlDocsFrame.html.

Each HSQLDB database comprises between 2 to 6 files in a directory, all with the same file name but each with a different extension. The possible extensions are:

- *backup*
- *data*
- *lobs*
- *log*
- *properties*
- *script*

The engine embedded in Base utilizes HSQLDB version 1.8.0, which was first released in July 2005. There are several reasons why some users may instead opt to use an external HSQLDB database, including:

- To utilize the additional features provided in later versions of HSQLDB. At the time of updating this chapter, HSQLDB version 2.6.1 was available (released in October 2021).
- To connect to a database that cannot be embedded, such as a preexisting, multi-user database.
- To avoid a potential disadvantage with embedded HSQLDB databases, arising from storing the data in an archive ODB format. Since the data is only committed when the archive is packed, there is a greater risk of data loss than when working with an external database.

It is possible to convert an existing embedded HSQLDB database, extracting it from the ODB package so that it can be used as an external database. However, instructions for this procedure are beyond the scope of this chapter.

To use an external HSQLDB database, connect using the JDBC driver that is provided in the HSQLDB JAR package. Since both Base and HSQLDB utilize Java, it is logical to make a connection between Base and an external HSQLDB database using JDBC. While it should be possible to connect Base to HSQLDB via ODBC (see Chapter 17 of the HyperSQL User Guide), details of this approach are not provided in this chapter.

The user interactions necessary to establish a JDBC connection to an external HSQLDB database have some commonality with the corresponding interactions for connecting to a MySQL or MariaDB database. However, there are also significant differences. The aim of the following subsections is to highlight and explain those differences, without repeating details that are identical to those already described for MySQL/MariaDB.

✓ **Note**

Recent releases of HSQLDB typically have many more features than earlier versions. As a result, there are likely be compatibility issues when trying to convert between an embedded database created using version 1.8.0 and an external database created using a more recent version. In particular, an external database that has been edited using HSQLDB 2.0+ cannot easily be converted into an embedded database under Base.

If you need to convert your database from one HSQLDB version to another, useful information can be found in the sections entitled “Upgrading Databases” and “Backward Compatibility Issues” in the HyperSQL User Guide.

! **Caution**

The integration of the external HSQLDB JAR package *hsqldb.jar* within the Java class path for LibreOffice may mean that embedded databases can no longer be accessed correctly by Base. Base may try to use the external HSQLDB for the embedded databases. While there are steps that could be taken by more advanced users to avoid such a problem (for example, writing the class path for the respective database file using a macro or inserting the path directly into the *content.xml* file of the ODB file), these are beyond the scope of this chapter.

Initial preparations before connecting to a HSQLDB database

If you are satisfied with the functionality of HSQLDB version 1.8.0, but simply want to use it for an external database, then no further download is needed from the Internet. You can simply utilize the *hsqldb.jar* file that was supplied with your version of LibreOffice, normally located in the /program/classes/ sub-folder of the program’s installation folder. This approach avoids any problems arising from incompatibility between different versions. Otherwise, you can download current and legacy versions of HSQLDB through the HyperSQL website.

Refer to the HyperSQL User Guide for detailed instructions about downloading and installing the HSQLDB software for your computer configuration, and for details of how to access any tool that you require to enter SQL commands prior to connecting with Base. After installation, follow any setup instructions specified in the documentation. These may include making sure that the password for the default admin user (SA, which stands for System Administrator) has been set to a string that is appropriate to maintain the security of your data. The default password for user SA is empty. You may also prefer to change the admin username to something other than SA. For simplicity in the remainder of this section, it is assumed that the HSQLDB database is accessed using the default SA username with an empty password, but this is not recommended for a real-world database.

HSQLDB is notable in that when a connection attempt is made to a database that does not currently exist, a new, empty database is created. This means that no specific preparatory actions are needed to create a database before connecting to it.

Identify the location of the *hsqldb.jar* file downloaded from the Hyper SQL website (for example, *C:\Users\username\Downloads\hsqldb-2.6.1\hsqldb-2.6.1\hsqldb\lib\hsqldb.jar*), moving the file to a different location if more appropriate for you. This JAR contains both the HyperSQL engine and the HyperSQL JDBC driver. Whether connecting for single-user access or for multi-user access, carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database to inform Base where it can find this JAR file (see Initial preparation subsection on page 93).

Note

Currently the HSQLDB download zip package includes one version of the *hsqldb.jar* file that is compatible with Java 11 or later and another that is compiled with JDK 8. A further version compiled with JDK 6 can be separately downloaded from the HyperSQL website. Make sure that you use the correct version for your computer configuration.

In-process, single-user connection

For single-user access, use an in-process connection as described in the HyperSQL User Guide. To configure this type of connection, specify the database path in the connection URL.

The first time that you make an in-process connection to an external HSQLDB database, carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database (see Using the Database Wizard subsection on page 95), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **JDBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to HSQLDB.
- For a JDBC connection, Base displays Step 2 (*Set up JDBC connection*) of the Database Wizard (Figure 111).
- Enter details of the database path in the following format: *jdbc:hsqldb:file:C:\my_databases\hsqldb\libretest;default_schema=true;shutdown=true.*
  - Replace the string “C:\my_databases\hsqldb\libretest” with the path to your database.
  - When connecting Base to an external HSQLDB database file, the connection string should include the property "*default_schema=true*", which ensures that Base only works with HSQLDB’s PUBLIC schema. Also include the property "*shutdown=true*", which ensures that the database is shut down when its last connection is closed. See the HyperSQL User Guide for more information about these properties.
- Enter the name of the driver class as *org.hsqldb.jdbc.JDBCDriver*.
- Click the **Test Class** button to check that the driver’s JAR file is accessible to Base. If successful, Base displays a confirmation dialog (Figure 105). Click the **OK** button to remove this dialog from the screen.
- At Step 3 (*Set up user authentication*) of the Database Wizard, enter *SA* in the *User name* field and make sure that the **Password required** checkbox is not ticked.

With an in-process connection to an external HSQLDB database, the Status bar of the main Base window shows the following information:

- The name of the connector used (*JDBC*).
- The path to the database file as entered at Step 2 of the Database Wizard (*hsqldb:file:C:\my_databases\hsqldb\libretest;default_schema=true;shutdown=true* in this case).
- The user name that was used to connect to the HSQLDB database (*SA* in this case).

Server / multi-user connection

For multi-user access, use one of the server modes described in the HyperSQL User Guide. In this example, the HyperSQL HSQL Server mode is used.

If you need to start the server, details of the steps required can be found in the HyperSQL User Guide. Typically this will require a shell or terminal command of the form:

```
C:\> java -cp c:/my_databases/hsqldb/lib/hsqldb.jar
org.hsqldb.server.Server --database.0 file:c:/my_databases/mydb --
dbname.0 libretest
```

The following points may help clarify this command:

1) The *java* command must lie on your shell's command search path.
2) In this example, the HSQLDB JAR package is located in the folder *c:/my_databases/hsqldb/lib*.
3) In this example, the database files are named *mydb.** and are located in the folder *c:/my_databases*.
4) The name *libretest* can be used to reference the database when Base is connected to the server.

The first time that you make a connection using the HyperSQL HSQL Server mode, carry out steps similar to those for a JDBC connection to a MySQL/MariaDB database (see Using the Database Wizard subsection on page 95), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **JDBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to HSQLDB.
- For a JDBC connection, Base displays Step 2 (*Set up JDBC connection*) of the Database Wizard (Figure 111).
- Enter details of the database and server using the following format: *jdbc:hsqldb:hsql://localhost/libretest;default_schema=true;shutdown=true*.
  - Note that there is no mention of the database file, as this was specified when running the server. Only the public name defined for *dbname.0* is needed.
  - In server modes, the *localhost* is the name describing the local computer's address. In a networked environment, this may be replaced with an IP address or computer name.
  - See the In-process, single-user connection subsection above for explanations of the *default_schema=true* and *shutdown=true* properties.
- Enter the name of the driver class as *org.hsqldb.jdbc.JDBCDriver*.
- Click the **Test Class** button to check that the driver's JAR file is accessible to Base. If successful, Base displays a confirmation dialog (Figure 105). Click the **OK** button to remove this dialog from the screen.
- At Step 3 (*Set up user authentication*) of the Database Wizard, enter *SA* in the *User name* field and make sure that the **Password required** checkbox is not ticked.

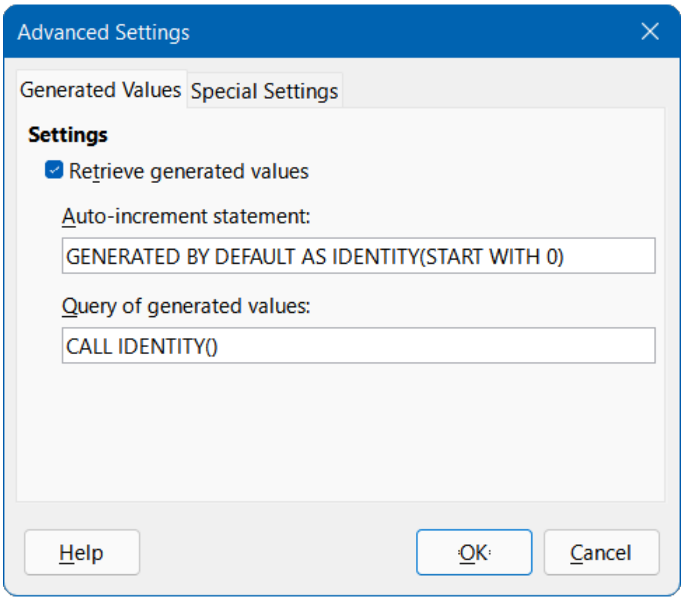
With a HyperSQL HSQL Server mode connection to an external database, the Status bar of the main Base window shows the following information:

- The name of the connector used (*JDBC*).
- The name of the database file as entered at Step 2 of the Database Wizard (*hsqldb:hsql://localhost/libretest;default_schema=true;shutdown=true* in this case).
- The user name that was used to connect to the HSQLDB database (*SA* in this case).

Possible limitations and workarounds

Auto-increment values

Before creating an auto-incrementing primary key field, select **Edit > Database > Advanced Settings** on the Menu bar and click on the *Generated Values* tab. Tick the **Retrieve generated values** checkbox, enter `GENERATED BY DEFAULT AS IDENTITY(START WITH 0)` in the *Auto-increment statement* field, enter `CALL IDENTITY()` in the *Query of generated values* field, and click the **OK** button (Figure 115).



Settings

☑ Retrieve generated values

Auto-increment statement: GENERATED BY DEFAULT AS IDENTITY(START WITH 0)

Query of generated values: CALL IDENTITY()

Figure 115: Advanced Settings dialog, Generated Values tab for external HSQLDB connection

These adjustments may enable you to successfully define an auto-incrementing primary key field. If Base displays an error message indicating a NOT NULL check constraint error when entering table data, then this can often be corrected by creating a primary key field without an auto value and then executing the command `ALTER TABLE "table_name" ALTER COLUMN "ID" INT GENERATED BY DEFAULT AS IDENTITY(START WITH 0)` via the **Tools > SQL** option. *table_name* and *ID* are the names of the table and primary key field respectively.

No access to tables

Sometimes a HSQLDB database can be open in Base but there is no access to the tables. In this case we recommend selecting **Tools > SQL** on the Menu bar and entering the `SHUTDOWN SCRIPT` command. Hopefully, subsequently closing and then re-opening the database will clear the problem, although this may not be successful if an “error in script file” has already been reported.

SQLite databases

Overview

SQLite is an open-source, embedded SQL database engine. Unlike the database systems covered in earlier sections of this chapter, SQLite does not have a separate server. SQLite databases consist of a single file that may be stored anywhere in the file system and can be connected to Base using an appropriate driver. More information can be found on the SQLite website at https://www.sqlite.org/. Extensive documentation about SQLite can be found at https://www.sqlite.org/docs.html.

Tip

SQLite supports a limited subset of SQL’s ALTER TABLE command. Take care when designing tables because it is difficult to modify a column’s data type once the table has been created.

Base can connect to an external SQLite database file using an ODBC driver. The user interactions necessary to establish such a connection have some commonality with the interactions for connecting to a MySQL or MariaDB database. However, there are also significant differences. The aim of the following subsections is to highlight and explain those differences, without repeating details that are identical to those already described for MySQL/MariaDB.

Base does not currently provide a direct connection to a SQLite database file.

The developers of SQLite do not supply a downloadable JDBC driver. Although there are several SQLite JDBC drivers available on the Internet, we are not currently aware of one that will support a reliable connection between Base and SQLite.

Initial preparations before connecting to a SQLite file

Before making a connection, the SQLite database file of interest must exist. If your database file does not exist already then you can create an empty file in several ways:

- Use one of the many options provided by your operating system. For example, on Windows you could use *File Explorer* to navigate to the folder where you want to place the file, right-click, select **New**, select **Text Document**, and enter the name of the file. The Windows command prompt also provides commands that, when used imaginatively, are able to create empty files (for example, *type nul > libretest.db*). Equivalent facilities are available on Linux (for example, > *libretest.db*).
- Use a plain text editor, such as *Notepad* on Windows, *Vim* on Linux, or *TextEdit* on macOS. Make sure that the size of the file is zero bytes after creation.
- Use the *sqlite3* command line shell for SQLite, which may be installed when a driver is installed or can be downloaded (along with other tools) from https://www.sqlite.org/download.html.

Detailed documentation for *sqlite3* can be found at https://www.sqlite.org/cli.html. The following commands should suffice to create a new database file named *libretest.db*:

1) At a shell, terminal, or operating system prompt, enter:
```
sqlite3 libretest.db
```
2) At the *sqlite>* prompt, enter:
```
.databases
```
sqlite3 provides feedback of the exact location of the file that has been created.
3) At the *sqlite>* prompt, enter:
```
.exit
```
sqlite3 closes.

SQLite is not prescriptive about the extension of a database file – many users will utilize “.sqlite”, others will choose “.db”, but it is a personal decision.

SQLite does not provide username and password control for a database file.

Connecting with ODBC

Initial preparation

To use Base to connect to a SQLite database file through the ODBC API, there should be a properly configured ODBC driver installed on your computer. If the driver is not currently available on your computer, then you will need administration rights to install the new software.

The developers of SQLite do not supply a downloadable ODBC driver. However there are several such drivers available on the Internet, some of which are open-source and others may be proprietary. One such driver is available on Christian Werner’s website at http://ch-werner.de/sqliteodbc/, where it is stated that the driver has been tested on Windows and Linux. Platform-dependent installation instructions can be found in the SQLiteODBC Documentation at http://ch-werner.de/sqliteodbc/html/index.html.

Note

At the time of drafting this section, the developer of the driver does not make builds available for macOS. However other developers have contributed several macOS builds for selected legacy versions of the driver.

Make sure that you follow the instructions given in the driver documentation to create an ODBC Data Source Name (DSN) for your database. For the purposes of the example in the next subsection, assume that a data source has been configured as follows:

- The name of the data source is set to *libretest-SQLite*.
- The data source includes the database name as *C:\my_databases\libretest.db*.
- Login credentials for the database server are not defined within the data source.

Using the Database Wizard

The first time that you make an ODBC connection to a particular SQLite database file, carry out steps similar to those for an ODBC connection to a MySQL/MariaDB database (Using the Database Wizard subsection on page 91), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **ODBC** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) of the Database Wizard when making a connection to a MySQL/MariaDB database are not relevant to SQLite.
- At Step 2 (*Set up ODBC connection*), enter the name of the previously defined ODBC data source that is to be used to access the SQLite database (*libretest-SQLite* in our example). You can click the **Browse** button to access the Data Source dialog (Figure 101), where you can select the data source of interest from a list of those available and then click the **OK** button.
- Do not enter any information at Step 3 (*Set up user authentication*) of the Database Wizard.

When the connection is established and the ODB file has been created, Base updates its main window to show information about the SQLite database file.

With an ODBC connection to a SQLite database file, the Status bar of the main Base window shows the following information:

- The name of the connector used (*ODBC*).
- The name of the ODBC data source (*libretest-SQLite* in this case).

dBASE databases

Connecting to a dBASE database

dBASE databases utilize the DBF file type, which is a standard format for storing structured data. Base can connect to a database comprising one or more DBF files located in a single folder. Each DBF file should have the extension “.dbf” (in lowercase on systems where file names are case-sensitive) and stores the data for one database table, including both field descriptors and data records. Tables defined in DBF files are conceptually similar to sheets in Calc.

There are no intrinsic links / relations between fields in different tables – this means, for example, that if an employee leaves your company and you delete her details from the *employee* table, she might still receive next month's salary if you forget to also delete her from the *salary* table! If such links are required for a connected dBASE database, you can create them using Base facilities (such as forms, queries, and macros).

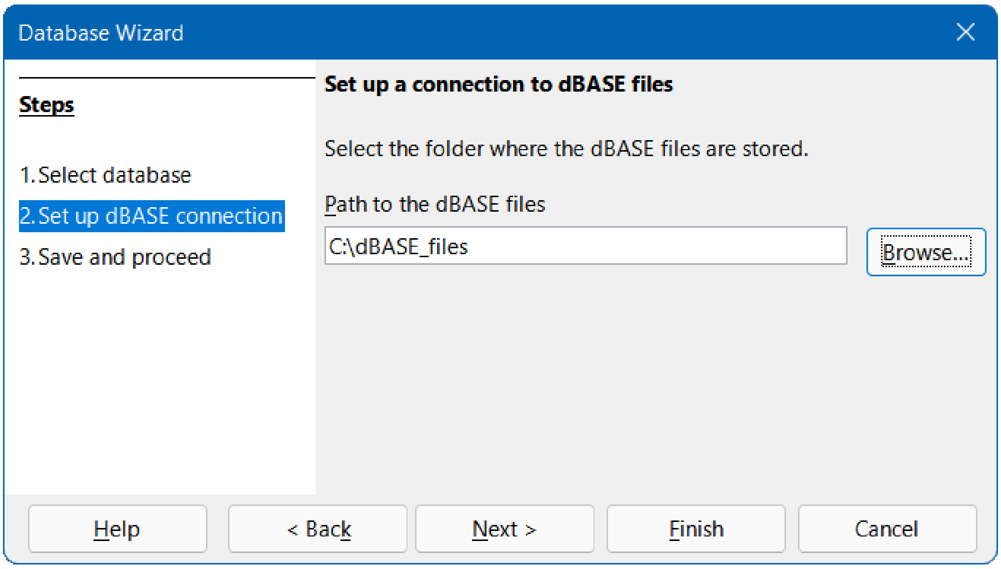
Tip

LibreOffice Calc can save a spreadsheet as a DBF file (although formulas and formatting are lost) and can read a DBF file into a spreadsheet. The latter facility may be helpful on occasions when you need to quickly view the data in a dBASE table that is not already connected in Base.

The first time that you make a connection to a particular dBASE database, carry out steps similar to those for a direct connection to a MySQL/MariaDB database (see Connecting directly subsection on page 86), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **dBASE** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) and Step 3 (*Set up MySQL server data*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to dBASE. Instead, Base displays Step 2 (*Set up dBASE connection*) of the Database Wizard (Figure 116).

  Enter the location of the folder containing your dBASE files. You can click the **Browse** button to access the Select Folder dialog, where you can navigate to the folder of interest and then click the **Select Folder** button.



Database Wizard

Steps

1. Select database
2. Set up dBASE connection
3. Save and proceed

Set up a connection to dBASE files

Select the folder where the dBASE files are stored.

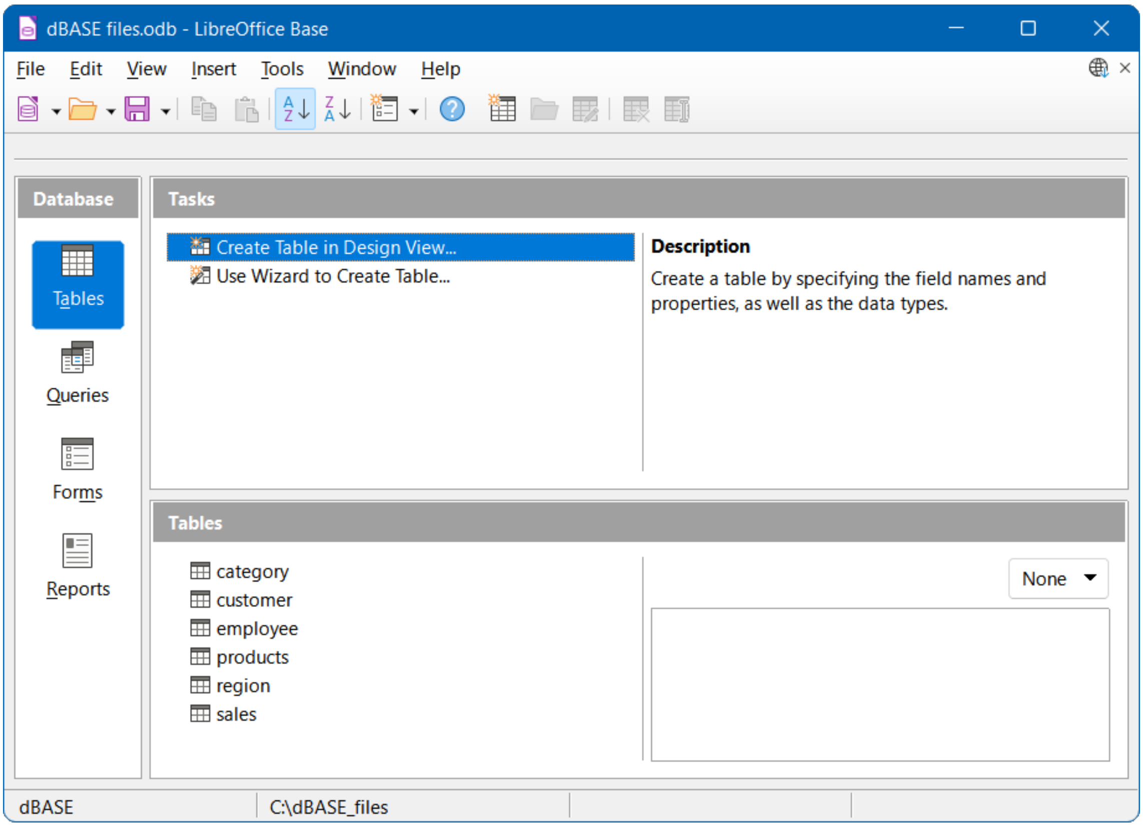
Path to the dBASE files: C:\dBASE_files

Figure 116: Step 2 of Database Wizard: Set up dBASE connection

- The options presented on Step 4 *(Set up user authentication*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to dBASE.

When the connection is established and the ODB file has been created, Base updates its main window to show information about the dBASE database (Figure 117), including all DBF files in the folder. In the case shown, Base has found six DBF files in the given folder and these are named:

- category.dbf
- customer.dbf
- employee.dbf
- products.dbf
- region.dbf
- sales.dbf


dBASE files.odb - LibreOffice Base
File
Edit
View
Insert
Tools
Window
Help
Database
Tables
Queries
Forms
Reports
Tasks
Create Table in Design View...
Use Wizard to Create Table...
Description
Tables
None

Create a table by specifying the field names and properties, as well as the data types.

category
customer
employee
products
region
sales

dBASE
C:\dBASE_files

Figure 117: Tables in a dBASE file

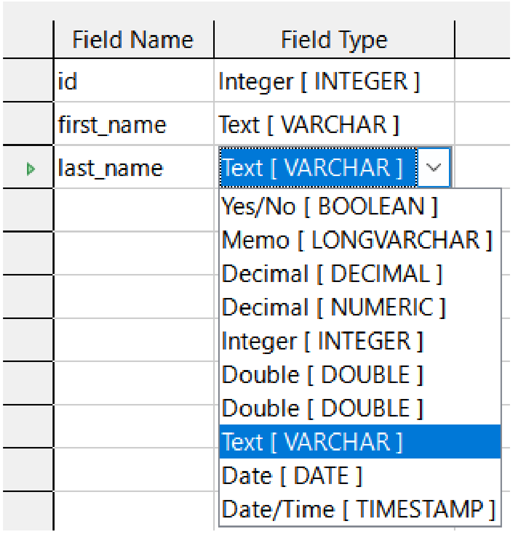
With a connection to a dBASE database, the Status bar of the main Base window shows the following information:

- The name of the connector used (*dBASE*).
- The folder containing the database files (C:*\dBASE_files* in this case).

New tables can be created through the Base user interface and these will appear as additional DBF files in the specified folder.

Possible limitations and workarounds

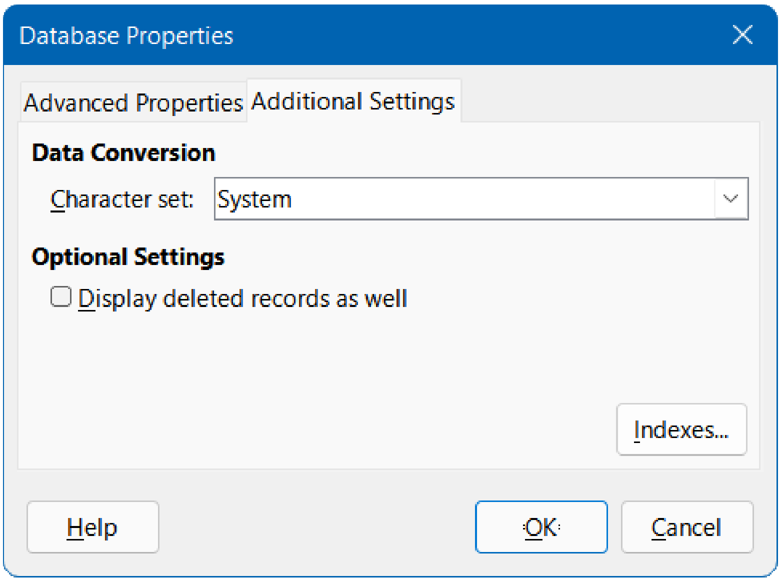
1) The DBF file format does not support the concept of declaring a table column as a primary or foreign key.
2) The DBF file format limits the length of field names in tables to ten characters.
3) The number of different field type options available when creating a new dBASE table is significantly less than when creating tables for most other databases (for example, when creating a table in an embedded HSQLDB database). This can be seen in Figure 118.



Field Name	Field Type
id	Integer [INTEGER]
first_name	Text [VARCHAR]
last_name	Text [VARCHAR]

Figure 118: Field types available when creating a new dBASE table

4) By default, Base adopts the character encoding of the operating system. It is therefore possible for legacy dBASE files to develop errors when special characters are imported. The character set can be corrected using the *Character set* menu on the *Additional Settings* tab of the Database Properties dialog (Figure 119), which is accessed by selecting **Edit > Database > Properties** on the Menu bar.



Database Properties

Data Conversion

Character set: System

Optional Settings

☐ Display deleted records as well

Figure 119: Database Properties dialog, Additional Settings tab – changing the character set

5) When copying a dBASE table into another database (for example, an embedded HSQLDB database), the import wizard has problems with automatic recognition of numeric field types and Yes/No fields (Bug 53027). This may require you to make subsequent corrections.

Spreadsheets and Writer documents

Overview

Base can connect to an existing spreadsheet or word processor document, using tabular data from the connected source to populate the table(s) of a database. This method of access does not allow you to modify any data in the connected document. However, Base can be used to view information, run queries, and create reports based on data that is already contained in the connected document. Base can connect to Calc spreadsheets, Microsoft Excel spreadsheets, Writer documents, and Microsoft Word documents.

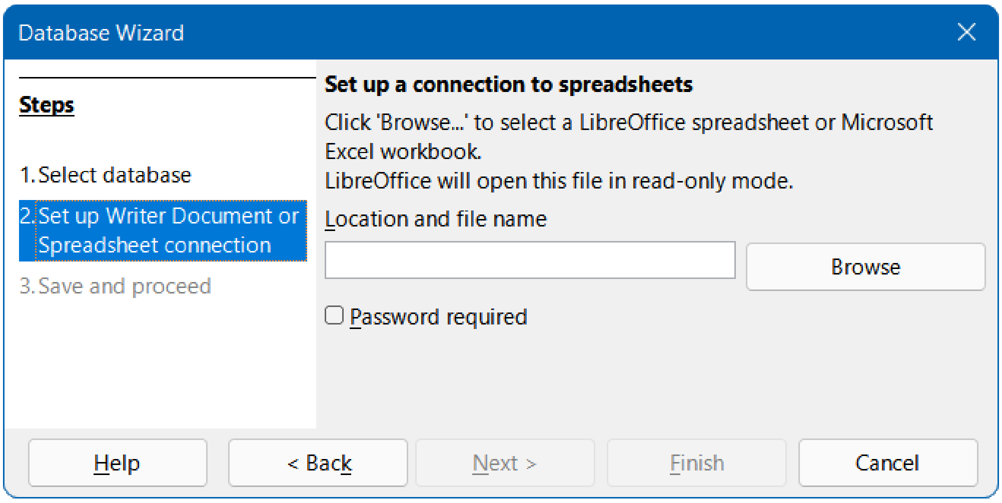
Connect to spreadsheet

The first time that you make a connection to a particular Calc or Microsoft Excel spreadsheet, carry out steps similar to those for a direct connection to a MySQL/MariaDB database (see Connecting directly subsection on page 86), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **Spreadsheet** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) and Step 3 (*Set up MySQL server data*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to a **Spreadsheet** connection. Instead, Base displays Step 2 (*Set up Writer Document or Spreadsheet connection*) of the Database Wizard (Figure 120).

  Enter the location and name of the spreadsheet file. You can click the **Browse** button to access a file explorer dialog, where you can navigate to the file of interest and then click the **Open** button.

  If the spreadsheet of interest is password-protected, click the **Password required** option.



Database Wizard

Set up a connection to spreadsheets

Click 'Browse...' to select a LibreOffice spreadsheet or Microsoft Excel workbook.
LibreOffice will open this file in read-only mode.

Location and file name

□ Password required

Figure 120: Step 2 of Database Wizard: Set up Writer Document or Spreadsheet connection

Base processes the spreadsheet data as follows:

- One database table is produced for each sheet in the spreadsheet, with a database table given the same name as its corresponding sheet.
- One database table is produced for each database range defined in a Calc spreadsheet, with a database table given the same name as its corresponding database range. This feature is not applicable to “standard” named ranges in Calc spreadsheets or to Microsoft Excel spreadsheets.

In addition, Base makes unavailable interactions to create tables, views, relationships, and queries that cross multiple tables.

When the connection is established and the ODB file has been created, Base updates its main window to show information from the spreadsheet.

With a connection to a spreadsheet, the Status bar of the main Base window shows the following information:

- The name of the connection type (*Spreadsheet*).
- The location and name of the connected spreadsheet (as entered at Step 2 of the Database Wizard).

At some later time when you open this ODB file again and then use Base to access the table data, the Authentication Required dialog (similar to that shown in Figure 99) may be displayed, depending whether you selected the **Password required** option at Step 2 of the Database Wizard. You do not need to reconfigure the connection but you do need to reenter your correct login credentials when requested.

Tip

It is possible for a Calc spreadsheet to display external data, sourced from the Internet – see Chapter 10, Linking Data, in the *Calc Guide* for more details. This Internet data can in turn be accessed in Base by connecting to the relevant spreadsheet.

Connect to word processor document

The first time that you make a connection to a particular Writer or Microsoft Word document, carry out steps similar to those for a connecting to a spreadsheet (see Connect to spreadsheet subsection on page 122), but with the following difference:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **Writer Document** option in the drop-down menu (instead of **Spreadsheet**).

Note that the appearance of Step 2 (*Set up Writer Document or Spreadsheet connection*) of the Database Wizard (Figure 120), is the same whether **Spreadsheet** or **Writer Document** is selected at Step 1.

Base processes the data in the document as follows:

- One database table is produced for each table in the document, with a database table given the same name as its table in the document. In Writer, table names may be viewed and modified through the Navigator, which can be accessed by selecting **View > Navigator** on the Menu bar.
- Data in the document that is not in a table is ignored.

In addition, Base makes unavailable interactions to create tables, views, relationships, and queries that cross multiple tables.

When the connection is established and the ODB file has been created, Base updates its main window to show information from the document.

With a connection to a document, the Status bar of the main Base window shows the following information:

- The name of the connection type (*Writer Document*).
- The location and name of the connected document (as entered at Step 2 of the Database Wizard).

At some later time when you open this ODB file again and then use Base to access the table data, the Authentication Required dialog (similar to that shown in Figure 99) may be displayed, depending whether you selected the **Password required** option at Step 2 of the Database Wizard. You do not need to reconfigure the connection but you do need to reenter your correct login credentials when requested.

Editing spreadsheet data with both Calc and Base open

Data in a connected Calc spreadsheet cannot be edited in Base. The following steps indicate how to modify spreadsheet data with both a Base database and its connected Calc spreadsheet open simultaneously.

1) Make sure that the associated spreadsheet is NOT open and then open the Base file.
2) Open the associated spreadsheet. Calc opens the file with read-only protection – note the message in the upper part of the Calc window stating “This document is open in read-only mode”. Click the adjacent **Edit Document** button.
3) Open a table of interest in Base. If existing data is changed in the corresponding area within Calc, select **Data > Refresh** on the Menu bar of the Base Table Data View window to see the changed data in the Base file.
4) If additional data is added to the corresponding area within Calc, the previous step may not show the new data in Base. In this case close the Base Table Data View window and select **View > Refresh Tables** on the Menu bar of the main Base window. Then, open the table again and the new data should be visible.

Note that steps 3) and 4) above change the spreadsheet data without saving those changes. Extra steps would be required to save the modified Calc file with the same name and in its original location.

A similar strategy can be used to change the data in a Writer file while viewing the changes in Base.

Text databases

Overview

There may be occasions when you need to read data into a database, but find that the data is not in a form directly supported by the drivers provided in Base. Such data can often be converted into CSV (comma-separated values) or other suitable text format, possibly via Calc. Base provides two different methods for including data from text files in database tables:

1) If you are using an embedded or external HSQLDB database, then you can take advantage of the HSQLDB-specific facility to create Text Tables.
2) Alternatively, you can connect to an existing set of text files using the Database Wizard.

These options are described in the following subsections.

HSQLDB Text Tables

HSQLDB Text Tables are CSV or other delimited files treated as SQL tables. A Text Table of this kind cannot be created directly through the Base graphical user interface but instead is created by issuing a `CREATE TEXT TABLE` command via the **Tools > SQL** option. For example:

```
CREATE TEXT TABLE "company_data" (
  "Date" DATE PRIMARY KEY,
  "Open" DOUBLE,
  "High" DOUBLE,
  "Low" DOUBLE,
  "Close" DOUBLE,
  "Adj Close" DOUBLE,
  "Volume" BIGINT
);
```

creates an empty Text Table designed to contain historic data about a company’s share price.

This is followed by a `SET TABLE` command to specify the file and separator character that the Text Table uses. For example:

```
SET TABLE "company_data"
SOURCE "share_prices.csv;encoding=UTF-8;ignore_first=true";
```

links the file *share_prices.csv* to the *company_data* Text Table. This command assumes that the CSV file is in the same folder as the database.

For more information, see the Text Tables chapter of the HyperSQL User Guide (https://hsqldb.org/doc/2.0/guide/guide.pdf).

The following factors should be noted when using HSQLDB Text Tables within Base:

1) After creating a Text Table, you may need to select **View > Refresh Tables** on the Menu bar before the new table appears in the *Tables* area of the main Base window. In the *Tables* area, the appearance of the icon for a Text Table is different to that for a “normal” table.
2) It is possible for the underlying text file to be open (and edited) simultaneously by other external programs, such as spreadsheet or text editing applications. This behavior could cause confusion and loss of data.
3) When a record is modified in Base, the order of records in the underlying data file will be altered. Specifically, the original location of the record is emptied and the updated record is appended at the end of the file.
4) Be aware of the warning in the HyperSQL User Guide that Text Tables are not as resilient to machine crashes as other types of tables. This is because other table types incorporate additional mechanisms to maintain the integrity of data.

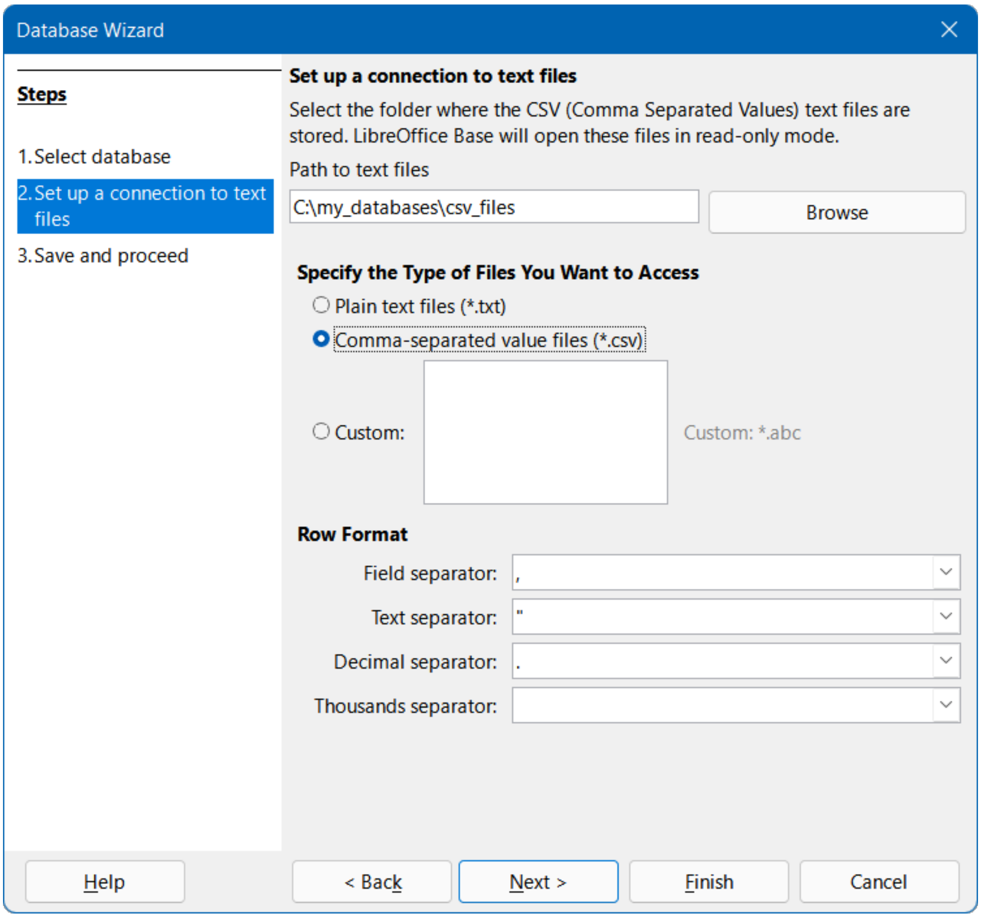
Connecting to text files

In Base you can create a complete database by accessing a set of text files in a folder. These files will often be in CSV format but this is not essential, although they must all utilize the same format. The files will be opened read-only, with no write access.

While text databases of this form are often not suitable for general use, they can be used to quickly search in a CSV file for records of interest, or to import a CSV file into another database by using the copy function. A CSV file need only be moved into the specified folder and then it can be easily searched or copied.

The first time that you make a connection to a particular text database, carry out steps similar to those for a direct connection to a MySQL/MariaDB database (see Connecting directly subsection on page 86), but with the following important differences:

- After clicking the **Connect to an existing database** option on Step 1 (*Select database*) of the Database Wizard, select the **Text** option in the drop-down menu (instead of **MySQL**).
- The options presented on Step 2 (*Set up MySQL connection*) and Step 3 (*Set up MySQL server data*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to a text database. Instead, Base displays Step 2 (*Set up a connection to text files*) of the Database Wizard (Figure 121).



Set up a connection to text files

Select the folder where the CSV (Comma Separated Values) text files are stored. LibreOffice Base will open these files in read-only mode.

Path to text files: C:\my_databases\csv_files

Specify the Type of Files You Want to Access

○ Plain text files (*.txt)

● Comma-separated value files (*.csv)

○ Custom: Custom: *.abc

Row Format

Field separator: ,

Text separator: "

Decimal separator: .

Thousands separator:

Figure 121: Step 2 of Database Wizard: Set up a connection to text files

- Enter the location of the folder containing your text files. You can click the **Browse** button to access the Select Folder dialog, where you can navigate to the folder of interest and then click the **Select Folder** button.
- Select the type of file to be used, either **Plain text files (*.txt)**, **Comma-separated value files (*.csv)**, or **Custom**. In the latter case, specify the file extension to be used.
- Adjust (where necessary) the four separators used in the rows of all the selected files.

- The options presented on Step 4 (*Set up user authentication*) of the Database Wizard when making a direct connection to a MySQL/MariaDB database are not relevant to text databases.

When the connection is established and the ODB file has been created, Base updates its main window to show information about the text database. The *Tables* area will list the tables created, one for each file of the correct type located in the specified folder. The name of the table will be set to the corresponding file name (without extension). In addition, Base makes unavailable interactions to create tables, views, relationships, and queries that cross multiple tables.

With a connection to a text database, the Status bar of the main Base window shows the following information:

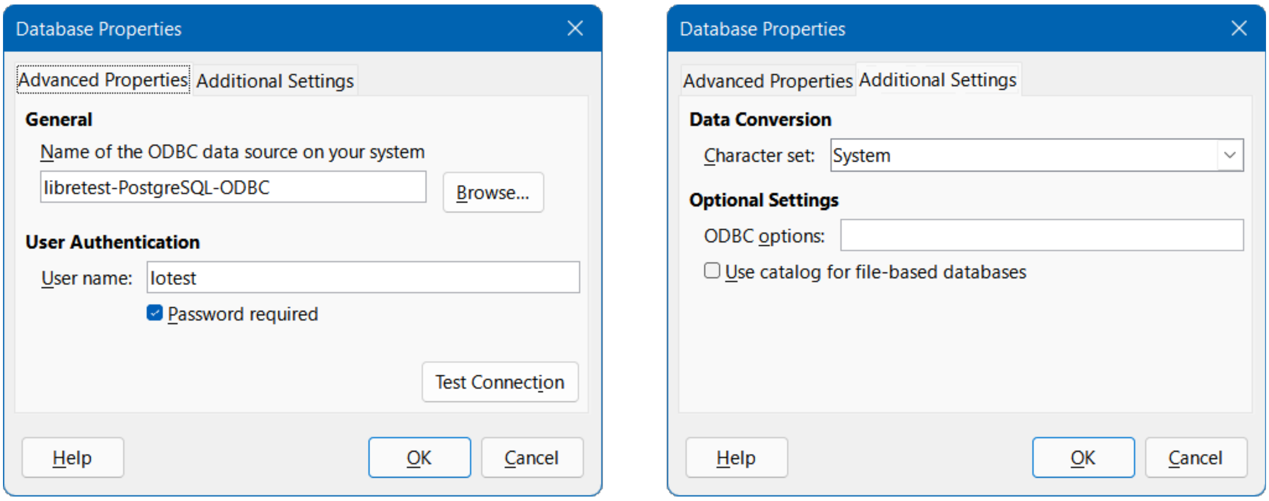
- The name of the connector used (*Text*).
- The folder containing the database files (*C:\my_databases\csv_files* in this case).

Subsequent editing of connection properties

In most cases, the instructions given in previous sections of this chapter will result in a connection to an external database that operates as expected. However, occasionally the connection may not function as intended. For example, something may have changed in the basic connection parameters (such as the connection string, driver name, or database user name); a different character set may be needed; or subforms may not work correctly. Changes can be made to the connection properties by selecting **Edit > Database** on the Menu bar and then selecting one of three options:

1) **Properties**. The Database Properties dialog enables changes to the properties of the current connection, but does not support selection of an alternative connection type. It provides one or more tabs where adjustments can be made to connection parameters, most of which were originally configured through the Database Wizard.
2) **Connection Type**. Invokes the Database Properties Wizard which enables switching to a different connection type. For example, you can switch from an ODBC connection to a JDBC or direct connection. This may be useful if you want to test which connection type best suits your application. The options presented at each step are similar to those in the Database Wizard.
3) **Advanced Settings**. The Advanced Settings dialog provides one or more tabs containing controls to adjust various settings that are not configured when using the Database Wizard. The available options depend on database and connection type. Depending on the database system, there are different commands to automatically generate incrementing values – the *Generated Values* tab can be used to control the handling of automatically generated values for new data records – for more information, search for “Generated Values” in the Base module of the Help system. The *Special Settings* tab enables the specification of various detailed options that influence how Base interacts with the database – for more information, search for “Special Settings” in the Base module of the Help system.

Figure 122 provides a typical example of the Database Properties dialog for an ODBC connection to a PostgreSQL database. In this case the dialog provides two tabs, *Advanced Properties* and *Additional Settings*. On the *Advanced Properties* tab are controls to select an alternative ODBC data source or different login credentials. The *Additional Settings* tab enables selection of another character set and entry of optional ODBC settings.



General

Name of the ODBC data source on your system: libretest-PostgreSQL-ODBC

User Authentication

User name: lotest

☑ Password required

Data Conversion

Character set: System

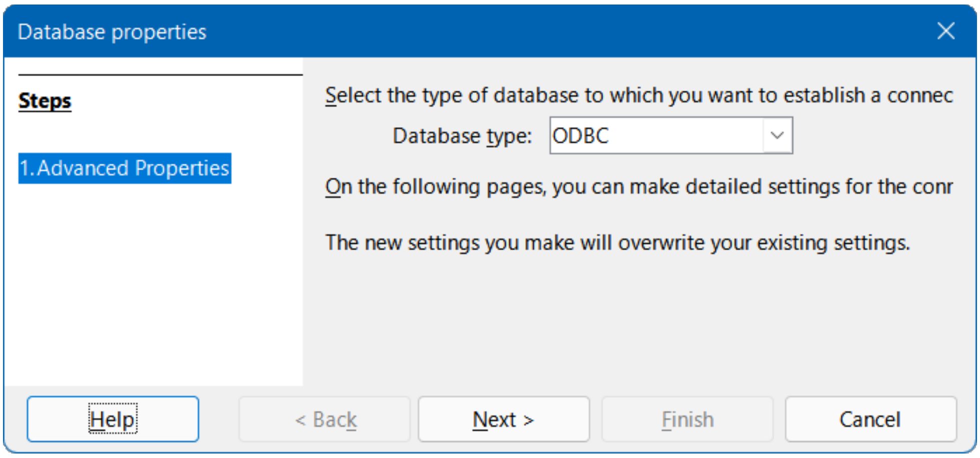
Optional Settings

ODBC options:

☐ Use catalog for file-based databases

Figure 122: Database Properties dialog – Advanced Properties tab (left) and Additional Settings tab (right)

Figure 123 shows the first step of the Database Properties Wizard. By default the *Database type* drop-down is set to reflect the current connection type (**ODBC** in this example). Choose a different option from the menu to change the connection type and then click the **Next** button to start filling out the parameters of the new connection type, similar to the corresponding steps in the Database Wizard.



Database properties

Steps

1.Advanced Properties

Select the type of database to which you want to establish a connec

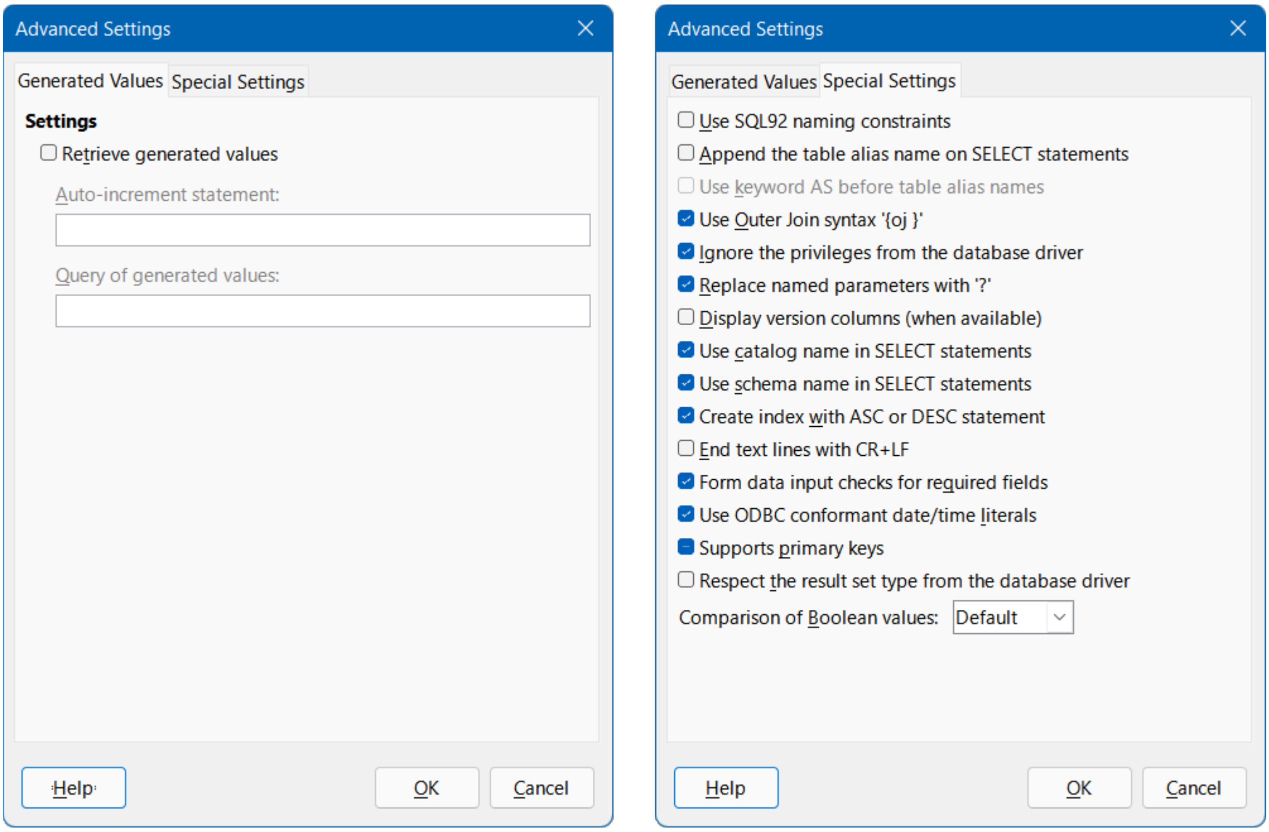
Database type: ODBC

On the following pages, you can make detailed settings for the conr

The new settings you make will overwrite your existing settings.

Figure 123: Step 1 of Database Properties Wizard: Advanced Properties

Figure 124 provides an example of the Advanced Settings dialog, for an ODBC connection to a PostgreSQL database. In this case the dialog provides two tabs, *Generated Values* and *Special Settings*. Note that some options on the *Special Settings* tab may be unavailable (grayed) if they cannot be changed for the underlying database (for example, the **Use keyword AS before table alias names** option in Figure 124).



Advanced Settings

Generated Values | Special Settings

Settings

☐ Retrieve generated values

Auto-increment statement:

Query of generated values:

Advanced Settings

Generated Values | Special Settings

☐ Use SQL92 naming constraints
☐ Append the table alias name on SELECT statements
☐ Use keyword AS before table alias names
☑ Use Outer Join syntax '{oj }'
☑ Ignore the privileges from the database driver
☑ Replace named parameters with '?'
☐ Display version columns (when available)
☑ Use catalog name in SELECT statements
☑ Use schema name in SELECT statements
☑ Create index with ASC or DESC statement
☐ End text lines with CR+LF
☑ Form data input checks for required fields
☑ Use ODBC conformant date/time literals
■ Supports primary keys
☐ Respect the result set type from the database driver

Comparison of Boolean values: Default

Figure 124: Advanced Settings dialog – Generated Values tab (left) and Special Settings tab (right)

There may be some advanced settings that cannot be changed through the Base graphical user interface. However, in some circumstances, an experienced user will be able to edit the *content.xml* file within the database’s ODB file to fix the problem. Here is an example that caused difficulties when migrating from LibreOffice 6.0 to LibreOffice 6.1: Subqueries in MySQL were no longer working as expected because the transfer of the connecting value (parameter) was prevented. This could be remedied by opening the *content.xml* file and locating the code:

```
<db:driver-settings db:system-driver-settings="" db:base-dn=""
db:parametername-substitution="false"/>
```

Changing this line to:

```
<db:driver-settings db:system-driver-settings="" db:base-dn=""/>
```

cured the problem that existed at the time. Alternatively, a macro could have been written to achieve the same result.

Base Guide 7.3

Chapter 3
Tables

General information on tables

Databases store data in tables. The main difference from the tables in a database and a range of cells in a simple spreadsheet is that the data type of the fields must be clearly defined beforehand. For example, a database does not allow a text field to contain numbers for use in calculations. Such numbers are displayed, but only as strings, whose actual numerical value is zero. Similarly, images cannot be included in all types of fields.

Details of which data types are available can be obtained from the Table Design window in Base. They are shown in the Appendix A to this book.

Simple databases are based on only one table. All data elements are entered independently, which can lead to multiple entry of the same data. A simple address book for private use can be created in this way. However, the address book of a school or a sports association could contain so much repetition of postcodes and locations that these fields are better placed in one or even two separate tables.

Storing data in separate tables helps:

- Reduce repeated input of the same content
- Prevent spelling errors due to repeated input
- Improve filtering of data in the displayed tables

When creating a table, you should always consider whether multiple repetitions, especially of text or images (which consume a lot of storage) may occur in the table. If so, you need to export them into another table. How to do this in principle is described in Chapter 1, Introduction to Base, in the section “A Simple Database – Test Example in Detail”.

Note

A relational database is a group of tables which are linked to one another through common attributes. The purpose of a relational database is to prevent duplicate entry of data elements as much as possible. Redundancies are to be avoided.

This can be achieved by:

– Separating content into as many unique fields as practical (for example, instead of using one field for a complete address, use separate fields for house number, street, city and postcode).

– Preventing duplicate data for one field in multiple records (for example by importing the postcode and city from another table).

These procedures are known as *Database Normalization*.

Relationships between tables

This chapter explains many of these steps in detail, using an example database for a library: `Media_without_Macros`. Constructing the tables for this database is an extensive job, as it covers not only the addition of items into a media library but also the subsequent loan of them.

Relationships for tables in databases

Tables in the internal database HSQLDB always have a distinctive, unique field, the *primary key*. This field must be defined before any data can be written into the table. Using this field, specific records in a table can be found.

In certain cases, a primary key is formed from several fields in combination. These fields must be unique when considered together. This is known as a *composite primary key*.

Note

The combination of the fields in a composite primary key is unique when each record of the table contains a unique combination of values for these fields.

Suppose there are two tables, Table1 and Table2. Table 2 may have a field that indicates a record in Table 1. Here, the primary key of Table 1 is written as a value in the Table 2 field. Table 2 now has a field that points to another table’s key field, known as a *foreign key*. This foreign key exists in Table 2 in addition to its primary key.

The more relationships there are between tables, the more complex is the design task. Figure 125 shows the overall table structure of this example database, scaled to fit the page size of this document. To read the content, zoom the page to 200%.

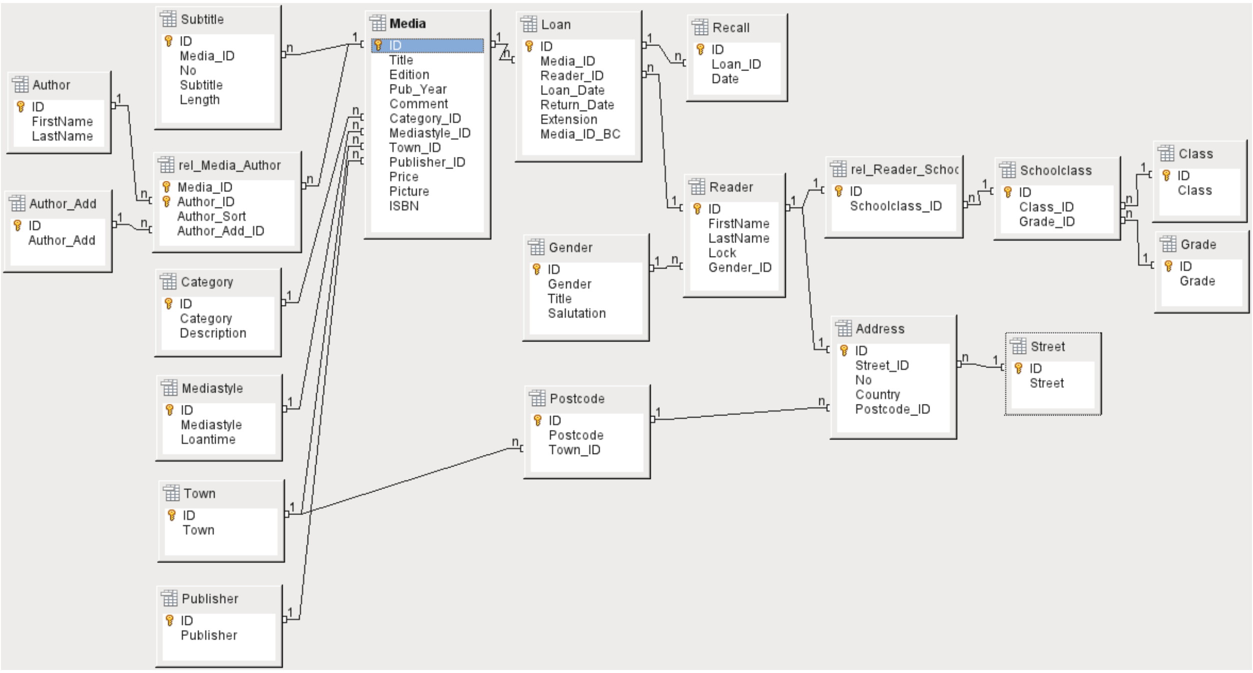


Figure 125: Relationship diagram for the Media_without_Macros database

One-to-many relationships

The `Media_without_Macros` database lists the titles of the media in one table. Because titles can have multiple subtitles or sometimes none at all, the subtitles are stored in a separate table.

A relationship in which a field can have more than one foreign key is known as a *one-to-many* (1:n) realtionship. Many subtitles may be assigned to one medium, for example the many track titles for a music CD. The primary key for the Media table is stored as a foreign key in the Subtitle table. Most relationships between tables in a database are one-to-many relationships.

Many-to-many relationships

A database for a library might contain a table for authors’ names and a table for the media. The connection between an author and, for example, books that author has written, is obvious. The library might contain more than one book by one author. It might also contain books with multiple authors. This relationship in which both tables may have more than one foreign key is known as *many-to-many* (n:m). Such relationships require a table that acts as an intermediary between the two tables concerned. This is represented in Figure 126 by the rel_Media_Author table.

Thus, in practice, the n:m relationship is solved by treating it as two 1:n relationships. In the intermediate table, the Media_ID can occur more than once, as can the Author_ID. But when using them as a pair, there is no duplication: no two pairs are identical. So this pair meets the requirements for the primary key for the intermediate table.

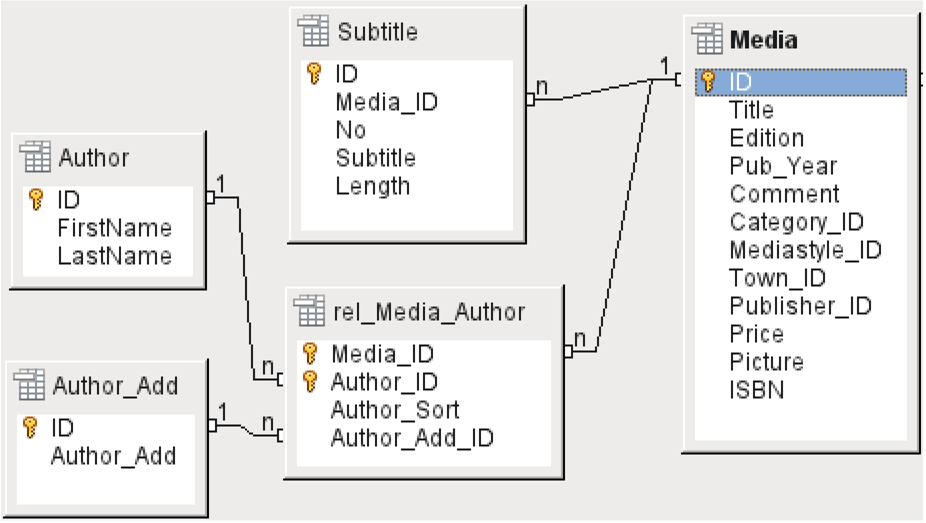


Figure 126: Example 1:n relationship; n:m relationship

Note

For a given value of Media_ID, there is only one title of the media and one ISBN. For a given value for Author_ID, there is only one Author's first and last name. So, for a given pair of these values, there is only one ISBN and only one Author. This makes the pair unique.

One-to-one relationships

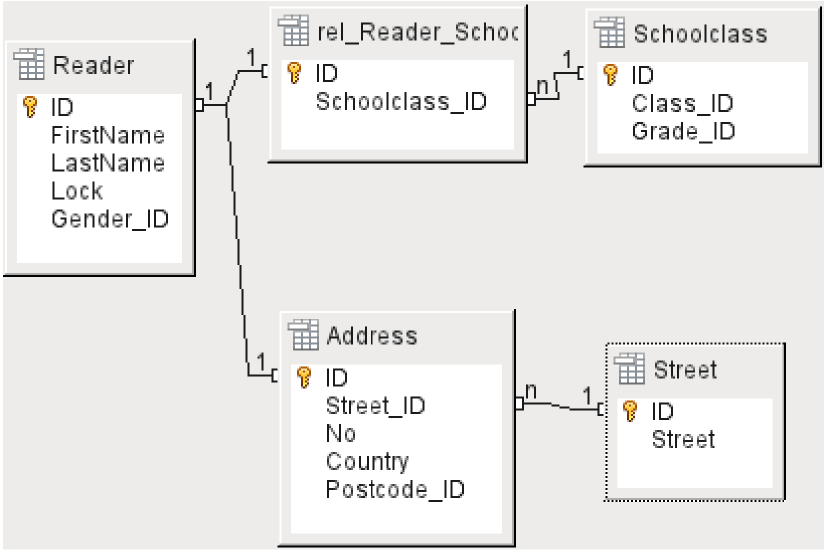


Figure 127: Example 1:1 relationship

In the Reader table, only the fields that are directly necessary were planned in advance. But our database covers two cases: a school library and a public library. For a school library database, the school class is also required with the Reader. From the school class records, you can find borrowers' addresses where necessary. Therefore it is not necessary to include these addresses in the school library database. The school class relationship of pupils is also separated from the reader table, because mapping to classes is not appropriate in all areas. There are 1:1 relationships between the reader and the address, and the reader and the class assignment.

In a database for a public library, the addresses of readers are required. For each reader there is a single address. If there are multiple readers at the same address, this structure would require the address to be entered again, since the primary key of the Reader table is entered directly as the primary key in the Address table. Primary key and foreign key are one and the same in the Address table. This is therefore a 1:1 relationship.

A 1:1 relationship does not signify that for every record in a table, there will be a corresponding record in another table. But **at most** there will be only one corresponding record. A 1:1 relationship therefore leads to fields being exported which will be filled with content for only some of the records.

Tables and relationships for the example database

The example database (Media_without_Macros) must satisfy three requirements: media additions and removals, loans, and user administration.

Media addition table

First, media must be added into the database so that a library can work with them. However, for a simple summary of a media collection, you could create easier databases with the wizard; that might be sufficient for home use.

The central table for media addition is the Media table (see Figure 128).

In this table all fields that are directly entered are assumed not to be also in use for other media with the same content. Duplication should therefore be avoided.

For this reason, planned fields in the table include the title, the ISBN, an image of the cover, and the year of publication. The list of fields can be extended if required. So, for instance, librarians might want to include fields for the size (number of pages), the series title, and so on.

The Subtitle table contains the detailed content of CDs. As a CD can contain several pieces of music, a record of the individual pieces in the main table would require a lot of additional fields (Subtitle 1, Subtitle 2, etc.) or the same item would have to be entered many times. The Subtitle table therefore stands in a n:1 relationship to the Media table.

The fields of the Subtitle table are (in addition to the subtitle itself) the sequence number of the subtitle and the duration of the track. The Length field must first be defined as a time field. In this way, the total duration of the CD can be calculated and displayed in a summary if necessary.

The authors have a n:m relationship to the media. One item can have several authors, and one author might have created several items. This relationship is controlled by the rel_Media_Author table. The primary key of this linking table is the foreign key, formed from the Author and Media tables. The rel_Media_Author table includes an additional sorting (Author_Sort) of authors, for example by the sequence in which they are named in the book. In addition, a supplementary label such as Producer, Photographer, and so on is added to the author where necessary.

Category, Mediastyle, Town, and Publisher have a 1:n relationship.

For the *Category*, a small library can use something like Art or Biology. For larger libraries, general systems for libraries are available. These systems provide both abbreviations and complete descriptions. Hence both fields appear under Category.

The *Mediastyle* is linked to the loan period Loantime. For example, video DVDs might on principle have a loan period of 7 days, but books might be loaned for 21 days. If the loan period is linked to any other criteria, there will be corresponding changes in your methodology.

The *Town* table serves not only to store location data from the media but also to store the locations used in the addresses of users.

Since *Publishers* also recur frequently, a separate table is provided for them.

The Media table has in total four foreign keys and one primary key, which is used as a foreign key in two tables, as shown in Figure 128.

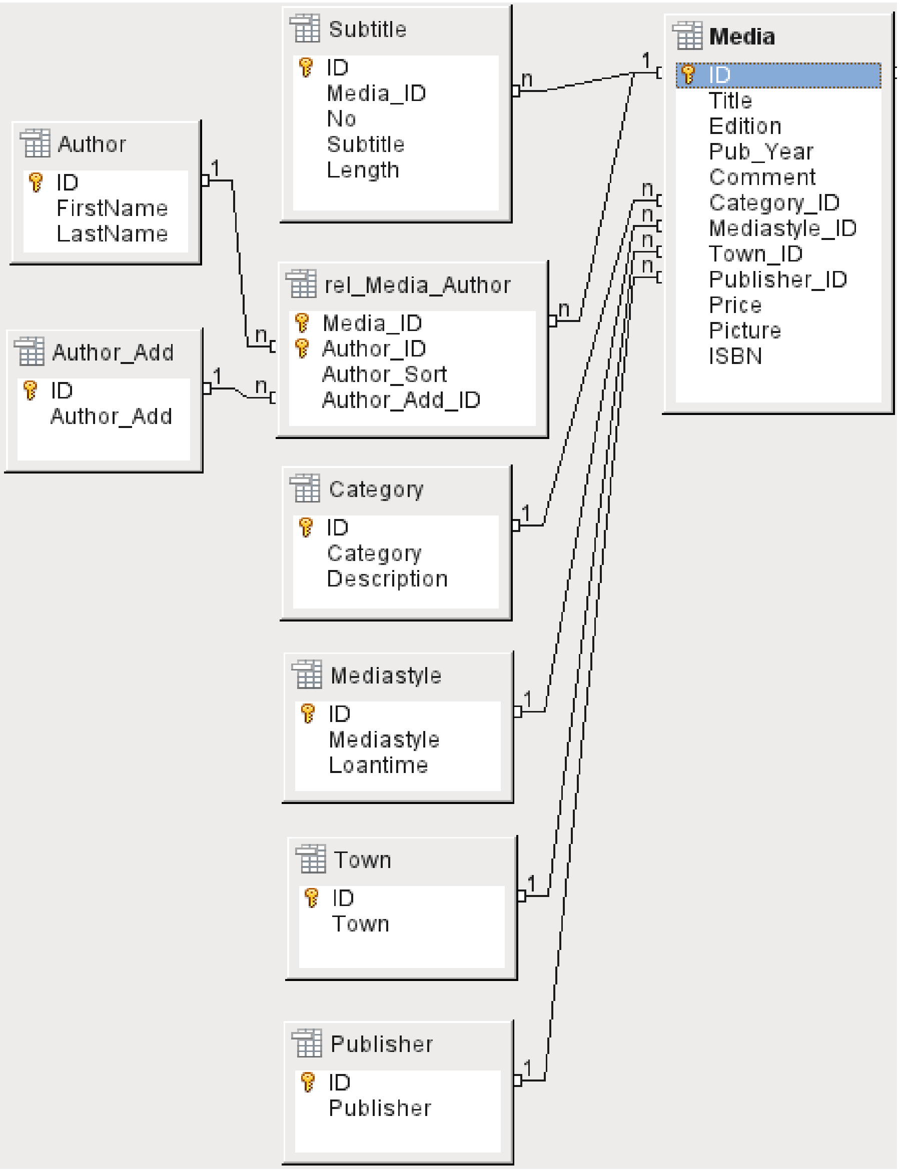


Figure 128: Media addition

Loan table

The central table is *Loan* (see Figure 129). It is the link between the Media and Reader tables.

When a medium is returned, much of its data can be deleted as it is no longer needed. But two of the fields should not be: ID, and Loan_Date. The former is the primary key. The latter is when the item was loaned. It serves two purposes. First it is useful to determine the most popular media. Second, if damage to an item is noticed while it is being taken out, this field will show who was the last person to borrow it. In addition, the Return_Date is recorded when the item is returned.

Similarly, Reminders are integrated into the loan procedure. Each reminder is separately entered into the *Recall* table so that the total number of reminders can be determined.

As well as an extension period in weeks, there is an extra field in the loan record that enables media to be loaned using a barcode scanner (Media_ID_BC). Barcodes contain, in addition to the individual Media_ID, a check digit which the scanner can use to determine if the value scanned in is correct. This barcode field is included here only for test purposes. It would be better if the primary key of the Media table could be directly entered in barcode form, or if a macro were used to remove the check digit from the entered barcode number before storage.

Finally we need to connect the Reader to the loan. In the Reader table, only the name, an optional lock, and a foreign key linking to the Gender table are included in the plan.

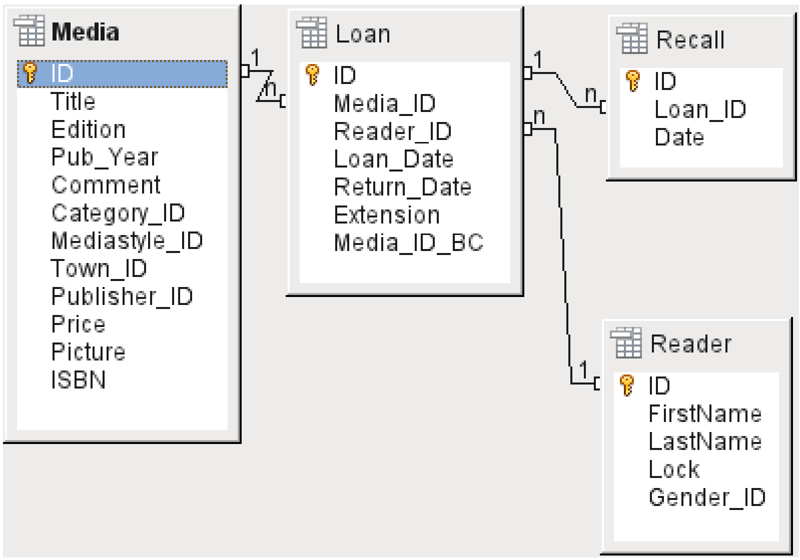


Figure 129: Loan table relationships

User administration table

For this table design, two scenarios are envisaged. The chain of tables shown in Figure 130 is designed for school libraries. Here there is no need for addresses, as the pupils can be contacted through the school. Reminders do not need to be sent out by post but can be distributed internally.

The Address chain is necessary in the case of public libraries. Here you need to enter data that will be needed for the creation of reminder letters. See Figure 130.

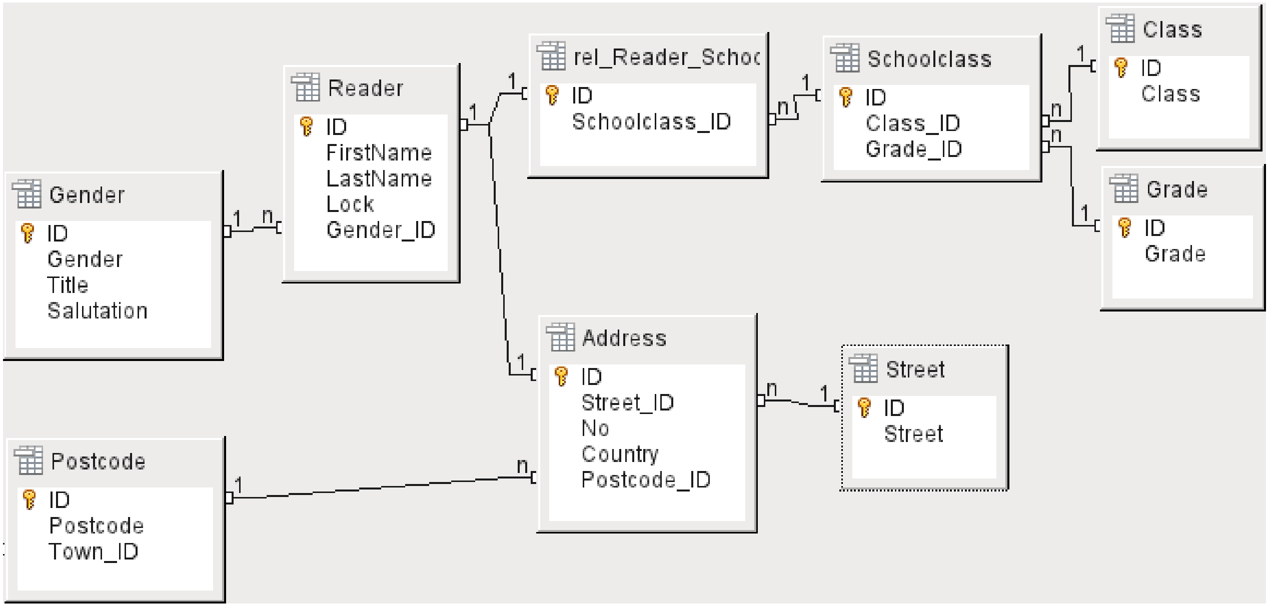


Figure 130: Readers - a School class chain and an Address chain

The *Gender* table ensures that the correct salutation is used in reminders. The writing of reminders can then be automated as far as possible. In addition, some given names can be equally masculine or feminine. Therefore the separate listing of gender is required even when reminders are written out by hand.

The *rel_Reader_Schoolclass* table, like the *Address* table, has a 1:1 relationship with the *Reader* table. This was chosen because either the school class or the address might be required. Otherwise the Schoolclass_ID could be put directly into the pupil table; the same would be true of the complete content of the address table in a public library system.

A *School class* usually consists of a year designation and a stream suffix. In a 4-stream school, this suffix might run from *a* to *d*. The suffix is entered in the *Class* table. The year is in a separate *Grade* table. That way, if readers move up a class at the end of each school year, you can simply change the year entry for everyone.

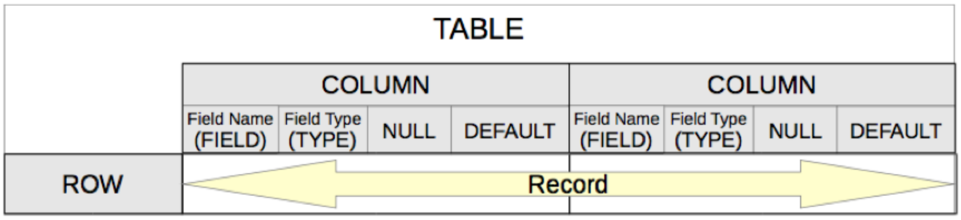
The *Address* is also divided. *Street* is stored separately because street names within an area are often repeated. Post code and town are separated because there are often several post codes for a single area and therefore more post codes than towns. So compared with the *Address* table, the Postcode table contains significantly fewer records and the *Town* table even fewer.

How this table structure is put to use is explained further in Chapter 4, Forms, in this book.

Creating tables

Most LibreOffice users will generally use the graphical user interface (GUI) exclusively to create tables. Direct entry of SQL commands becomes necessary when, for example, a field must subsequently be inserted at a particular position, or a standard value must be set after the table has been saved.

Table terminology: Figure 131 shows the standard division of tables into columns and rows.



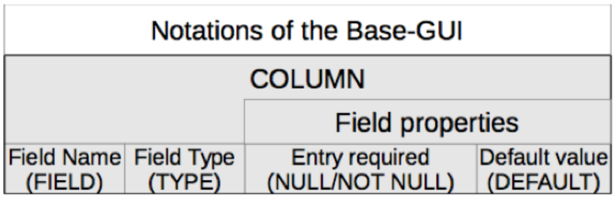
TABLE

	COLUMN				COLUMN			
	Field Name (FIELD)	Field Type (TYPE)	NULL	DEFAULT	Field Name (FIELD)	Field Type (TYPE)	NULL	DEFAULT
ROW	Record							

Figure 131: Standard table terminology

Each data record is stored in its own row of the table. Individual columns are largely defined by the field, the type, and the rules that determine if the field can be empty. According to the type, the size of the field in characters can also be determined. In addition, a default value can be specified to be used when nothing was entered into the field.

In the Base GUI, the terms for a column are described somewhat differently, as shown in Figure 132.



Notations of the Base-GUI

COLUMN			
		Field properties	
Field Name (FIELD)	Field Type (TYPE)	Entry required (NULL/NOT NULL)	Default value (DEFAULT)

Figure 132: Base table terminology

Field becomes Field Name, Type becomes Field Type. Field Name and Field Type are entered into the upper area of the Table Design window. You can set additional column properties under the Field properties on the bottom. These additional properties depends on the column type defined above and are also limited by the user interface. To overcome the interface limitations such as setting the default value of a date field to the actual date of entry, use an appropriate SQL command (see “Direct entry of SQL commands” on page 151).

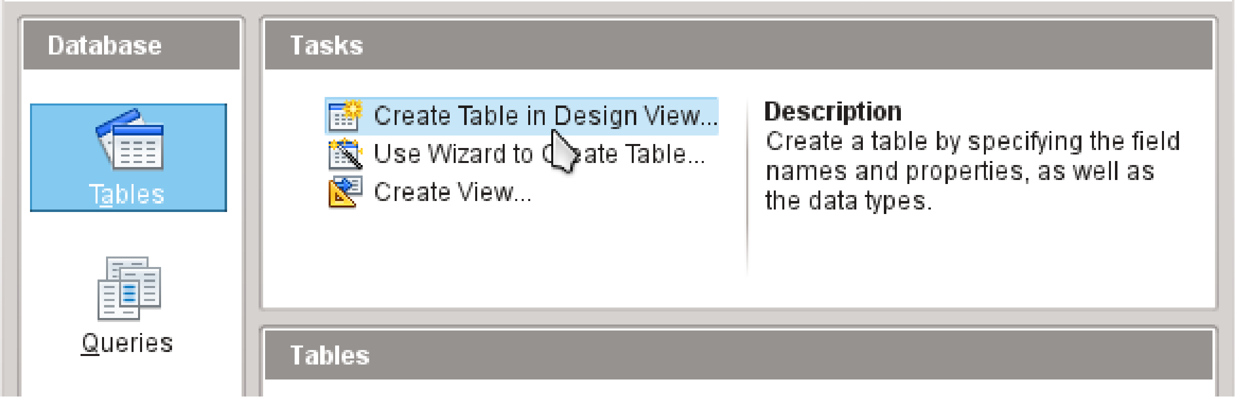
Note

Default values: The term “Default value” in the GUI does not mean what the database user generally understands as a default value. The GUI displays a certain value visibly, which is saved with the data.

The default value in a database is stored in the table definition. It is then written into the field whenever the field is empty in a new data record. SQL default values do not appear when editing table properties.

Creation using the graphical user interface

Database creation using the graphical user interface (GUI) is explained step by step, using the Media table as an example.



Description
Create a table by specifying the field names and properties, as well as the data types.

Figure 133: Click Create Table in Design View

Launch the table editor by clicking on **Create Table in Design View**.

4) ID field:

   e) In the first column, enter the Field Name *ID*. Then type the Tab key to move to the Field Type column. Alternatively, click with the mouse on the next column to select it, or press the *Enter* key.

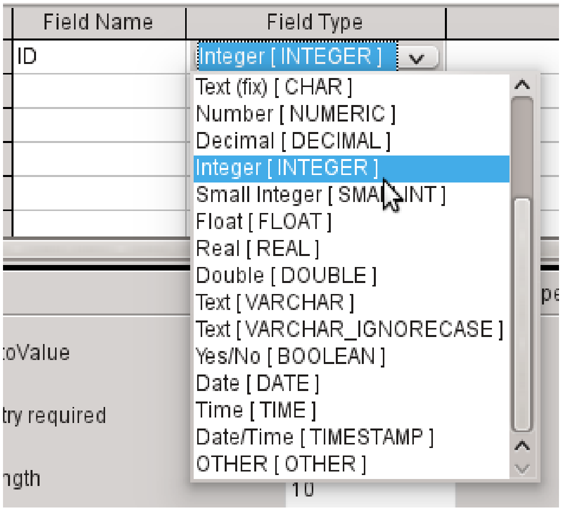


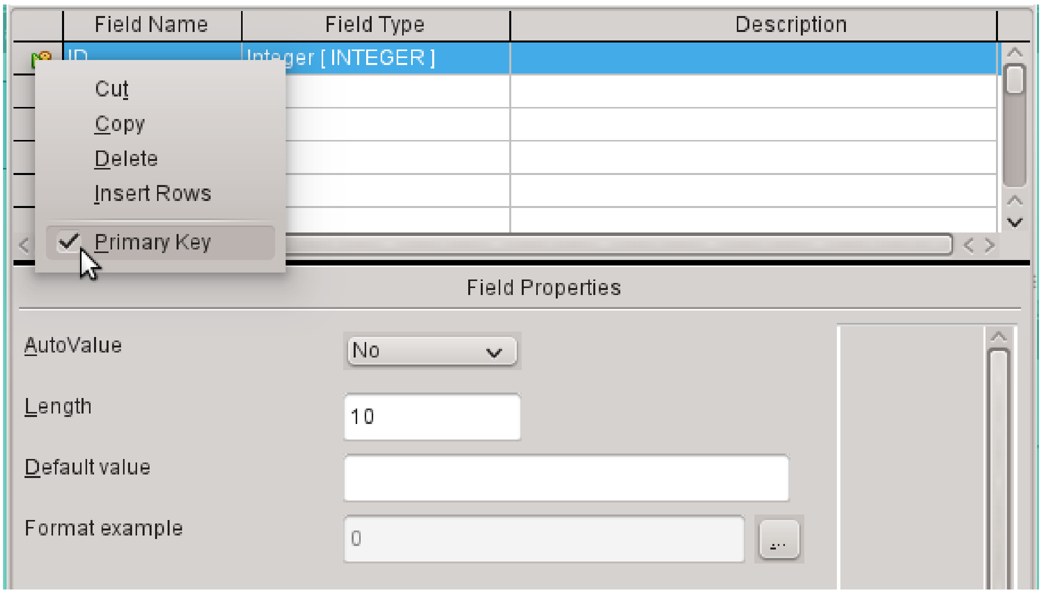
Figure 134: Select Integer type for ID field

f) Select `Integer [INTEGER]` from the list as the field type. The default is `Text [VARCHAR]`. Integers can store a number with up to 10 digits. In addition, Integer is the only type available in the GUI that can be given an automatically incrementing value.

Tip

To rapidly make a selection from the Field Type list using the keyboard, type the key corresponding to the first letter of your choice. Repeated typing of this key can be used to change the selection. For example, typing D can change your selection from Date to Date/Time or to Decimal.

g) Set *ID* as the primary key by clicking the rectangle in front of its field name with the right mouse button and selecting Primary Key from the context menu. A key symbol appears before ID.



Field Name	Field Type	Description
ID	Integer [INTEGER]	

Field Properties

AutoValue: No

Length: 10

Default value:

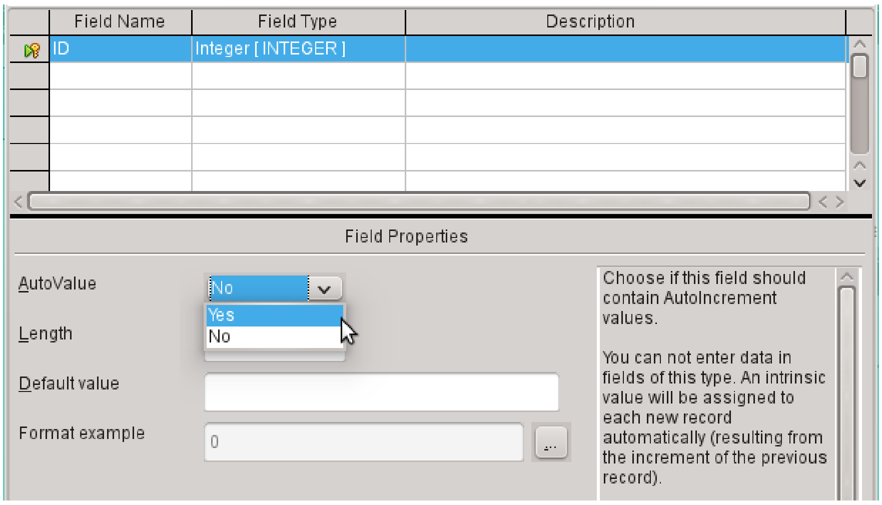
Format example: 0

Figure 135: Set primary key using the context menu

Note

The primary key serves one purpose only – to uniquely identify the record. Therefore you can use an arbitrary name for this field. In the example, we have used the commonly used name ID (identification).

h) Under Field properties for the ID field, change *AutoValue* from *No* to *Yes*. This causes the primary key to be automatically incremented. In the internal database, the count begins at 0.



Field Name	Field Type	Description
ID	Integer [INTEGER]	

Field Properties

AutoValue: No (Yes / No)

Length

Default value

Format example: 0

Choose if this field should contain AutoIncrement values.

You can not enter data in fields of this type. An intrinsic value will be assigned to each new record automatically (resulting from the increment of the previous record).

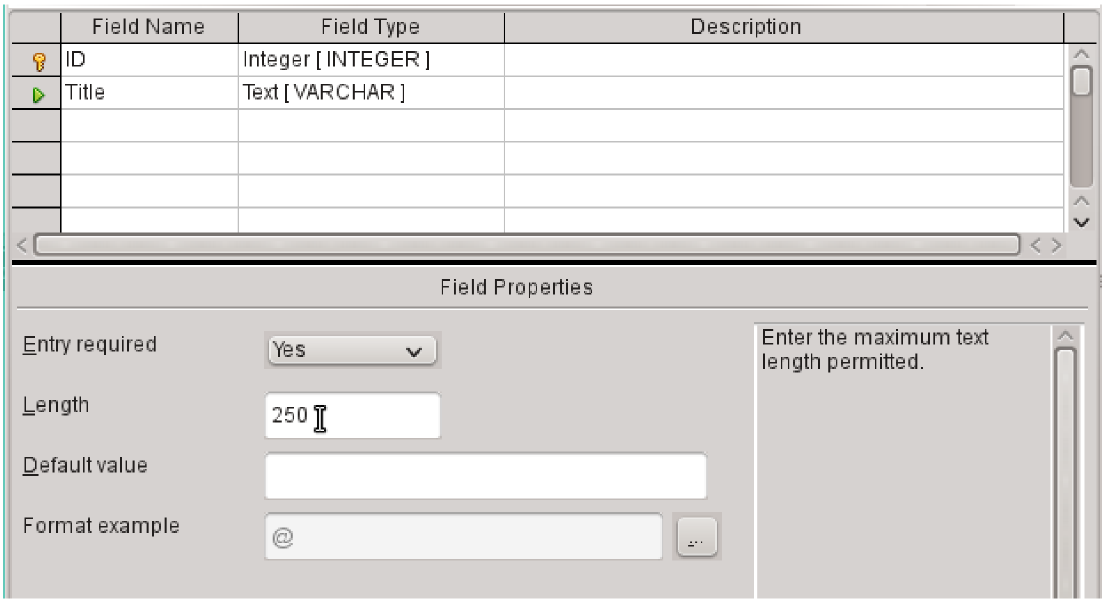
Figure 136: Set AutoValue to Yes for ID field

AutoValue can be set for only one field in a table. Choosing **AutoValue > Yes** automatically sets this field to be the primary key if a primary key has not been set already.

5) The next field is *Title*.

   a) The field name *Title* is entered in the Field Name column.

   b) The field type need not be altered here as it is already set to `Text [VARCHAR]`.



Field Name	Field Type	Description
ID	Integer [INTEGER]	
Title	Text [VARCHAR]	

Field Properties

Entry required: Yes

Length: 250

Default value

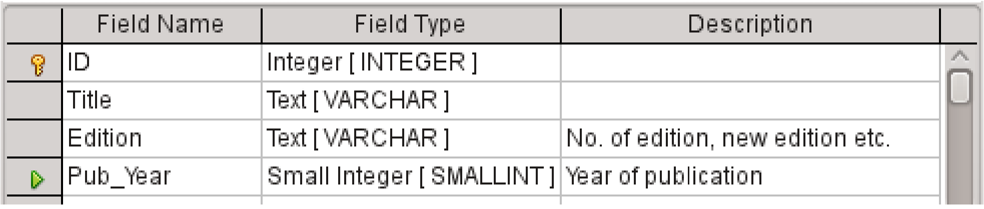
Format example: @

Enter the maximum text length permitted.

Figure 137: Add Title field

   c) In the Field properties, the field length should be increased to 250 for media titles.

   d) In Field properties, change *Entry required* from *No* to *Yes*. A medium without a title would make no sense.

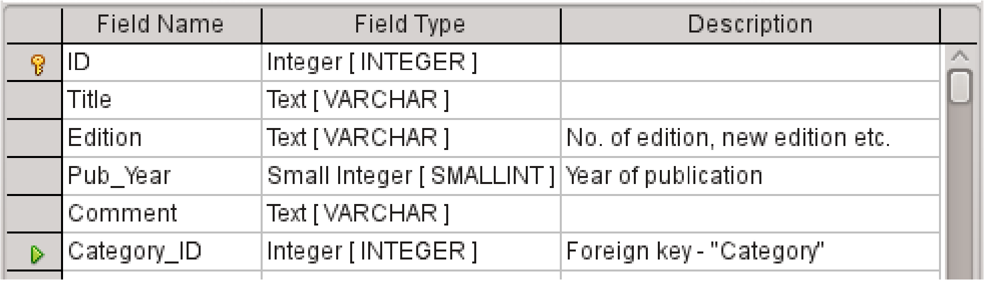


Field Name	Field Type	Description
ID	Integer [INTEGER]	
Title	Text [VARCHAR]	
Edition	Text [VARCHAR]	No. of edition, new edition etc.
Pub_Year	Small Integer [SMALLINT]	Year of publication

Figure 138: Add Edition and Pub_Year fields

6) *Description* could be anything. This column can also be left empty. The description serves only to explain the field content for people who want to view the table definition later.

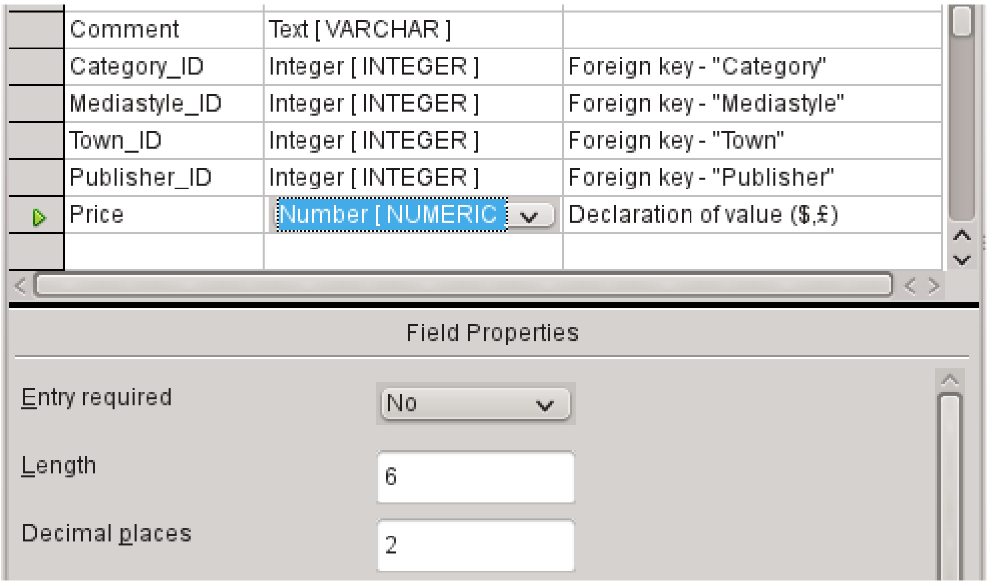
7) For the field Pub_Year, the `Small Integer [SMALLINT]` type has been chosen. This can contain an integer with a maximum size of 5 digits. Making the publication date an integer ensures that it will not contain any alphabetic characters.



Field Name	Field Type	Description
ID	Integer [INTEGER]	
Title	Text [VARCHAR]	
Edition	Text [VARCHAR]	No. of edition, new edition etc.
Pub_Year	Small Integer [SMALLINT]	Year of publication
Comment	Text [VARCHAR]	
Category_ID	Integer [INTEGER]	Foreign key - "Category"

Figure 139: Add Comment and Category_ID fields

8) For the Category_ID field, we have chosen the `Integer` type. In the Category table, the primary key should have this field type, so what is entered here as a foreign key must have the same type. This also applies to the following foreign keys Mediastyle_ID, Town_ID, and Publisher_ID.



Comment	Text [VARCHAR]	
Category_ID	Integer [INTEGER]	Foreign key - "Category"
Mediastyle_ID	Integer [INTEGER]	Foreign key - "Mediastyle"
Town_ID	Integer [INTEGER]	Foreign key - "Town"
Publisher_ID	Integer [INTEGER]	Foreign key - "Publisher"
Price	Number [NUMERIC	Declaration of value ($,£)

Field Properties

Entry required: No

Length: 6

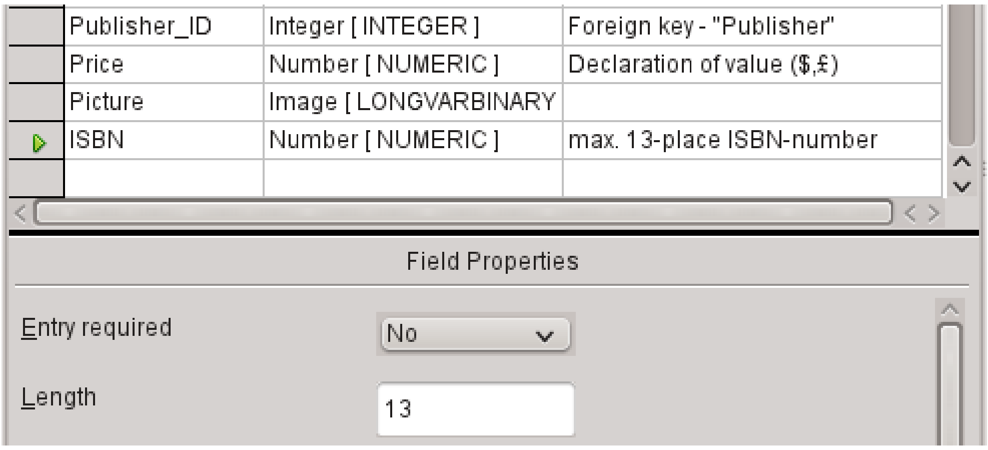
Decimal places: 2

Figure 140: Add Mediastyle_ID, Town_ID, Publisher_ID, and Price fields

9) For the Price field, use the `[NUMERIC]` or `[DECIMAL]` type. Both these field types can contain values with a decimal point. Under Field Properties, set a length of 6 characters. This should be sufficient for the prices of our media.

   a) The number of decimal places is set to 2. This provides a maximum price of 9999.99 since the decimal point itself is not included in the count.

   b) It is not necessary to specify the $ character in the format, as a formula will handle this.



	Publisher_ID	Integer [INTEGER]	Foreign key - "Publisher"
	Price	Number [NUMERIC]	Declaration of value ($,£)
	Picture	Image [LONGVARBINARY	
▷	ISBN	Number [NUMERIC]	max. 13-place ISBN-number

Field Properties

Entry required: No

Length: 13

Figure 141: Add Picture and ISBN fields

10) For the ISBN field, use the [NUMERIC] type. This can be set exactly to the correct field length for an ISBN. ISBNs are 10 or 13 characters long. They will be stored as numbers without a separator. The length is set to a maximum of 13 characters. The number of decimal places is set to zero.

11) Save the table with the name Media.

We have now created the main table for the example database. All the other tables can be created in a similar way. Be careful that the field types and field properties match what is going to be stored in those fields. This is different from a spreadsheet, in which a column can contain a mixture of properties.

Note

The order of fields in the table can be changed only until the table is first saved in the GUI. When data is subsequently entered directly into the table, the field order is fixed. However, the order can still be freely changed in queries, forms, and reports.

Primary keys

If no primary key is set when the table is designed, you will be asked when saving the table whether a primary key should be created. This indicates that a significant field is lacking in the table. Without a primary key, the HSQLDB database cannot access the table in the GUI. This field is usually named ID and given the type INTEGER with *AutoValue* > *Yes* to automatically increment the value. Clicking **Yes** in the primary key dialog automatically creates a primary key field. Clicking No or Cancel in the primary key dialog allows you to designate an existing field as the primary key, by right-clicking on the green arrow at the left of the corresponding field.

You can also use a combination of fields as your primary key. The fields must be declared as primary key together (hold the Control or Shift key down). Then a right-click makes the combination of all highlighted fields the primary key.

If information is imported into this table from others (for example an address database with externally stored postcodes and towns), a field with the same data type as the primary key of the other table must be included. Suppose the Postcode table has as its primary key a field called ID with the type Tiny Integer. The Address table must then have a field Postcode_ID with the Tiny Integer type. What is entered into the Address table is always the number that serves as the primary key for the given location in the Postcode table. This means that the Address table now has a foreign key in addition to its own primary key.

A rule for naming fields in a table is that no two fields can have the same name. Therefore you cannot have a second field called ID occurring as a foreign key in the Address table.

The field type can be altered only to a limited extent. Increasing a property (length of a text field, greater size in number) is always allowed, as all values already entered will match the new conditions. Decreasing a property is more likely to cause problems and can lead to a loss of data.

Time fields in tables cannot be created to contain fractions of a second. For that, you need a Timestamp field. However the GUI only allows you to create a Timestamp field with date, hour, minute and second. You will need to modify this field afterward using **Tools > SQL.**

```
ALTER TABLE "Table_name" ALTER COLUMN "Field_name" TIMESTAMP(6)
```

The parameter “6” makes the Timestamp field capable of storing fractions of a second.

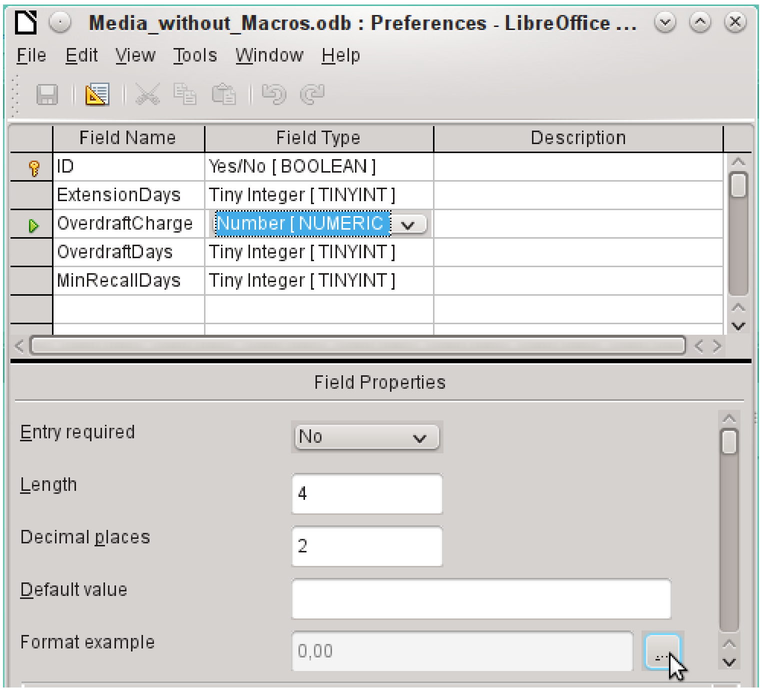
Formatting fields

Formatting presents the values in the database to the user and allows the input of values depending on the input conventions normal in that country. Without formatting, decimal places are marked off with a dot where most European countries use a comma (*4.21* instead of *4,21*). Date values are presented in the form *2014-12-22*. In setting up formatting, you must have regard for local standards.

Formatting only provides a representation of the contents. A date represented by a two-character year number is still stored as a four-character year. If a field is created for a number with two decimal places, like the overdue charge (called overdraft) in the following example, the number is stored with two decimal places, even if the formatting has mistakenly been set not to show them. A number with two decimal places can be entered into a field formatted without decimal places. The decimal part seems to disappear at input but becomes visible if the formatting is bypassed.

To display just a time, not a date, forms can be formatted to show only the necessary information, hiding the rest of the Timestamp field. In the case of storing time from a stopwatch for example, the minutes, seconds, and fractions of a second in a Timestamp can be displayed by using MM:SS.00 in the display format. A format without the date can be set later in Forms using the formatted field, but not directly into the Timestamp field.

The formatting of fields when the table is created or subsequently, via the field properties, uses a separate dialog:



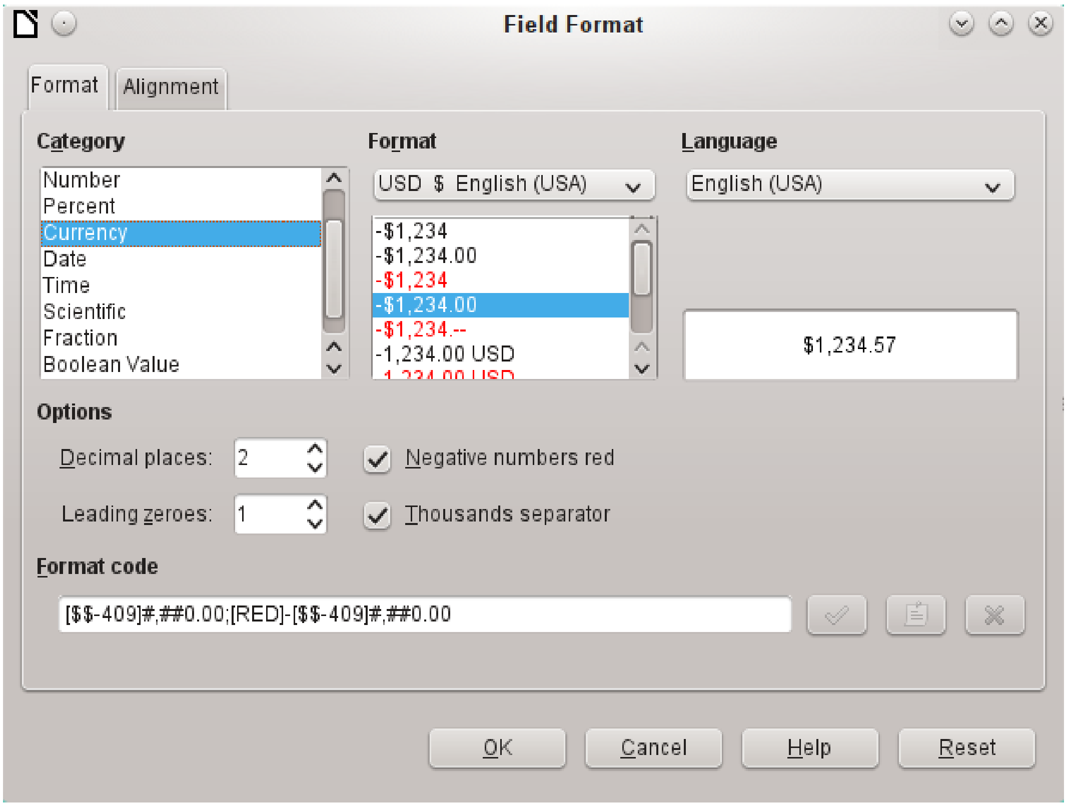
Field Name	Field Type	Description
ID	Yes/No [BOOLEAN]	
ExtensionDays	Tiny Integer [TINYINT]	
OverdraftCharge	Number [NUMERIC]	
OverdraftDays	Tiny Integer [TINYINT]	
MinRecallDays	Tiny Integer [TINYINT]	

Field Properties

Entry required	No
Length	4
Decimal places	2
Default value	
Format example	0,00

Figure 142: Click button adjacent to Format example to access Field Format dialog

The button next to **Field Properties > Format example** opens the dialog for changing the format.



Category: Currency

Format: USD $ English (USA); -$1,234.00

Language: English (USA)

Decimal places: 2

Leading zeroes: 1

☑ Negative numbers red

☑ Thousands separator

Format code: [$$-409]#,##0.00;[RED]-[$$-409]#,##0.00

Figure 143: Field Format dialog

When creating currency fields, take care that the numeric field has two decimal places set. Formatting can be carried out when creating the table in the GUI to use the appropriate currency during input. This only affects input into the table and into queries that use the input value without recalculation. In forms, the currency designation must be separately formatted.

Note

Base saves the formatting of tables when the fields are created or during data entry if the column formats are modified by right-clicking on the column headings. Column widths on the input screen are also saved when modified during data entry.

In queries, forms, or reports, the display formatting can be modified as needed.

In the case of fields that are to contain a percentage, take note that 1% must be stored as 0.01. Writing percentages thus requires at least two decimal places. If fractional percentages such as 3.45 need to be stored, the stored numeric value requires four decimal places.

Creating an index

Sometimes it is useful to index other fields or a combination of other fields in addition to the primary key. An index speeds up searching and can also be used to prevent duplicate entries.

Each index has a defined sort order. If a table is displayed without sorting, the sort order will be according to the content of the fields specified in the index.

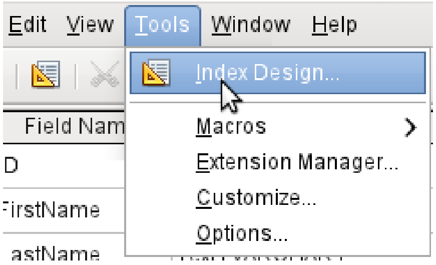
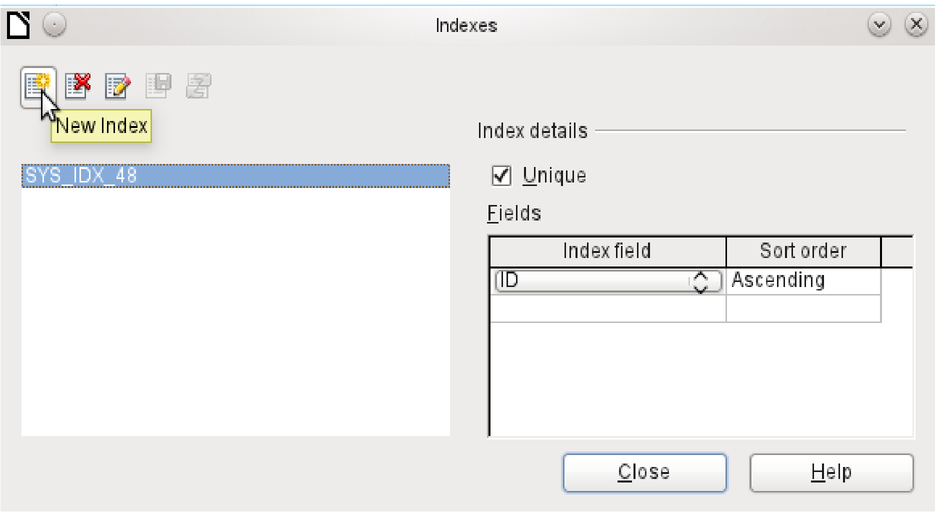


Figure 144: Select Tools > Index Design

Open the table for editing by right-clicking and using the context menu. Then you can access index creation with **Tools > Index Design**.



SYS_IDX_48

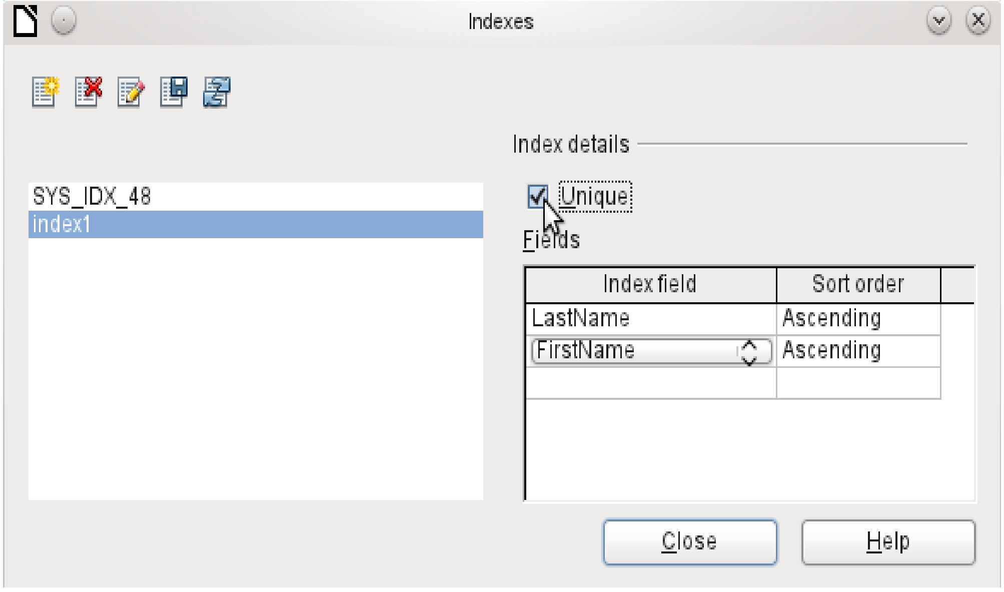
☑ Unique

Index field	Sort order
ID	Ascending

Figure 145: Create a new Index

On the Indexes dialog (Figure 145), click **New Index** to create an index in addition to the primary key.

The new index is automatically given the name index1. The **Index field** specifies which field or fields are to be used for this index. At the same time you can choose the Sort order.



SYS_IDX_48
index1

☑ Unique

Index field	Sort order
LastName	Ascending
FirstName	Ascending

Figure 146: The index is defined as Unique

In principle, an index can also be created from table fields that do not contain unique values. However in Figure 146, the Index detail **Unique** has been checked, so that the LastName field together with the FirstName field can only have entries that do not already occur in that combination. So, for example, Robert Miller and Robert Maier are possible, and likewise Robert Miller and Eva Miller.

If an index is created for one field only, the uniqueness applies to that field. Such an index is usually the primary key. In this field each value may occur only once. Additionally, in the case of primary keys, the field cannot be NULL under any circumstances.

An exceptional circumstance for a unique index is when there is no entry into a field (the field is NULL). Since NULL can have any arbitrary value, an index using two fields is always allowed to have the same entry repeatedly in one of the fields as long as there is no entry in the other.

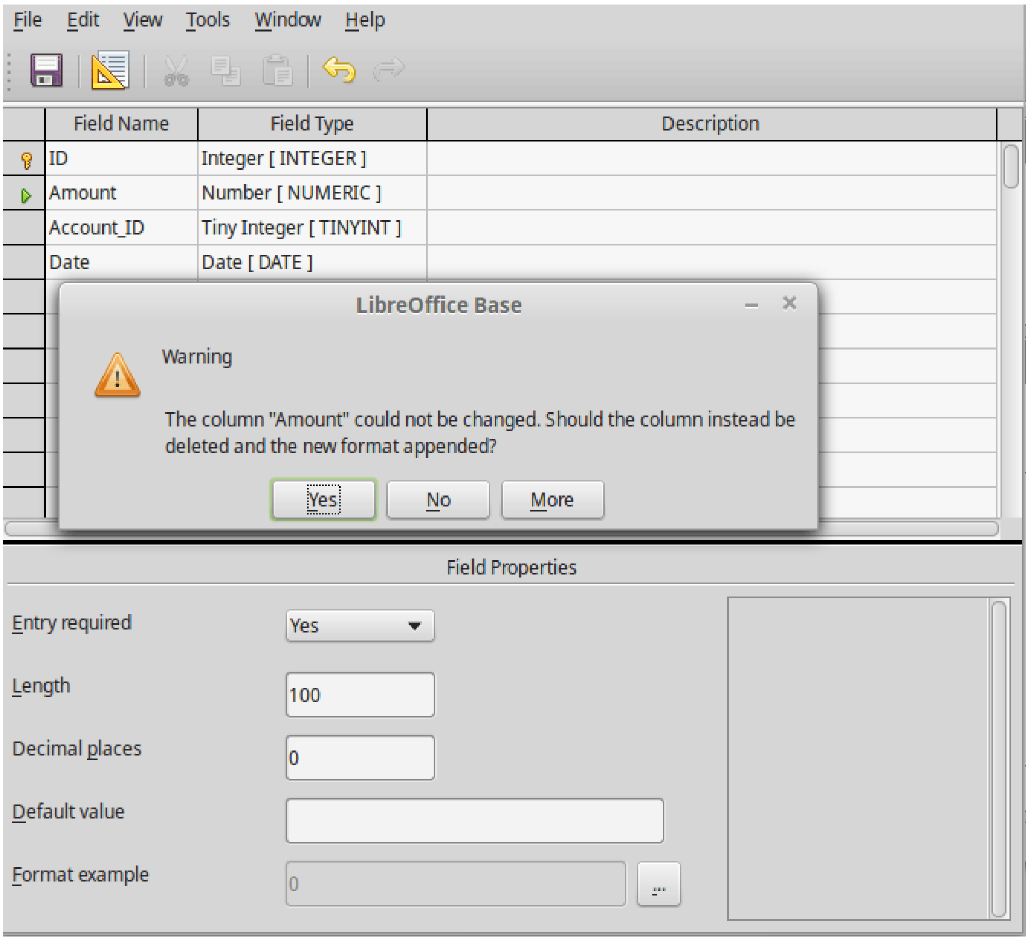
✓ **Note**

> **NULL** is used in databases to designate an empty cell, one that contains nothing. No calculation is possible using a NULL field. This contrasts with spreadsheets, in which empty fields automatically contain the value 0 (zero).

Example: In a media database, the media number and the loan date are entered when the item is loaned out. When the item is returned, a return date is entered. In theory, an index using the fields Media_ID and ReturnDate could easily prevent the same item from being loaned out repeatedly without the return date being noted. However, this will not work because the return date initially has no value. The index will prevent an item from being marked as returned twice with the same date but it will do nothing else.

Problems when modifying tables

It is best to create tables complete with all their required settings, so that changes in table configuration are not needed at a later time. When properties of fields (field name, mandatory entry, and so on) are changed later, this can lead to error messages which are not due to the GUI but to an attempt to modify the underlying database in an undesirable way.



	Field Name	Field Type	Description
	ID	Integer [INTEGER]	
	Amount	Number [NUMERIC]	
	Account_ID	Tiny Integer [TINYINT]	
	Date	Date [DATE]	

LibreOffice Base

Warning

The column "Amount" could not be changed. Should the column instead be deleted and the new format appended?

Entry required: Yes

Length: 100

Decimal places: 0

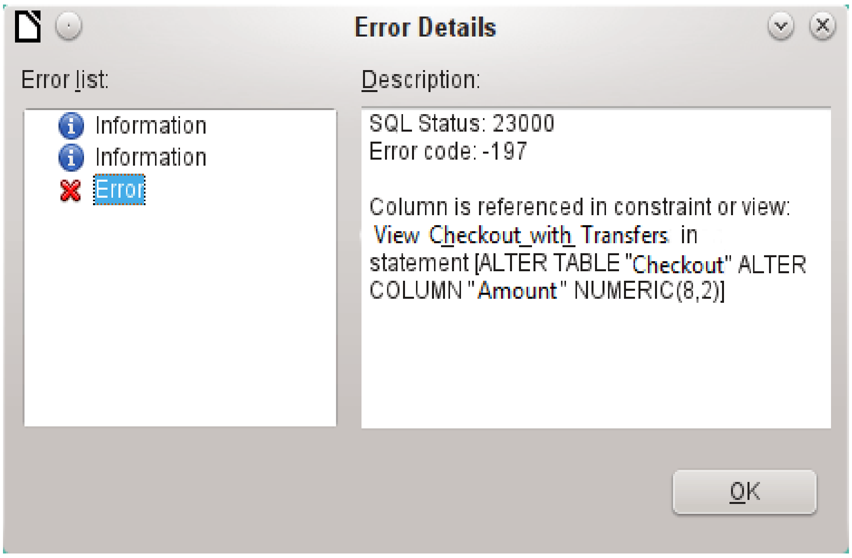
Default value:

Format example: 0

Figure 147: Error when attempting to change Entry required to Yes

In this case the Amount field is to be reset to Entry required=yes. The warning symbol notifies us that this change can lead to loss of data. A simple change is not possible because there may already be records that have no entry in this field.

Clicking **Yes** leads to a further error notice, as the structure of the database does not allow this field to be deleted. Clicking on No cancels the entire operation. The More option is provided whenever possible in order to give you additional information on solving the problem.



Error Details

Error list:

- Information
- Information
- Error

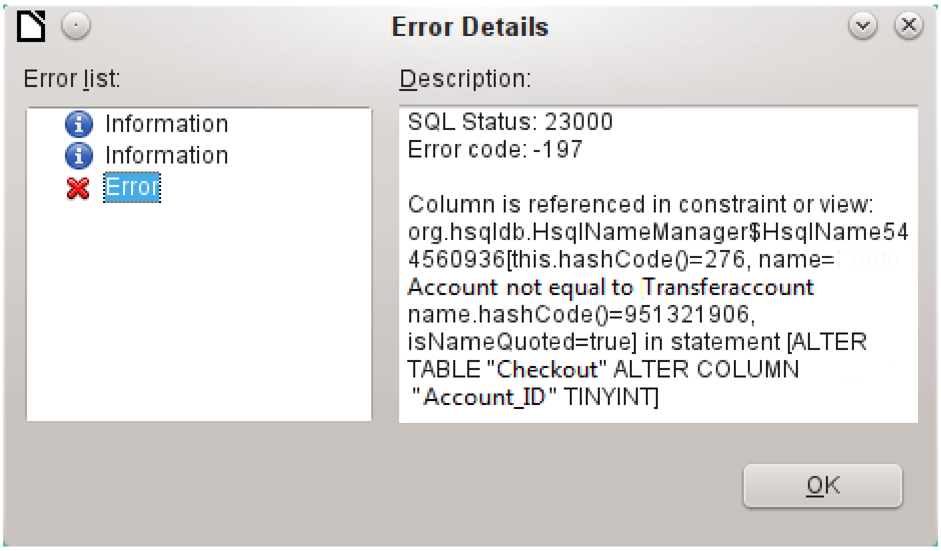
Description:

SQL Status: 23000
Error code: -197

Column is referenced in constraint or view: View Checkout_with_Transfers in statement [ALTER TABLE "Checkout" ALTER COLUMN "Amount" NUMERIC(8,2)]

Figure 148: Error Details dialog when unable to alter the Amount field

The error notice `Column is referenced in constraint or view` means:
The column with the field name “Amount” is referred to in another part of the database. This could be a constraining definition or a table view which was created by some user after the table itself was created. The above illustration shows that the name of the constraint or view is View_Checkout_with_Transfers. This makes it clear to the user where in the database changes need to be made. For example the SQL code for the view could first be saved as a query, and then the view could be destroyed and a new attempt made to recreate the field.



```
SQL Status: 23000
Error code: -197

Column is referenced in constraint or view:
org.hsqldb.HsqlNameManager$HsqlName54
4560936[this.hashCode()=276, name=
Account not equal to Transferaccount
name.hashCode()=951321906,
isNameQuoted=true] in statement [ALTER
TABLE "Checkout" ALTER COLUMN
 "Account_ID" TINYINT]
```

Figure 149: Error Details dialog when unable to alter the Account_ID field

In this case, the name of the constraint *Account not equal to Transferaccount* leads us to the definition for that constraint. The condition is that the value in the field named Account_ID is not allowed to be the same as the value in the field TransferAccount_ID. The column can only be altered if this condition is removed.

Now if a further error occurs, this is most likely to be caused by the corresponding field being linked to a field in another table by a defined relationship. In this case, the link must be broken by using **Tools > Relationships** before the change can be carried out.

Limitations of graphical table design

The sequence of fields in a table cannot be changed after the database has been saved. To display a different sequence requires a query.

Only the entry of direct SQL commands can insert a field into a specific position in the table. However, fields already created cannot be moved by this method.

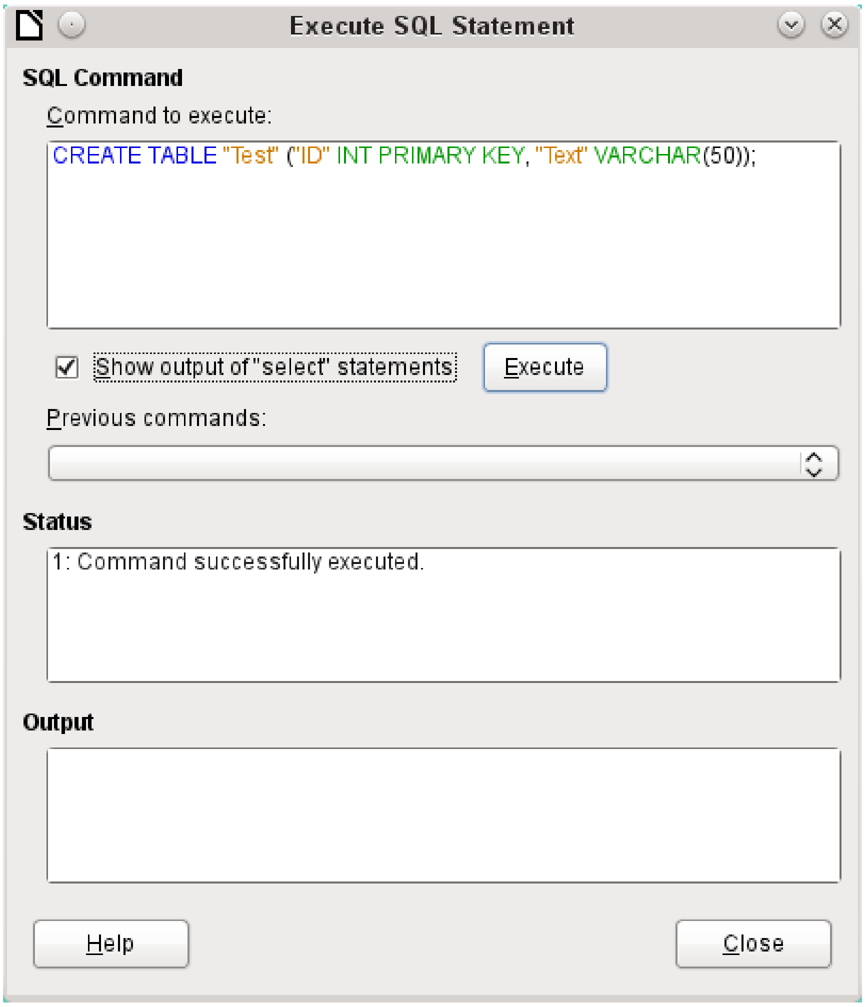
The properties of the tables must be set at the beginning: for example which fields must not be NULL and which must contain a standard value (Default). These properties cannot subsequently be changed using the GUI.

The default values you are able to set in the GUI are not as powerful as the possible default values within the database itself. For example, you cannot define the default for a date field as being the date of entry. That is only possible with directly entered SQL commands.

Direct entry of SQL commands

To enter SQL commands directly, go to **Tools > SQL**.

Commands are entered in the upper area of the window (Figure 150). In the lower (Status) area, the success or the reason for failure is shown. The results from queries can be displayed in the Output box if the checkbox is selected.



```
CREATE TABLE "Test" ("ID" INT PRIMARY KEY, "Text" VARCHAR(50));
```

☑ Show output of "select" statements

1: Command successfully executed.

Figure 150: Dialog for direct entry of SQL commands

A summary of the possible commands for the built-in HSQLDB engine can be found at http://www.hsqldb.org/doc/1.8/guide/ch09.html. The contents are described in the following sections. Some commands only make sense when dealing with an external HSQLDB database (Specify User, etc.). Where necessary, these are dealt with in the section “Working with external HSQLDB” in the Appendix to this handbook.

Note

LibreOffice is based on Version 1.8.0 of HSQLDB. The currently available server version is 2.5. The functions of the new version are more extensive. They can be reached at http://hsqldb.org/web/hsqlDocsFrame.html. The description of Version 1.8 is now at http://www.hsqldb.org/doc/1.8/guide/. A further description is given in the installation packages for HSQLDB, which can be downloaded from http://sourceforge.net/projects/hsqldb/files/hsqldb/.

Table creation

A simple command to create a usable table is:

```
CREATE TABLE "Test" ("ID" INT PRIMARY KEY, "Text" VARCHAR(50));
```

Breakdown of this command:

`CREATE TABLE "Test"`: Creates a table with the name "Test".

`( )`: the specified field names, field types and options are inserted between parenthesis.

`"ID" INT PRIMARY KEY,`: Field name ID with the numeric type integer as the primary key;

`"Text" VARCHAR(50)`: field name Text with the text type variable text length and the text size limited to 50 characters.

Parameters for the CREATE command:

```
CREATE [MEMORY | CACHED | [GLOBAL] TEMPORARY | TEMP | TEXT] TABLE
"Table name" ( <Field definition> [, ...] [,
<Constraint Definition>...] ) [ON COMMIT {DELETE | PRESERVE} ROWS];
```

`[MEMORY | CACHED | [GLOBAL] TEMPORARY | TEMP | TEXT]:`

Specifies the location of the newly created table. The default setting is `MEMORY`: HSQLDB creates all tables in core memory. This setting also applies to the tables that are written into the embedded database by LibreOffice Base. Another possibility would be to write the tables to the hard drive and use memory only to buffer access to the hard drive (`CACHED`).

Note

```
CREATE TEXT TABLE "Text" ("ID" INT PRIMARY KEY, "Text"
VARCHAR(50));
```

Creates a text table in HSQLDB. Now it must be linked to an external text file (for example a *.csv file): `SET TABLE "Text" SOURCE "Text.csv";`

Naturally the *Text.csv* file must have corresponding fields in the correct order. When creating the link, various additional options can be selected. For details see http://www.hsqldb.org/doc/1.8/guide/guide.html#set_table_source-section

Text tables are not write-protected against other programs. It can therefore happen that another program or user alters the table just as Base is accessing it. Text tables are used mainly for data exchange between different programs.

Tables in `TEXT` format (such as CSV) are not writable in internal databases that are set up purely in `MEMORY`, while Base cannot access `TEMPORARY` or `TEMP` tables. The SQL commands are carried out in this case but the tables are not displayed (and therefore cannot be deleted) using the GUI, and data entered via SQL is likewise not visible to the query module of the GUI, unless the automatic deletion of the contents after the final commit is prevented (with `ON COMMIT PRESERVE ROWS`). Any request in this case shows a table without any contents.

Tables built directly with SQL are not immediately displayed. You must either use **View > Refresh Tables** or simply close the database and then reopen it.

`<Field definition>:`

```
"Field name" Data type [(Number of characters[,Decimal places])]
[{DEFAULT "Default value" | GENERATED BY DEFAULT AS IDENTITY (START
WITH <n>[, INCREMENT BY <m>])}] | [[NOT] NULL] [IDENTITY] [PRIMARY
KEY]
```

Allows default values to be included in the field definition.

For text fields, you can enter text in single quotes or `NULL`. The only SQL function allowed is `CURRENT_USER`. This only makes sense if HSQLDB is being used as an external Server database with several users.

For date and time fields, a date, a time, or a combination of the two can be entered in single quotes or else NULL. You must ensure that the date follows the American conventions (yyyy-mm-dd), that time has the format hh:mm:ss, and that a combined date/time value has the format yyyy-mm-dd hh:mm:ss.

Allowed SQL functions:
for the current date – *CURRENT_DATE, TODAY, CURDATE()*
for the current time – *CURRENT_TIME, NOW, CURTIME()*
for the current data time stamp – *CURRENT_TIMESTAMP, NOW*.

For boolean Fields (yes/no) the expressions *FALSE, TRUE, NULL* can be entered. These must be entered without single quotes.

For numeric fields, any valid number in the range, or NULL is possible. Here too, if you enter `NULL`, do not use quotes. When entering decimals, make sure that the decimal point is a dot (period) and not a comma. (Some English speaking people use a comma as the decimal point.)

For binary fields (images, etc.) any valid hexadecimal string in single quotes or `NULL` is possible. A hexadecimal example string is: 0004ff, which represents 3 bytes: first 0, then 4 and finally 255 (0xff). As binary fields in practice need only be entered for images, you need to know the binary code of the image that is to serve as a default.

Note

Hexadecimal system: Numbers are based on 16. A mixed system consisting of the numbers 0 to 9 and the letters a to f provides 16 possible digits for each column. With two columns, you can have 16*16=256 possible values. This corresponds to 1 Byte (28).

NOT NULL: The field value cannot be `NULL`. This condition can only be given in the field definition.

Example:

```
CREATE TABLE "Test" ("ID" INT GENERATED BY DEFAULT AS IDENTITY (START WITH 10),
"Name" VARCHAR(50) NOT NULL, "Date" DATE DEFAULT TODAY);
```

A table called Test is created. The key field ID is defined as AutoValue, with values starting at 10. The input field Name is a text field with a maximum size of 50 characters. It must not be empty. Finally we have the date field Date which by default stores the current date, if no other date is entered. This default value only becomes effective when a new record is created. Deleting a date in an existing record leaves the field empty.

```
 <Constraint definition>:
    [CONSTRAINT "Name"]
UNIQUE ( "Field_name 1" [,"Field_name 2"…] ) |
PRIMARY KEY ( "Field_name 1" [,"Field_name 2"…] ) |
FOREIGN KEY ( "Field_name 1" [,"Field_name 2"…] )
REFERENCES "other_table_name" ( "Field_name_1" [,"Field_name 2"…])
[ON {DELETE | UPDATE}
{CASCADE | SET DEFAULT | SET NULL}] |
CHECK(<Search_condition>)
```

Constraints define conditions that must be fulfilled when data is entered. Constraints can be given a name.

`UNIQUE ("Field_name")`: the field value must be unique within that field
PRIMARY KEY ("Field_name"): the field value must be unique and cannot be *NULL* (primary key)
`FOREIGN KEY ("Field_name") REFERENCES <"other_table_name"> ("Field_name")`: The specified fields of this table are linked to the fields of another table. The

field value must be tested for referential integrity as foreign keys; that is, there must be a corresponding primary key in the other table, if a value is entered here.
`[ON {DELETE | UPDATE} {CASCADE | SET DEFAULT | SET NULL}]`: In the case of a foreign key, this specifies what is to happen if, for example, the foreign record is deleted. It makes no sense, in a loan table for a library, to have a user number for which the user no longer exists. The corresponding record must be modified so that the relationship between the tables remains valid. Usually the record is simply deleted. This happens if you select `ON DELETE CASCADE.`
`CHECK(<Search_condition>)`: Formulated as a `WHERE` condition, but only for the current record.

```
CREATE TABLE "Time_measurement" ("ID" INT PRIMARY KEY, "Start_time" TIME,
"End_time" TIME, CHECK ("Start_time" <= "End_time"));
```

The CHECK condition excludes the input of an end time value earlier than the start time. An attempt to do this produces an error message similar to:

```
Check constraint violation SYS_CT_357 table: Time_measurement …
```

The search constraint is assigned a name that is not very informative. Instead, the name could be defined in the table definition:

```
CREATE TABLE "Time_measurement" ("ID" INT PRIMARY KEY, "Start_time" TIME,
"End_time" TIME, CONSTRAINT "Start_time<=End_time" CHECK ("Start_time" <=
"End_time"));
```

This gives a somewhat clearer error message in that the name of the constraint involved then appears.

Constraints must be honored when establishing relationships between tables or the indexing for particular fields. The constraints are established using the «CHECK» condition, in the GUI using Tools > **Relationships**, and also in **indexes created in Table design under Tools > Index design.**

`[ON COMMIT {DELETE | PRESERVE} ROWS]:`

> The content of tables of the type `TEMPORARY` or `TEMP` are erased by default when you have finished working with a particular record (`ON COMMIT DELETE ROWS`). This allows you to create temporary records, which contain information for other actions to be carried out at the same time.

If you want a table of this type to contain data available for a whole session (from opening a database to closing it), choose `ON COMMIT PRESERVE ROWS`.

Table modification

Sometimes you might wish to insert an additional field into a particular position in the table. Suppose you have a table called *Addresses* with fields ID, Name, Street, and so on. You realize that perhaps it would be sensible to distinguish first names and last names.

```
ALTER TABLE "Addresses" ADD "FirstName" VARCHAR(25) BEFORE "Name";
```

`ALTER TABLE "Addresses"`: Alter the table called Addresses.

`ADD "FirstName" VARCHAR(25)`: insert the field *FirstName* with a length of 25 characters.

`BEFORE "Name"`: before the field Name.

The possibility of specifying the position of additional fields after the creation of the table is not available in the GUI.

```
ALTER TABLE "Table_name" ADD [COLUMN] <Field_definition> [BEFORE
"already_existing_field_name"];
```

The additional designation COLUMN is not necessary in cases where no alternative choices are available.

```
ALTER TABLE "Table_name" DROP [COLUMN] "Field_name";
```

The field Field_name is erased from the table Table_name. However this does not take place if the field is involved in a view or as a foreign key in another table.

```
ALTER TABLE "Table_name" ALTER COLUMN "Field_name" RENAME TO
"New_field_name"
```

Changes the name of a field.

```
ALTER TABLE "Table_name" ALTER COLUMN "Field_name" SET DEFAULT
<Standard value>};
```

Sets a specific default value for the field. NULL removes an existing default value.

```
ALTER TABLE "Table_name" ALTER COLUMN "Field_name" SET [NOT] NULL
```

Sets or removes a NOT NULL condition for a field.

```
ALTER TABLE "Table_name" ALTER COLUMN <Field definition>;
```

The field definition corresponds to the one from the Table creation with the following restrictions:

- The field must already be a primary key field to accept the property IDENTITY. IDENTITY means that the field has the property AutoValue. This is possible only for INTEGER or BIGINT fields. For these field type descriptions, see the Appendix to this handbook.
- If the field already has the property IDENTITY but it is not repeated in the field definition, the existing IDENTITY property is removed.
- The default value will become that specified in the new field definition. If the definition of the default value is left blank, any default already defined is removed.
- The property NOT NULL continues into the new definition, if not otherwise defined. This is in contrast to the default value.
- In some cases, depending on the type of modification, the table must be empty in order for the change to occur. In all cases the change will have effect only if it is possible in principle (for example a change from NOT NULL to NULL) and the existing values can all be translated (for example a change from TINYINT to INTEGER).

```
ALTER TABLE "Table_name" ALTER COLUMN "Field_name" RESTART WITH
<New_field_value>
```

This command is used exclusively for an IDENTITY field. It determines the next value for a field with the Autovalue function set. It can be used, for example, when a database is initially used with test data, and subsequently provided with real data. This requires the contents of the tables to be deleted and a new value such as "1" to be set for the field.

```
ALTER TABLE "Table_name"

ADD [CONSTRAINT "Condition_name"] CHECK (<Search_condition>);
```

This adds a search condition introduced by the word CHECK. Such a condition will not apply retrospectively to existing records, but it will apply to all subsequent changes and newly entered records. If a constraint name is not defined, one will be assigned automatically.

Example:

```
ALTER TABLE "Loan" ADD CHECK (IFNULL("Return_Date","Loan_Date")>="Loan_Date")
```

The Loan table needs to be protected from input errors. For example, you must prevent a return date being given that is earlier than the loan date. Now if this error occurs during the return process, you will get an error message `Check constraint violation …`

```
ALTER TABLE "Table_name"

ADD [CONSTRAINT "Constraint_name"] UNIQUE ("Field_name1", "Field_name2"…);
```

Here a condition is added that forces the named fields to have different values in each record. If several fields are named, this condition applies to the combination rather than the individual fields. `NULL` does not count here. A field can therefore have the same value repeatedly without causing any problems, if the other field in each of the records is `NULL`.

This command will not work if there is already a `UNIQUE` condition for the same field combination.

```
ALTER TABLE "Table_name"

ADD [CONSTRAINT "Constraint_name"] PRIMARY KEY ("Field_name1", "Field_name2"…);
```

Adds a primary key, optionally with a constraint, to a table. The syntax of the constraint is the same as when a table is created.

```
ALTER TABLE "Table_name" ADD [CONSTRAINT "Constraint_name"] FOREIGN KEY ("Field_name1", "Field_name2"…)

REFERENCES "Table_name_of_another_table" ("Field_name1_other_table", "Field_name2_other_table"…)

[ON {DELETE | UPDATE} {CASCADE | SET DEFAULT | SET NULL}];
```

This adds a foreign key (`FOREIGN KEY`) to the table. The syntax is the same as when a table is created.

The operation will terminate with an error message, if any value in the table does not have a corresponding value in the table containing that primary key.

Example: The Name and Address tables are to be linked. The Name table contains a field with the name Address_ID. The value of this should be linked to the ID field in the Address table. If the value **1** is found in Address_ID but not in the ID field of the Address table, the link will not work. It will not work either if the two fields are of different types.

```
ALTER TABLE "Table_name" DROP CONSTRAINT "Constraint_name";
```

This command removes the named constraint (`UNIQUE, CHECK, FOREIGN KEY`) from a table.

```
ALTER TABLE "Table_name" RENAME TO "new_table_name";
```

Finally this command changes only the name of a table.

Note

When you change a table using SQL, the change affects the database but is not necessarily apparent or effective in the GUI. When the database is closed and reopened, the changes appear in the GUI too.

Changes are also displayed if you choose **View > Refresh Tables** in the table container.

Deleting tables

```
DROP TABLE "Table name" [IF EXISTS] [RESTRICT | CASCADE];
```

Deletes the *Table name* table.

`IF EXISTS` prevents an error occurring if this table does not exist.

`RESTRICT` is the default arrangement and need not be explicitly chosen; it means that deletion does not occur if the table is linked to another table by the use of a foreign key or there is an active view of this table. Queries are not affected as they are not stored within HSQLDB.

If instead you choose `CASCADE`, all links to the table Table_name are deleted. In the linked tables, all foreign keys are set to NULL. All views referring to the named table are also completely deleted.

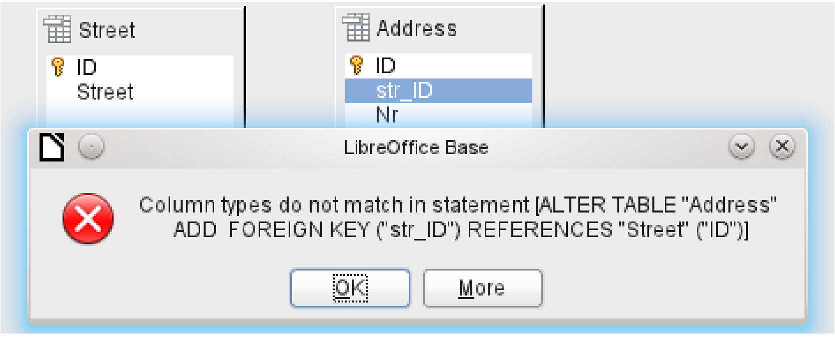
Linking tables

In principle you can have a database without links between tables. The user must then ensure during data entry, that the relationships between the tables remain correct. This usually occurs through the use of suitable input forms that manage this.

Deleting records in linked tables is not a simple matter. Suppose you wish to delete a particular street from the Street table in Figure 130, where this field is linked with the Address table as a foreign key in that table. The references in the Address table would disappear. The database does not allow this, once the relationship has been created. In order to delete the Street, the precondition must be fulfilled, that it is no longer referenced in the Address table.

Basic links are made using **Tools > Relationships**. This creates a connection line from the primary key in one table to the defined foreign key in the other.

You may receive the following error message when creating such a link:

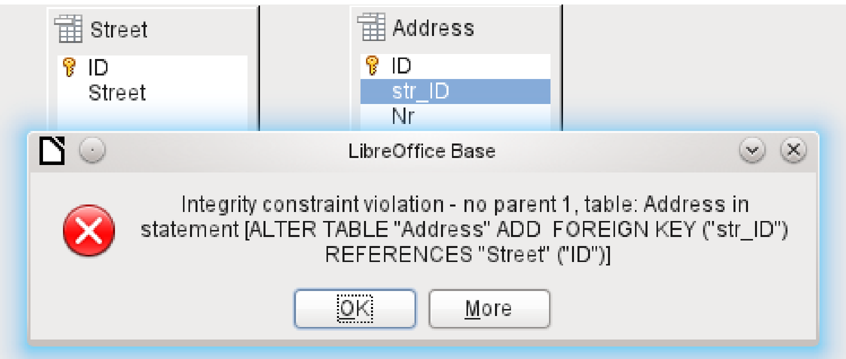


Column types do not match in statement [ALTER TABLE "Address" ADD FOREIGN KEY ("str_ID") REFERENCES "Street" ("ID")]

Figure 151: Column types do not match when creating a relationship

This message shows the error that occurred and the internal SQL command that caused the error.

`Column types do not match in statement`—As the SQL command is displayed as well, the reference is clearly to the columns Address.str_ID and Street.ID. For test purposes one of these fields was defined as an Integer, the other as Tiny Integer. Therefore no link could be created since the one field cannot have the same value as the other.



Integrity constraint violation - no parent 1, table: Address in statement [ALTER TABLE "Address" ADD FOREIGN KEY ("str_ID") REFERENCES "Street" ("ID")]

Figure 152: Integrity constraint violation when creating a relationship

In this case the column types match. The SQL statement is the same as in the first example. But again there is an error:

`Integrity constraint violation – no parent 1, table: Address...` —The integrity of the relationship could not be established. In the str_ID field of the Address table, there is a number **1**, which is not present in the ID field of the Street table. The parent table here is Street, since its primary key is the one that must exist. This error is very common, when two tables are to be linked and some fields in the table with the prospective foreign key already contain data. If the foreign key field contains an entry that is not present in the parent table (the table containing the primary key), this is an invalid entry.

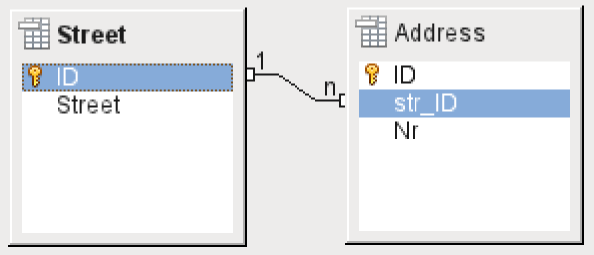
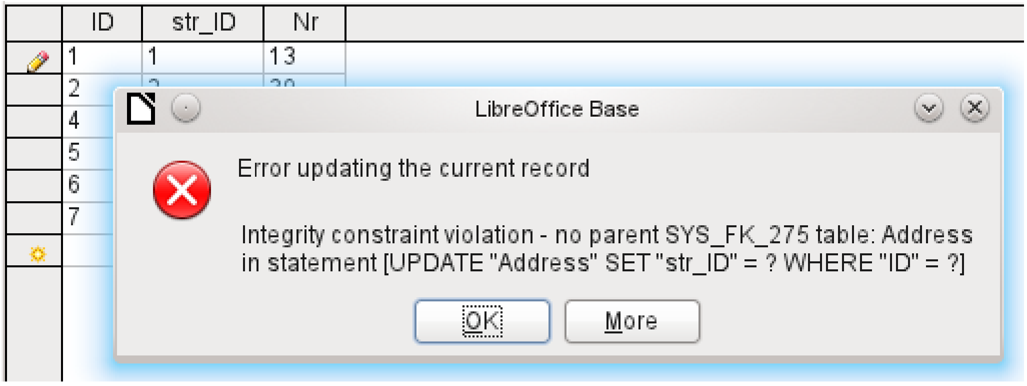


Figure 153: Relationship successfully created

If the linking is carried out successfully and subsequently there is an attempt to enter a similarly invalid record into the table, you get the following error message:



ID	str_ID	Nr
1	1	13
2	2	20
4		
5		
6		
7		

Error updating the current record

Integrity constraint violation - no parent SYS_FK_275 table: Address in statement [UPDATE "Address" SET "str_ID" = ? WHERE "ID" = ?]

Figure 154: Integrity constraint violation when entering data

Again this is an integrity violation. Base refuses to accept the value 1 for the field str_ID after the link has been made because the Street table contains no such value in the ID field.

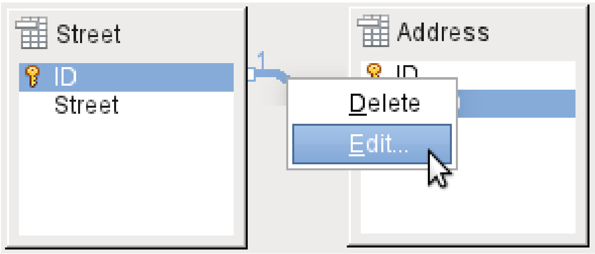


Figure 155: Edit links with a right-click

The properties of a link can be edited so that the deletion of a record from the Street table will simultaneously set to NULL the corresponding entries in the Address table.

The properties shown in Figure 155 always relate to an action linked to the change in a record from the table containing the corresponding primary key. In our case this is the Street table. If the *primary key of a record* in this table is *altered (Update)*, the following actions might take place.

No action

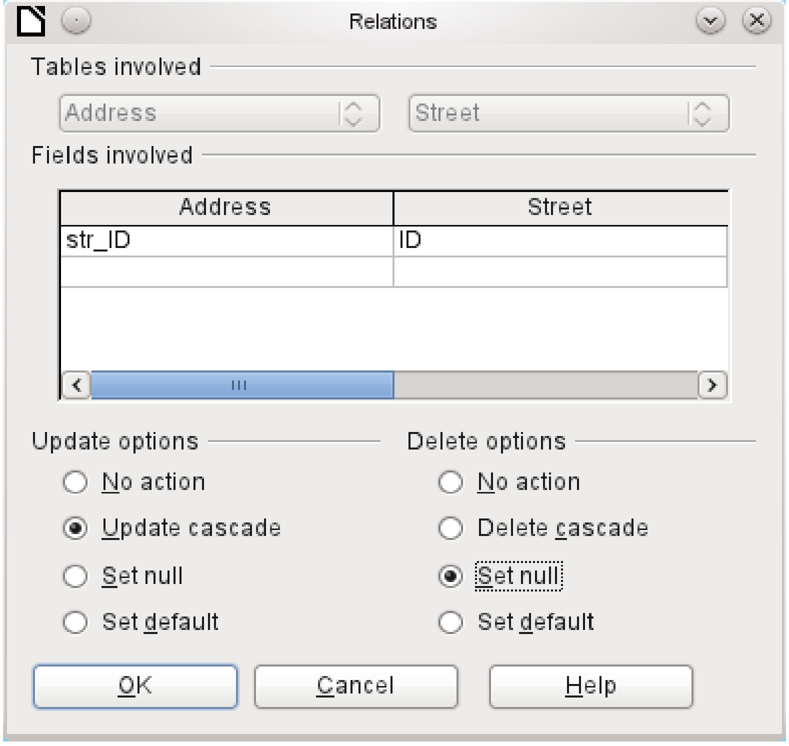
Changing the primary key Street.ID is not allowed in this case, as it would break the relationship between the tables.

Update cascade

If the primary key Street.ID is changed, the foreign key is automatically changed to its new value. This ensures that the linkage is not damaged. For example, if a value is changed from 3 to 4, all records from the Address table that contain the foreign key Address.Street_ID with the value 3, have it changed to 4.

Set null

All records which contain this particular primary key will now have no entry in the foreign key field Address.Street_ID; the field will be NULL.



Tables involved

Address | Street

Fields involved

Address	Street
str_ID	ID

Update options

○ No action
● Update cascade
○ Set null
○ Set default

Delete options

○ No action
○ Delete cascade
● Set null
○ Set default

Figure 156: Editing the properties of a relationship

Set default

If the primary key Street_ID is changed, the value of Address.Street_ID originally linked to it is set to the previously defined default value. For this purpose we need an unambiguous definition of a default value. If the default is set using the SQL statement:

```
ALTER TABLE "Address" ALTER COLUMN "Street_ID" SET DEFAULT 1;
```

the link definition ensures that the field will return to this value in the case of an Update. So if the primary key in the Street table is changed, the corresponding foreign key in the Address table will be set to 1. This is useful when a record is required to have a street field, in other words this field cannot be NULL. But be careful: if 1 is not in use, you will have created a link to a non-existent value. It is therefore possible to destroy the integrity of the relationship.

Caution

If the default value in a foreign key field is not linked to a primary key value of the foreign table, a link to a value would be created that isn't possible. The referential integrity of the database would be destroyed.

It would be better **not** to use the possibility to set the value to default.

If a record is *deleted* from the Street table, the following options are available.

No Action

No action takes place. If the requested deletion affects a record in the Address table, the request will be refused.

Cascading Delete

If a record is deleted from the Street table and this affects a record in the Address table, that record will also be deleted.

That might seem strange in this context but there are other table structures in which it makes sense. Suppose you have a table of CDs and a table which stores the titles on these CDs. Now if a record in the CD table is deleted, many titles in the other table have no meaning as they are no longer available to you. In such cases, a cascading deletion makes sense. It means that you do not need to delete all the titles before deleting the CD from the database.

Set to Null

This is the same as for the update option.

Set to Default

This is the same as for the update option and requires the same precautions.

Tip

The No Action option should be avoided in most cases in order to avoid displaying error messages from the database to the user, since these may not always be comprehensible to the user.

In **Tools > Relationships**, dragging with the mouse creates foreign keys that refer to a single field in another table. To link to a table that has a composite primary key, go to **Tools > Relationships**, then **Insert > New Relation,** or use the corresponding button. A dialog appears for editing the properties of a relationship with a free choice of available tables.

Entering data into tables

Databases that consist of only a single table usually do not require an input form unless they contain a field for images. However as soon as a table contains foreign keys from other tables, users must either remember which key numbers to enter or they must be able to look at the other tables simultaneously. In such cases, a form is useful.

Entry using the Base GUI

Tables in the table container are opened by double-clicking them. If the primary key is an automatically incrementing field, one of the visible fields will contain the text *AutoValue*. No entry is possible into the AutoValue field. Its assigned value can be altered if required, but only after the record has been committed.

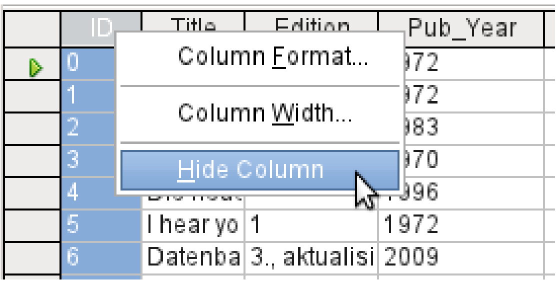


Figure 157: Entry into tables – Hiding columns

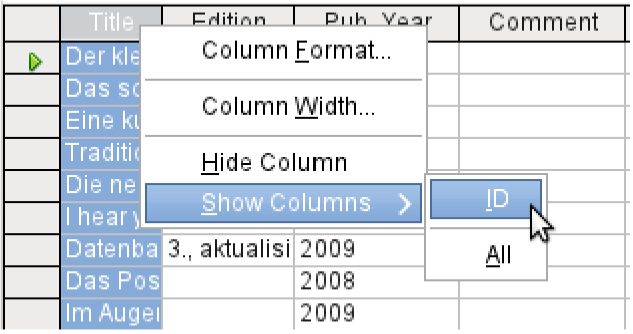


Figure 158: Entry into tables – Unhiding columns

Individual columns in the Table Data View can be hidden. For example, if the primary key field does not need to be visible, this can be specified in the table in data entry view by right-clicking on the column header. This setting is stored with the GUI. The column continues to exist in the table and can always be made visible again.

Entry into the table usually takes place from left to right using the keyboard with the Tab or Enter keys. You can also use the mouse.

When you reach the last field of a record, the cursor automatically jumps to the next record. The previous entry is committed to storage. Additional storage using **File > Save** is not necessary and indeed not possible. The data is already in the database.

Caution

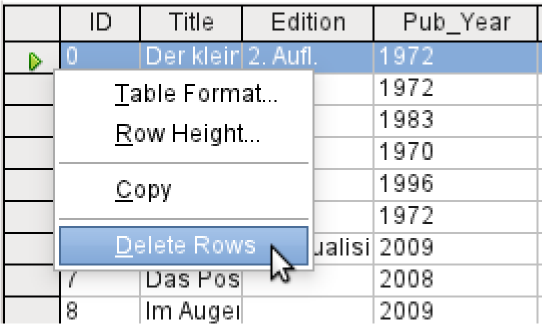
For the HSQLDB, data is in working memory. It will only be transferred to the hard drive when Base is closed. This is not always desirable from the viewpoint of data security because if Base for some reason does not close down in an orderly fashion, it can lead to loss of data.

If no data is entered into a field that has been previously defined during table design as mandatory (`NOT NULL`), the appropriate error message is displayed:

```
Attempt to insert null into a non-nullable column …
```

The corresponding column, the table and the SQL command (as translated by the GUI) are also displayed.

Changing a record is easy: find the field, enter a different value, and leave the row again.

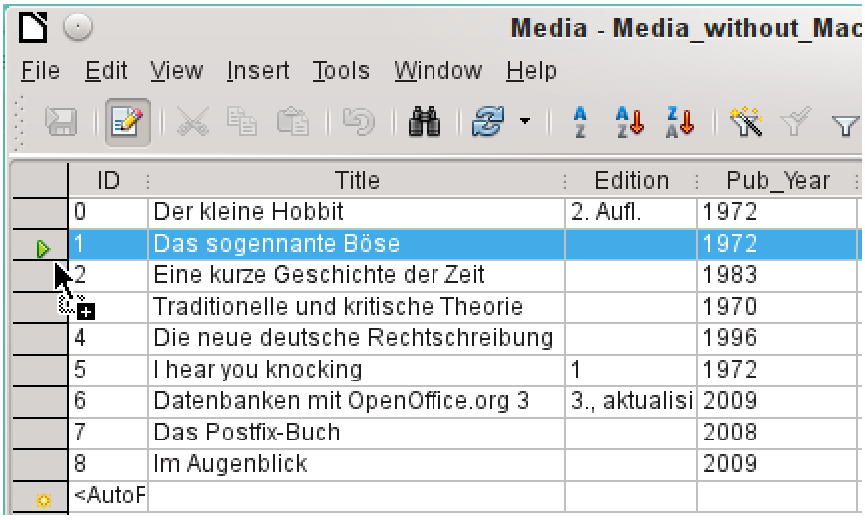


ID	Title	Edition	Pub_Year
0	Der kleir	2. Aufl.	1972
			1972
			1983
			1970
			1996
			1972
		ualisi	2009
7	Das Pos		2008
8	Im Augei		2009

Figure 159: Delete table row(s)

To delete a record, select the row by clicking its header (the grey area to the left), right-click and choose **Delete Rows**.

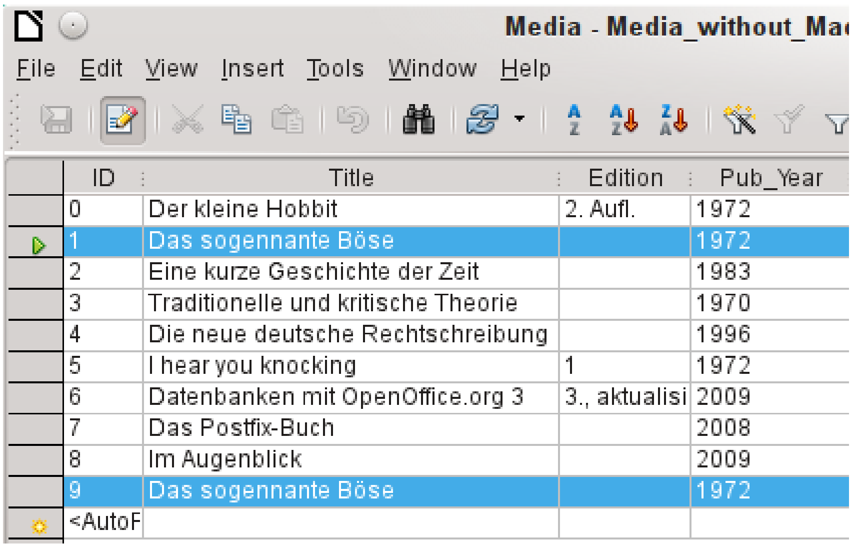
There is a method, rather well hidden, to copy complete rows. For this to work, the primary key of the table must be defined as AutoValue.



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

Figure 160: Copy table row

First, the row header is clicked with the left mouse button. Next, hold down the button and drag the mouse. The cursor will change to a symbol with a + sign. This means that the record is copied to the last entry to the table.



ID	Title	Edition	Pub_Year
0	Der kleine Hobbit	2. Aufl.	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	1	1972
6	Datenbanken mit OpenOffice.org 3	3., aktualisi	2009
7	Das Postfix-Buch		2008
8	Im Augenblick		2009
9	Das sogenannte Böse		1972
<AutoF			

Figure 161: Row with ID 1 copied to end of table as ID 9

The record with the primary key **1** is inserted as a new record with the new primary key **9**.

If the control or shift key is used to highlight a group of records, these will be copied as a group.

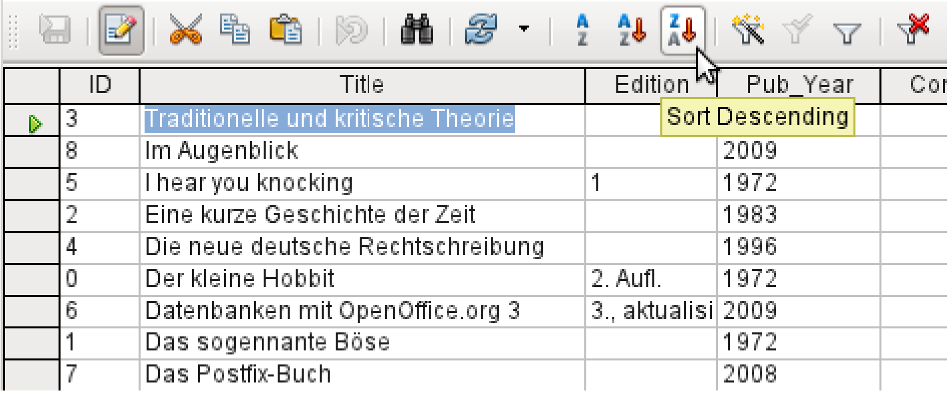
Tip

The column headers can be dragged to a suitable width for input. If this is done in a table, Base automatically saves the new column width in the table.

The column widths in tables affect those in queries. If the columns in a query are too narrow, widening them will have only a temporary effect. The new width will not be saved. It is better to widen the column in the table so that it will appear properly in queries without the need to resize.

The Sort, Search, and Filter functions are very useful for retrieving particular records.

Sorting tables

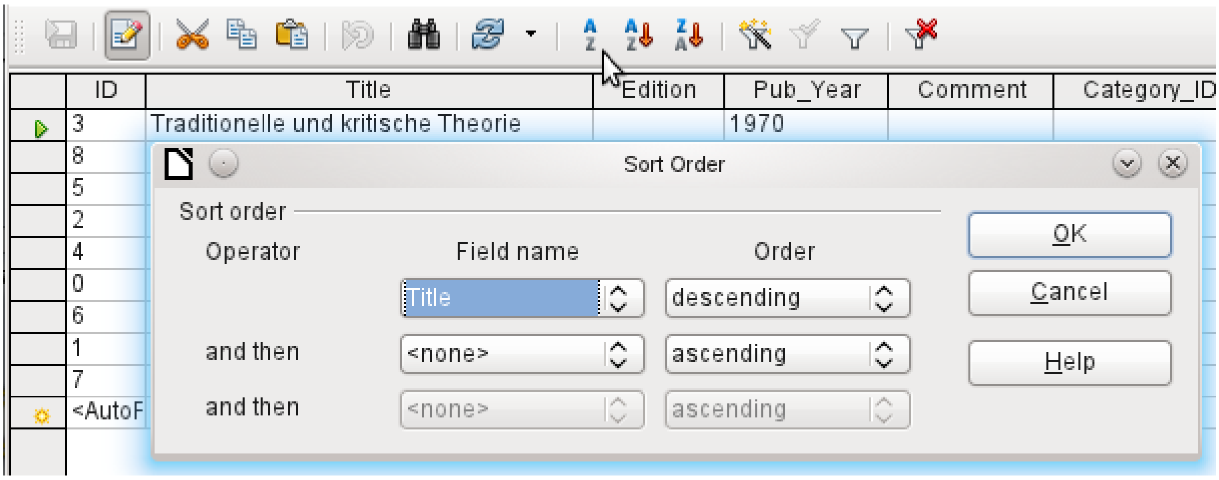


ID	Title	Edition	Pub_Year	Cor
3	Traditionelle und kritische Theorie			
8	Im Augenblick		2009	
5	I hear you knocking	1	1972	
2	Eine kurze Geschichte der Zeit		1983	
4	Die neue deutsche Rechtschreibung		1996	
0	Der kleine Hobbit	2. Aufl.	1972	
6	Datenbanken mit OpenOffice.org 3	3., aktualisi	2009	
1	Das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

Figure 162: Quick sort

The *A* > *Z* and Z > A buttons allow for quick sorting. First, select a field. Then, click on the button corresponding to ascending or descending sort, and the data is sorted by that column. Figure 162 shows a descending sort by the Title field.

Quick sort will only sort by one column. To sort by several columns simultaneously, a more advanced sort function is provided to the left of the quick sort buttons:



ID	Title	Edition	Pub_Year	Comment	Category_ID
3	Traditionelle und kritische Theorie		1970		
8					
5					
2					
4					
0					
6					
1					
7					
<AutoF					

Sort Order

Sort order

Operator	Field name	Order
	Title	descending
and then	<none>	ascending
and then	<none>	ascending

Figure 163: Sorting by more than one column

The field name of the column and the current sort order are selected. If a previous quick sort has been carried out, the first row will already contain the corresponding field name and sort order.

Searching tables

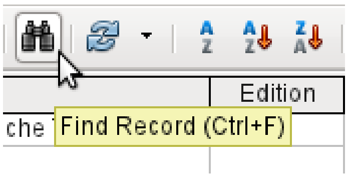


Figure 164: Find Record icon on Table Data toolbar

The **Find Record** button is a simple method to locate records in a large table. However, the search function is very slow for large databases, as the search does not use a SQL command within the database. For a quicker search, instead of using Find Record, use a query. To eliminate frequent modification of the query, it can be designed to run using parameters. See Chapter 5, Queries, in the section “Using Parameters in Queries”.

Tip

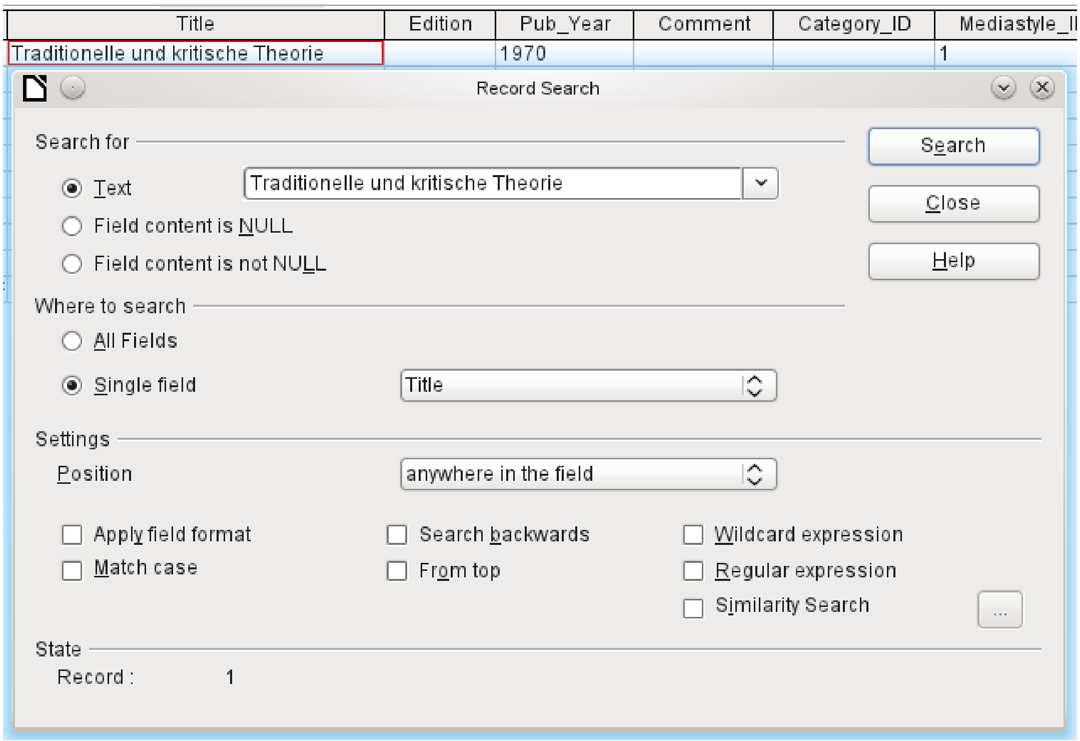
Before you search, make sure the columns you will be searching are wide enough to show correctly the records that you will find. The search window remains in the foreground and you will not be able to correct the settings for column width in the underlying table. To reach the table, you must break off the search.

The Find Record button automatically populates the search term with the contents of the field from which it was invoked.

To make the search effective, the search area should be limited as far as possible. It would be pointless to search for the above text from the Title field from the Author field. Instead, the field name Title is already suggested as the single Field name.

Further settings for the search can make things easier through specific combinations. You can use the normal SQL placeholders ("_" for a variable character, "%" for an arbitrary number of variable characters, or "\" as an escape character to enable these special characters themselves to be searched for).

Regular expressions are described in detail in LibreOffice Help.



Title	Edition	Pub_Year	Comment	Category_ID	Mediastyle_I
Traditionelle und kritische Theorie		1970			1

Record Search

Search for

- ● Text: Traditionelle und kritische Theorie
- ○ Field content is NULL
- ○ Field content is not NULL

Where to search

- ○ All Fields
- ● Single field: Title

Settings

Position: anywhere in the field

- □ Apply field format
- □ Match case
- □ Search backwards
- □ From top
- □ Wildcard expression
- □ Regular expression
- □ Similarity Search

State

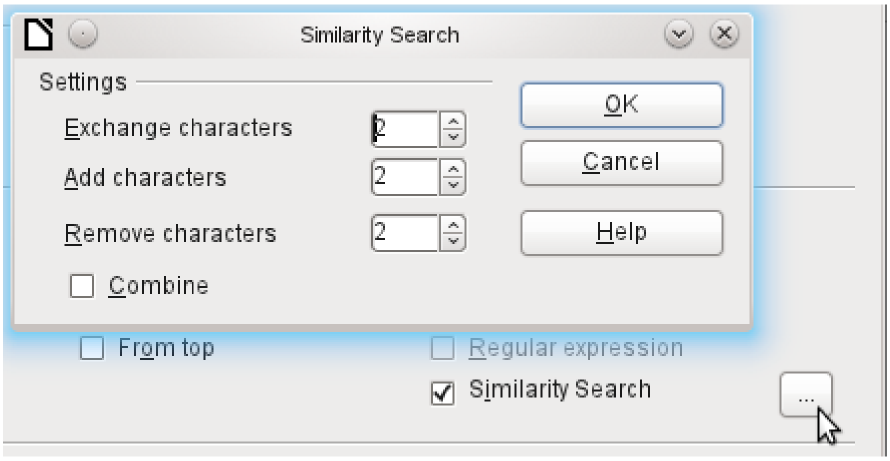
Record : 1

Figure 165: Entry mask for a record search

The similarity search function is useful when you need to exclude spelling mistakes. The higher the values that you set, the more records will be shown in the final list.

This search module is most suitable for people who know, from regular use, exactly how to achieve a given result. Most users are more likely to succeed in finding records by using a filter.

Chapter 4 of this book describes the use of forms for searching, and how the use of SQL and macros can accomplish a keyword search.



Similarity Search

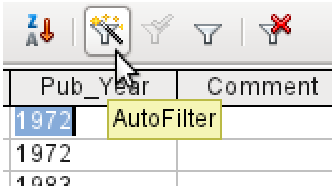
Settings

- Exchange characters: 2
- Add characters: 2
- Remove characters: 2
- □ Combine

- □ From top
- □ Regular expression
- ☑ Similarity Search

Figure 166: Limiting the similarity search

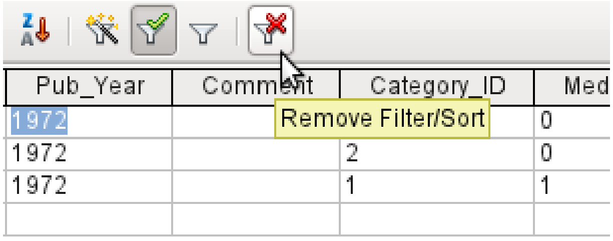
Filtering tables



Pub_Year	Comment
1972	
1972	

Figure 167: AutoFilter icon on Table Data toolbar

You can filter a table quickly by using the **AutoFilter**. Place the cursor in the cell of a field, and one click on the icon causes the filter to take over the content of this field. Only those records are shown for which the chosen field has the same content as the selected cell. The figure below shows filtering according to an entry in the **Pub_Year** column.



Pub_Year	Comment	Category_ID	Med
1972			0
1972		2	0
1972		1	1

Figure 168: Reset Filter/Sort icon on Table Data toolbar

The filter is active, as shown by the filter icon with a green check mark. The filter symbol is shown pressed in. This button is a toggle, so if it is clicked again, the filter continues to exist, but all records are now shown. So, if you want, you can always click it to return to the filtered state.

Clicking on the **Remove Filter/Sort** icon at the extreme right causes all existing filters and sorts to be removed. The filters become inactive and can no longer be recovered with their old values.

Tip

You can still enter records normally into a filtered table or one that has been restricted by a search. They remain visible in the table view until the table is updated by clicking the **Refresh** button.

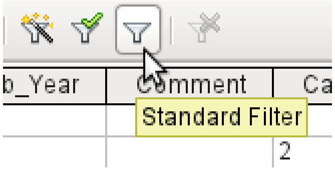
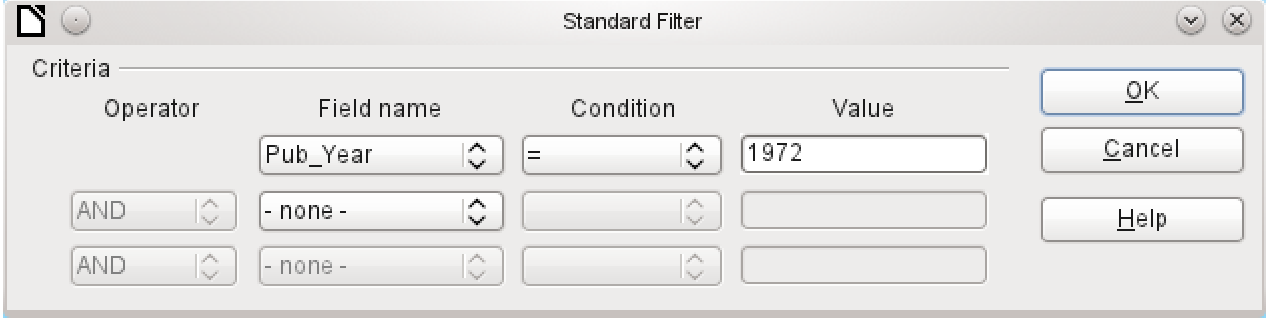


Figure 169: Standard Filter icon on Table Data toolbar

The **Standard Filter** icon opens a dialog in which you can filter using several simultaneous criteria, similar to sorting. If AutoFilter is in use, the first line of the Standard Filter will already show this existing filter criterion.



Operator	Field name	Condition	Value
	Pub_Year	=	1972
AND	- none -		
AND	- none -		

Figure 170: Multiple data filtering using the Standard Filter

The Standard Filter provides many of the functions of SQL data filtering. The following SQL commands are available.

Table 4: Conditions available for Standard Filters

GUI Condition	***Description***
=	Exact equality; corresponds to **like**, but without any additional placeholders
<>	Unequal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
like	For text, written in quotation marks (' '); "_" for a variable character, "%" for an arbitrary number of variable characters
not like	Opposite of `like`, in SQL `NOT LIKE`
empty	No entry, not even a space character. In SQL this is expressed by the term `NULL`
Not empty	Opposite of empty, in SQL `NOT NULL`

Before one filter criterion can be combined with another, the following row must have at least one field name selected. In Figure 170, the word – none – is shown instead of a field name, so the combination is not active. The combination operators available are `AND` and `OR`.

The field name can be a new field name or a previously selected one.

Even for large data collections, the number of retrieved records can be reduced to a manageable set with skillful filtering using these three possible conditions.

In the case of filtering forms too, there are some further possibilities (described in Chapter 4) which are not provided by the GUI.

Direct entry using SQL

Direct data entry using SQL is useful for entering, changing or removing multiple records with one command.

Entering new records

```
INSERT INTO "Table_name" [( "Field_name" [,…] )]
{ VALUES("Field value" [,…]) | <Select-Formula>};
```

If no Field_name is specified, all fields must be completed and in the right order (as laid down in the table). That includes the automatically incremented primary key field, where present. The values entered can also be the result of a query (`<Select-Formula>`). More exact information is given below.

```
INSERT INTO "Table_name" ("Field_name") VALUES ('Test');
CALL IDENTITY();
```

In the table, in the column Field_name, the value `Test` is inserted. The automatically incremented primary key field ID is not touched. The corresponding value for ID needs to be created separately by using `CALL IDENTITY()`. This is important when you are using macros, so that the value of this key field can be used later on.

```
INSERT INTO "Table_name" ("Field_name") SELECT "Other_fieldname" FROM
"Name_of_other_table";
```

In the first table, as many new records are inserted into Field_name, as are present in the column Other_fieldname in the second table. Naturally a Select-Formula can be used here to limit the number of entries.

Editing existing records

```
UPDATE "Table_name" SET "Field_name" = <Expression> [, ...] [WHERE
<Expression>];
```

When you are modifying many records at once, it is very important to check carefully the SQL command you are entering. Suppose that all students in a class are to be moved up one year:

```
UPDATE "Table_name" SET "Year" = "Year"+1
```

Nothing could be faster: All data records are altered with a single command. But imagine that you must now determine which students should not have been affected by this change. It would have been simpler to check a Yes/No field for the repetition of a year and then to move up only those students for which this field was not checked:

```
UPDATE "Table_name" SET "Year" = "Year"+1 WHERE "Repetition" = FALSE
```

These conditions only function when the field in question can only take the values `FALSE` and `TRUE`; it may not be `NULL`. It would be safer if the condition were formulated as `WHERE` "`Repetition`" <> `TRUE`.

If you should subsequently want a default value to be entered in a particular field wherever this is empty, you can do this with the command:

```
UPDATE "Table" SET "Field" = 1 WHERE "Field" IS NULL
```

You can alter several fields at once by directly assigning values to them. Suppose that a table for books includes the names of their authors. It is discovered that Erich Kästner has frequently been entered as Eric Käschtner.

```
UPDATE "Books" SET "Author_first_name" = 'Erich', "Author_surname" = 'Kästner'
WHERE "Author_first_name" = 'Eric' AND "Author_surname" = 'Käschtner'
```

Other calculation steps are also possible with Update. If, for example, wares costing more than $150.00 are to be included in a special offer and the price reduced by 10%, this can be carried out as follows:

```
UPDATE "Table_name" SET "Price" = "Price"*0.9 WHERE "Price" >= 150
```

When you choose the data type CHAR, the field has a fixed width. Where necessary, text is padded with null characters. If you convert this to VARCHAR, these null characters remain. To remove them use the right-trim function:

```
UPDATE "Table_name" SET "Field_name" = RTRIM("Field_name")
```

Deleting existing records

```
DELETE FROM "Table_name" [WHERE <Expression>];
```

Without the conditional expression the command

```
DELETE FROM "Table_name"
```

deletes the entire content of the table.

For this reason it is preferable for the command to be more specific. For example, if the value of the primary key is given, only this precise record will be deleted.

```
DELETE FROM "Table_name" WHERE "ID" = 5;
```

If, in the case of a loan, the media record is to be deleted when the item is returned, this can be done using

```
DELETE FROM "Table_name" WHERE NOT "Return_date" IS NULL;
```

or alternatively with

```
DELETE FROM "Table_name" WHERE "Return_date" IS NOT NULL;
```

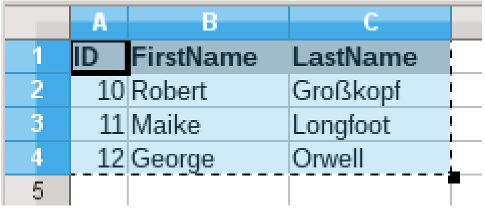
Importing data from other sources

Sometimes there are complete data sets in another program which need to be imported into Base via the clipboard. This may involve creating a new table or adding records to an existing one.

Note

To import data using the clipboard, the data format must be readable by Base. This will always be the case for data files opened in LibreOffice.

For example, if tables from an external database are to be read into an *.odb file, that database must first be opened in LibreOffice or registered with LibreOffice as a data source. See “Accessing external databases” in Chapter 2, Creating a Database.



	A	B	C
1	ID	FirstName	LastName
2	10	Robert	Großkopf
3	11	Maike	Longfoot
4	12	George	Orwell
5			

Figure 171: Data to be copied into a table

Above a small example table has been copied from the spreadsheet program Calc onto the clipboard. Then it is pasted into Base’s Table container. Of course this could also have been done by selecting it with the left mouse button and then dragging it across.

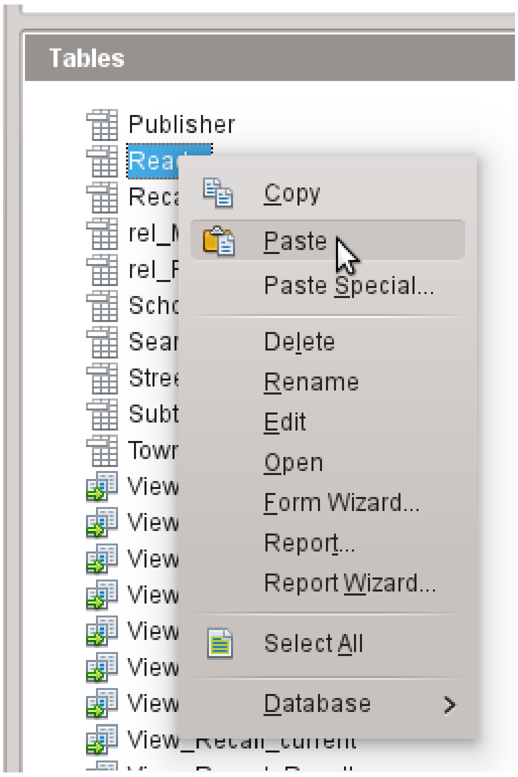


Figure 172: Paste data into required table

In the Table container, right-click to open the context menu for the table to which the records are to be added.

Adding imported records to an existing table

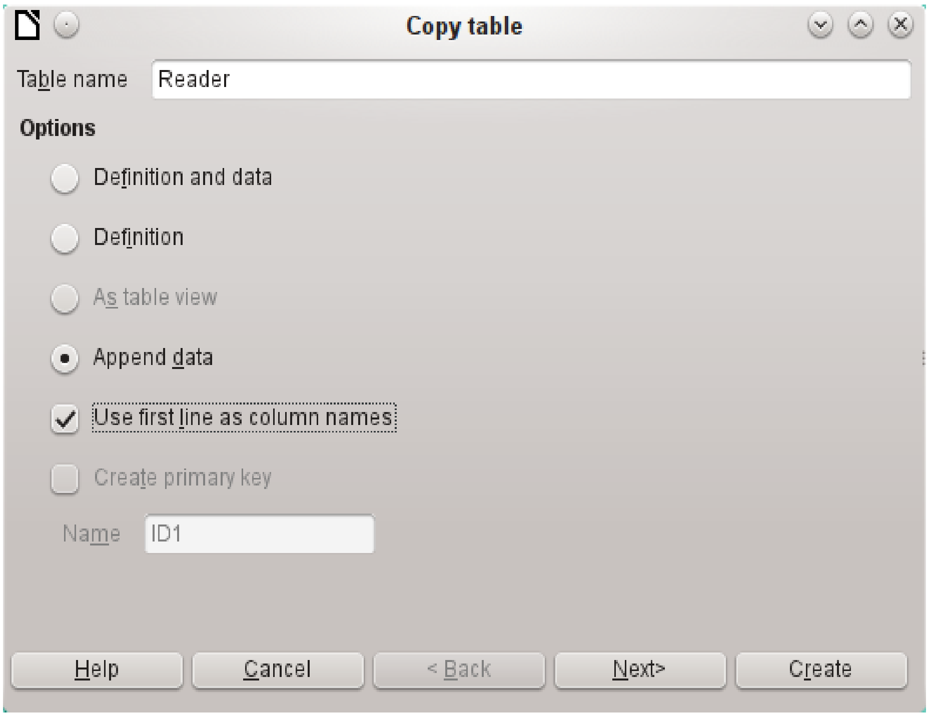


Table name: Reader

Options

○ Definition and data

○ Definition

○ As table view

● Append data

☑ Use first line as column names

□ Create primary key

Name: ID1

Figure 173: Copy table dialog - appending data

The name of the table appears in the Import wizard. At the same time *Append data* is selected. *Use first line as column names* may or may not be required, depending on your version of

LibreOffice. If the records are to be appended, then no data definition is required. A primary key must also be available for use.

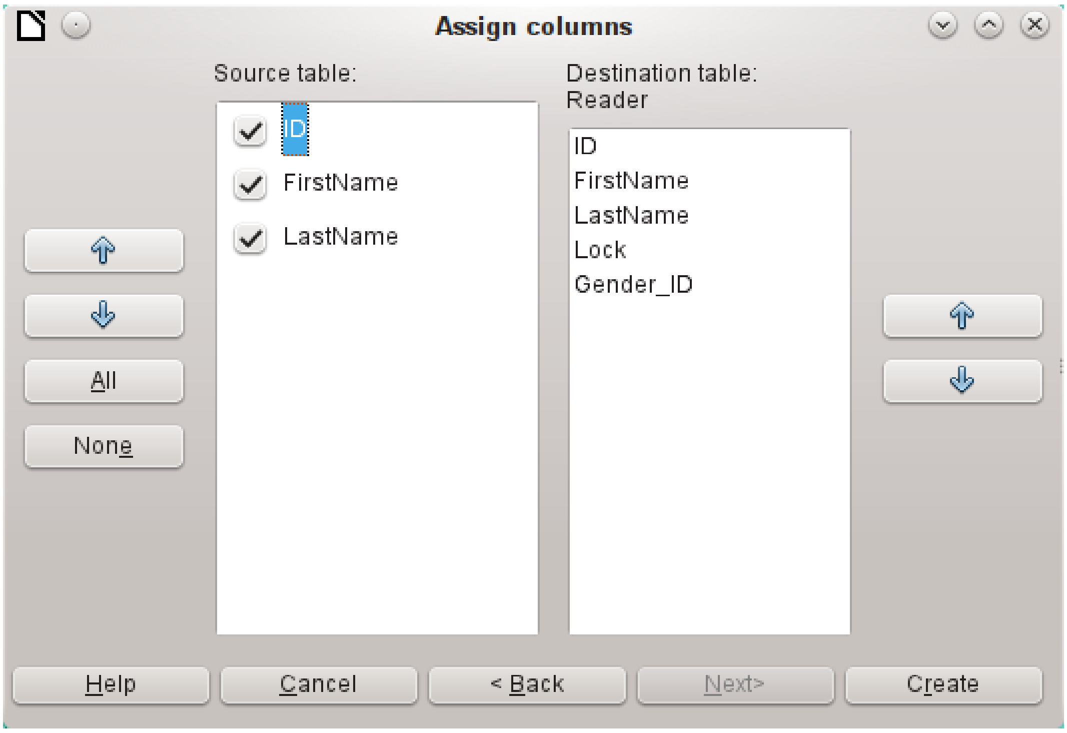


Figure 174: Assign columns data - appending data

The columns of the Calc source table and the destination table in Base do not have to agree in their sequence, names, or overall number. Only the elements selected from the left hand side are transferred. The correspondence between source and destination tables must be adjusted using the arrow buttons to either side.

This completes the import.

The import can lead to problems if:

- Fields in the destination table require a mandatory entry, but the source table provides no data for them.
- Field definitions in the destination table are inconsistent with those in the source table (for instance, a name is to be entered into a numeric field, or the destination field has too few characters for the data).
- The source table provides data inconsistent with those of the destination table, for example non-unique values for primary keys or other fields defined as unique.

Creating a new table for imported data

When the Import wizard is launched, the previously selected table name appears automatically. This table name must be changed if you are creating a new table, as it is forbidden to have a table with the same name as an existing one. The name of this table is Names. *Definition and data* are to be transferred. The first row is to be used as column headers.

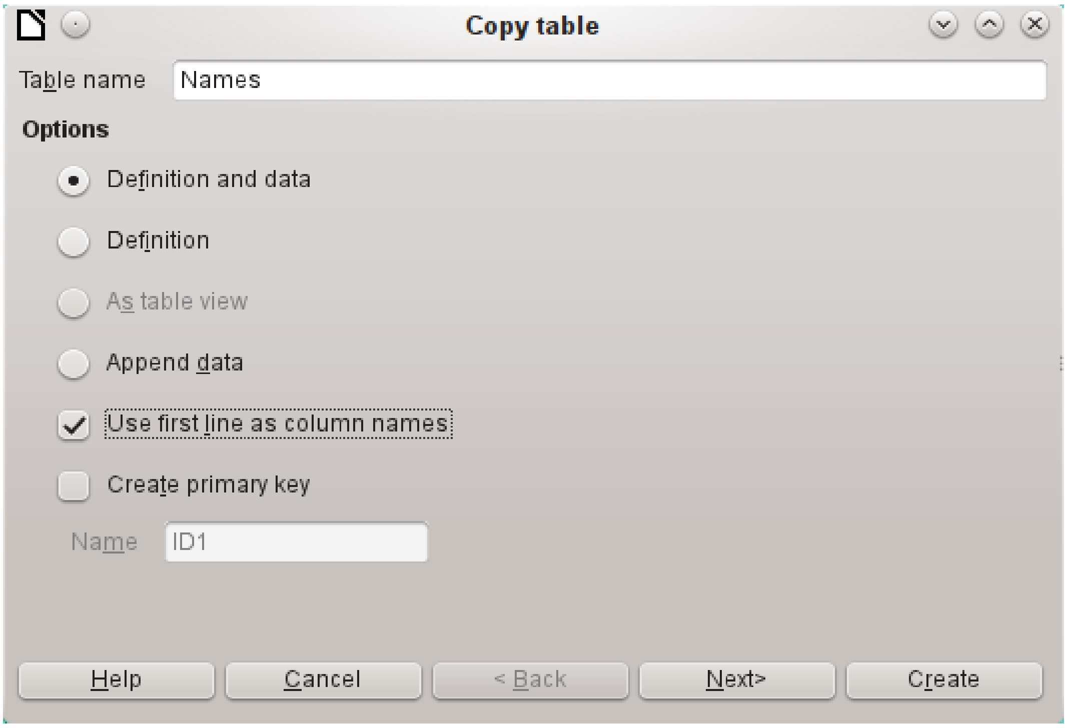
At this point you can create a new, additional field for a primary key. The name of this field must not exist as a column header in the Calc table. Otherwise you get the error message:

The following fields are already set as primary keys: ID

However, this message does not explain the situation quite correctly.

If you want an existing field to serve as your primary key, do not select *Create primary key*. In this case you will establish your primary key field on the third page of the Wizard dialog.

On import, the table definition and data are transferred.



Copy table

Table name: Names

Options

● Definition and data

○ Definition

○ As table view

○ Append data

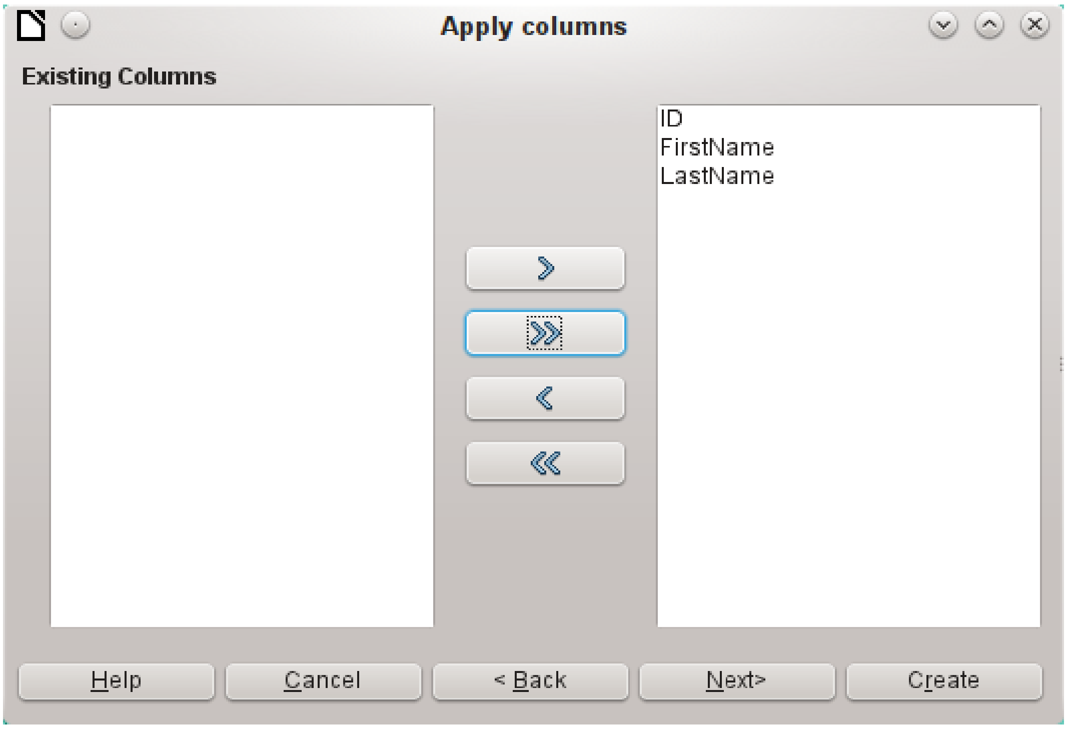
☑ Use first line as column names

□ Create primary key

Name: ID1

Figure 175: Copy table dialog - creating a new table

All available columns are transferred.



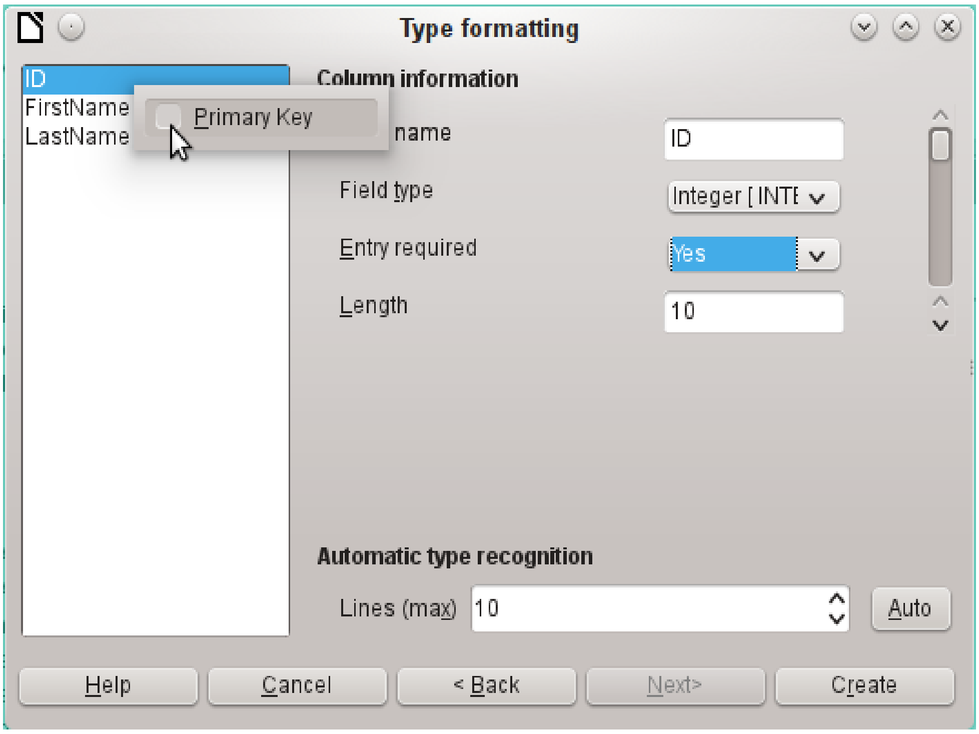
Apply columns

Existing Columns

ID
FirstName
LastName

Figure 176: Apply columns dialog

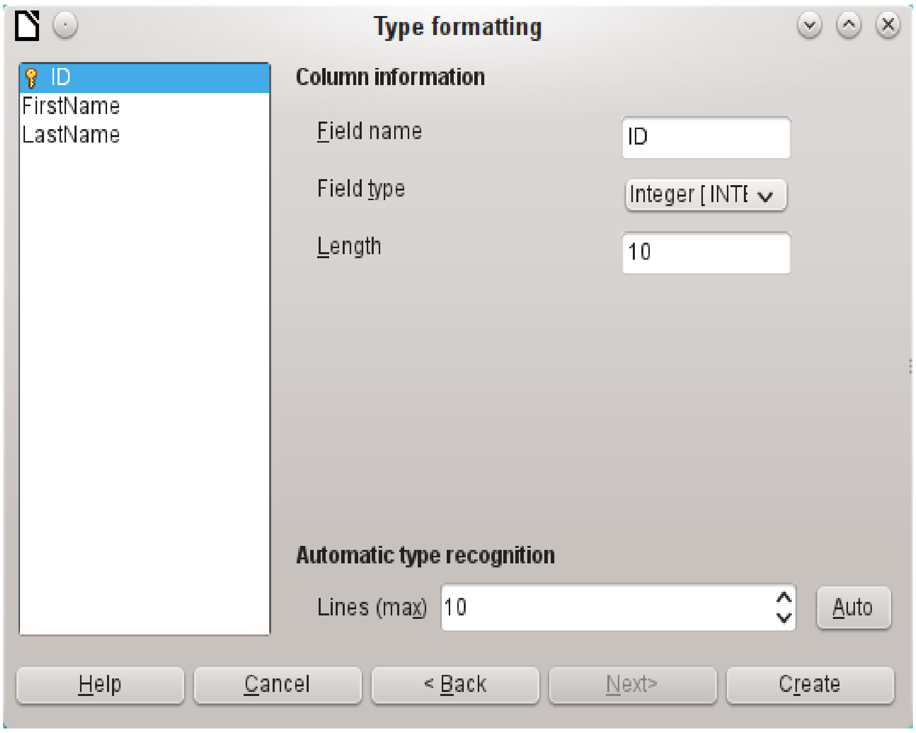
The formatting of table types often requires adjustment. Usually the fields have been predefined as text fields with a very large size. Numeric and date fields should therefore be reset using **Type formatting > Column information > Field type**. In the case of decimal numbers, you will need to check the number of decimal places.



Field name: ID
Field type: Integer [INTE
Entry required: Yes
Length: 10
Lines (max): 10

Figure 177: Type formatting dialog - setting the primary key

The option to choose a primary key is present, somewhat obscurely, in the context menu of the field that is to contain it. In this example, the ID field has been formatted in a way that will allow its use as a primary key. This must now be set explicitly using the context menu of the field name, if a primary key was not created as an additional field in the Copy table window of the wizard.



Field name: ID
Field type: Integer [INTE
Length: 10
Lines (max): 10

Figure 178: Type formatting dialog - ID set as the primary key

When you click the **Create** button, the table is created and filled with the copied data.

The new primary key is not an AutoValue key. To create one of these, the table must be opened for editing. You can then carry out further formatting operations.

Splitting data on import

Sometimes source data are not available in the desired form. Addresses, for example, are often entered into spreadsheets as a single field, including the town and postcode. When importing these, you might wish to place those elements in a separate table, which can then be linked to the main table.

The following is a possible way to create this relationship directly:

1) The complete table with all address information is imported into Base as a table called Addresses. See the previous chapters for details.
2) The Postcode and Town fields are read with a query, copied and stored as a separate Postcode_Town table. For this, an ID field is added and specified as a primary key with AutoValue.

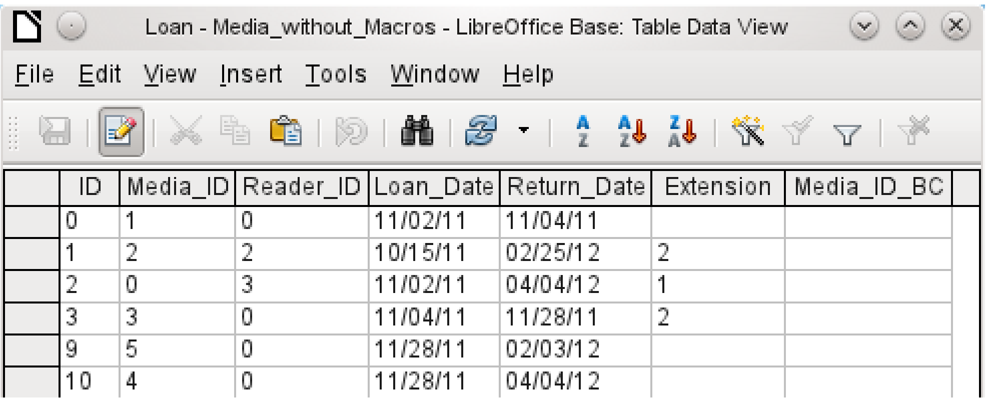
   Here is the query:
   ```
   SELECT DISTINCT "Postcode", "Town" FROM "Addresses"
   ```
3) A new field called Postcode_ID is added to the Addresses table.
4) Using **Tools > SQL**, an update is carried out for this table:
   ```
   UPDATE "Addresses" AS "a" SET "a"."Postcode_ID" = (SELECT "ID" FROM
   "Postcode_Town" WHERE "Postcode"||"Town" = "a"."Postcode"||"a"."Town")
   ```
5) The Addresses table is opened for editing and the Postcode and Town fields deleted. This change is saved and the table closed again.

This separates the tables so that a 1:n relationship can be created between the Postcode_Town table and the Addresses table. This relationship is defined using **Tools > Relationships.**

For details on SQL code, see also Chapter 5, Queries.

Problems with these data entry methods

Entry using a table alone takes no account of links to other tables. This is clear from an example of a media loan.


Loan - Media_without_Macros - LibreOffice Base: Table Data View
File
Edit
View
Insert
Tools
Window
Help

ID	Media_ID	Reader_ID	Loan_Date	Return_Date	Extension	Media_ID_BC
0	1	0	11/02/11	11/04/11		
1	2	2	10/15/11	02/25/12	2	
2	0	3	11/02/11	04/04/12	1	
3	3	0	11/04/11	11/28/11	2	
9	5	0	11/28/11	02/03/12		
10	4	0	11/28/11	04/04/12		

Figure 179: Part of the Loan table

The *Loan* table consists of foreign keys for the item being loaned (Media_ID) and the corresponding reader (Reader_ID) as well as a loan date (Loan_Date). In the table, therefore, we need to enter at the time of the loan two numeric values (Media number and Reader number) and a date. The primary key is automatically entered in the ID field. Whether the reader actually corresponds to the number is not apparent unless a second table for the readers is open at the same time. Whether the item was loaned out with the correct number is also not apparent. Here the loan must rely on the label on the item or on another open table.

All this is much easier to accomplish using forms. Here the users and the media can be looked up using list box controls. In forms, the names of user and item are visible and their numeric identifiers are hidden. In addition, a form can be so designed that a user can be selected first, then a loan date, and each set of media are assigned this one date by number. Elsewhere these numbers can be made visible with the exactly corresponding media descriptions.

Direct entry into tables is useful only for databases with simple tables. As soon as you have relationships between tables, a specially designed form is better. In forms, these relationships can be better handled by using sub-forms or list fields.

Base Guide 7.3

Chapter 4
Forms

Forms make data entry easier

Forms are used when direct entry into a table is inconvenient, to pick up errors in data entry promptly, or when too many tables make direct management of data impossible.

Note

A form in Base is a structure invisible to the user. It serves within Base to allow contact with the database. What is visible to the user is the set of controls, which serve for the entry or display of text, numbers, etc. These controls are divided by the GUI into various types.

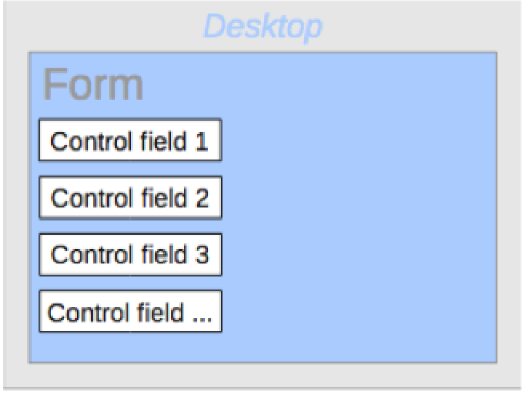


Figure 180: Schematic representation of a simple form

The term *Form* has two meanings. It can stand for the whole content of the input window which is used to manage data for one or more tables. Such a window can hold one or more main forms, and each of these can contain subforms. The word Form is also used for these partial areas. It should be clear from the context which meaning is intended so that misunderstandings should be avoided.

Creating forms

The simplest way to create a form is to use the Form Wizard. Use of it to create a form is described in Chapter 8, Getting Started with Base, in the *Getting Started Guide*. That chapter also explains how you can further modify the form after using the Wizard.

This guide describes the creation of a form without using the Wizard. It also describes the properties of the various types of controls in a form.

A simple form

We start by using the task **Create Form in Design View** in the Forms area of the main Base window.

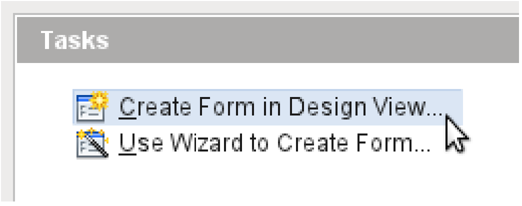


Figure 181: Creating a new form in Design View

This calls up the Form Editor and the form is shown in the Design View window (Figure 182).

The Form Controls toolbar is docked on the left side. The Form Design toolbar (Figure 184) is docked at the bottom. If these toolbars do not appear automatically, use **View > Toolbars** to display them. Without these toolbars, it is not possible to create a form.

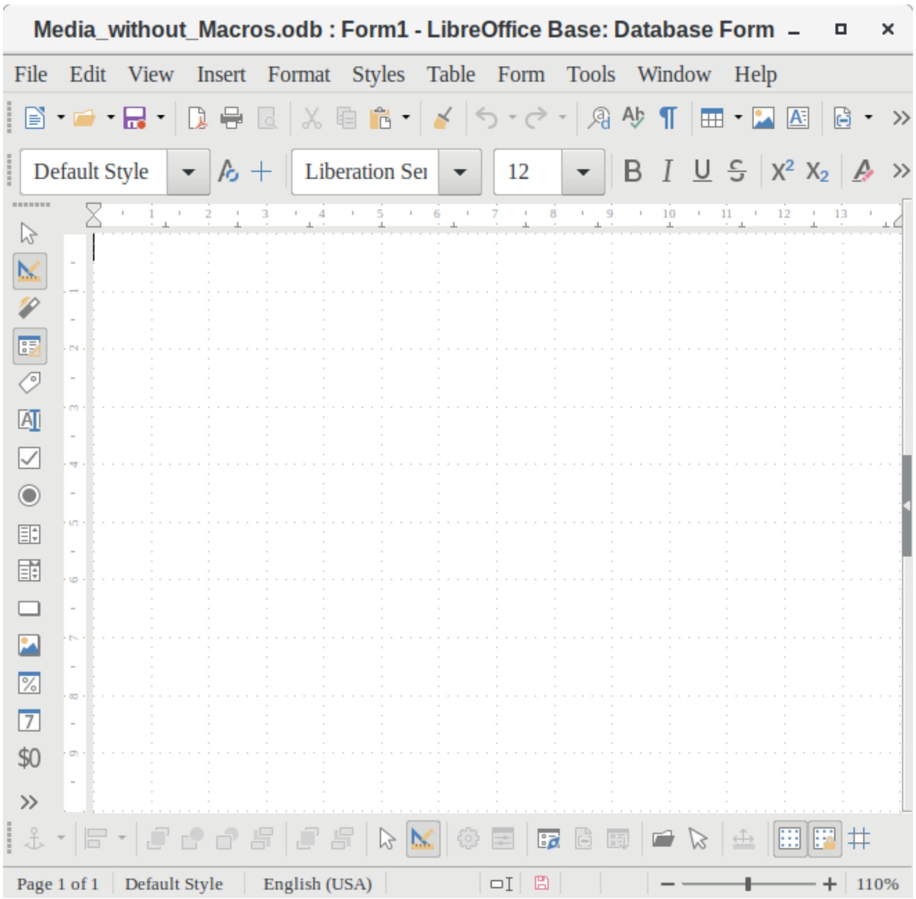

Media_without_Macros.odb : Form1 - LibreOffice Base: Database Form
File Edit View Insert Format Styles Table Form Tools Window Help
Default Style
Liberation Ser
12
Page 1 of 1
Default Style
English (USA)
110%

Figure 182: The Database Form window supports the design of forms in Design View

The blank area shows a grid of dots. This grid helps you to position the controls accurately, especially in relation to each other. The symbols at the right end of the Form Design toolbar show that the grid is visible and active. The last three symbols should be visible and active. If not all are, click the ones that are not.

Tip

The form editor work actually as a Writer document. All Writer resources are available for editing the form, including styles, direct formatting, spell checking, objects and more.

Toolbars for form design

A form is created on the empty page. This can be done in two ways:

- Use the Form Navigator to set up a form, or
- Design the form controls and set up the form by using the context menu.

Setting up a form with the Form Navigator

To display the Form Navigator, click the Form Navigator button (shown in Figure 184). A window appears (Figure 183); it shows only one folder, labeled Forms. This is the highest level of the area that we are editing. Several forms can be accommodated here.

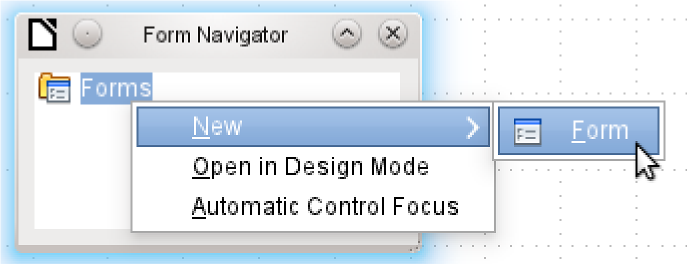


Figure 183: Using the Form Navigator to create a new form

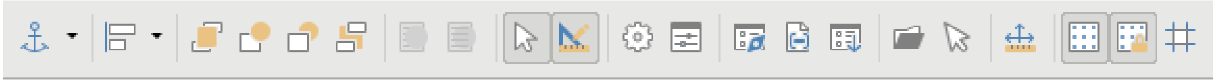


Figure 184: Form Design toolbar

In the Form Navigator (Figure 183), right-click on Forms to open a context menu. Choose **New > Form** to create a new form. The other choices in the context menu (Open in Design Mode and Automatic Control Focus) correspond to buttons in Figure 186; we will discuss them later.

Tip

To launch a form with the cursor in the first field, use the option **Automatic control focus**. What counts as the first element is determined by the form's activation sequence.

The form carries the default name *Form*. You can change this name immediately or later. It has no significance unless you need to access some part of the form using macros. The only thing you need to ensure is that two elements with the same name do not occur on the same level in the folder tree.

The context menu of the form (Figure 185) provides the way to create form properties.

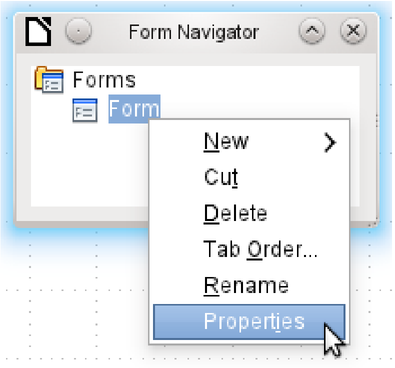


Figure 185: Accessing the Form Properties dialog

Creating a form using a form field

The Form Controls toolbar (Figure 186) contain the buttons with which the controls (label plus field) can be created. Not all of the buttons are directly visible along the left side of the form. Click the bottom button (») to see the rest of the visible buttons. To see all of the buttons both visible or not, right-click any part of the toolbar and select **Visible Buttons**. The visible buttons are checked; the unchecked ones are not.

This permits selecting the buttons that the user wants to use regularly while removing the undesirables. As an example, the Table Control button is not visible, but can be made so by first locating it in the list of all of the button and then checking it.

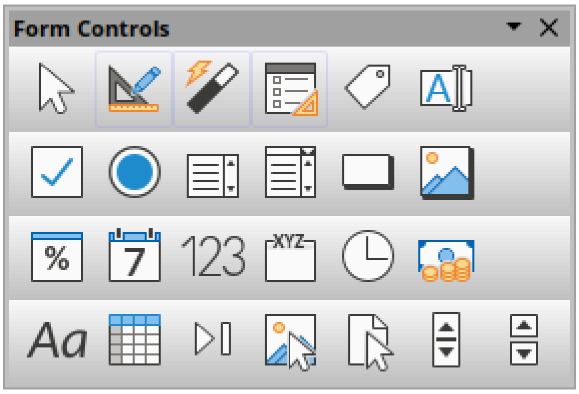


Figure 186: Form Controls toolbar

When you select a form control, you automatically create a form. For example, suppose you choose a text field: the cursor changes shape and a rectangular shape may be drawn on the white surface of the form. Then, on the stippled surface of the form, a text field appears.

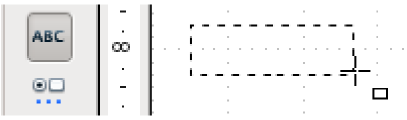


Figure 187: Inserting a text box

Now you can create the form by right-clicking and using the context menu for the control (Figure 188).

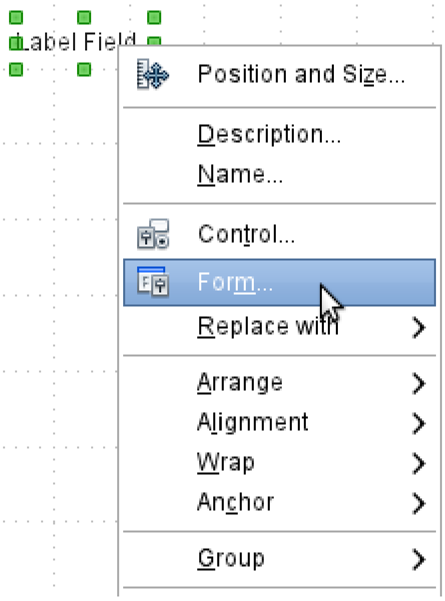


Figure 188: Accessing the Form Properties dialog from a form control's context menu

Select the **Form** menu option (highlighted in the illustration) to set properties for the form you have just created. The form has the default name Form.

External forms

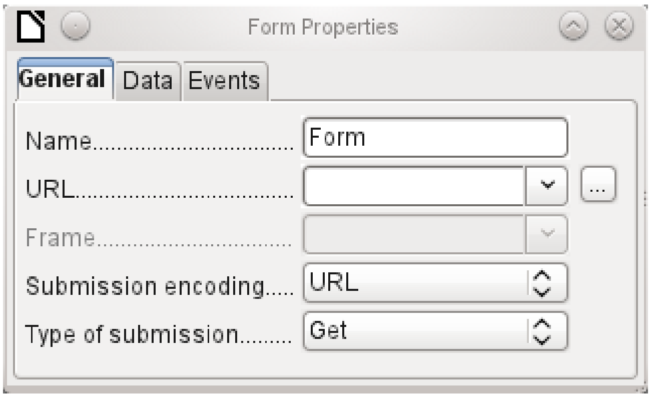
As well as forms that are created within Base, there is also a possibility to create forms in Writer or Calc. These are described in Chapter 7, Linking to Databases.

Form properties

When the form properties are called up using the context menu in the Form Navigator or the context menu of a form control, a Form Properties window appears. It has three tabs: *General*, *Data* and *Events*.

General tab

Here you can change the Name of the form. In addition there are design possibilities that have no significance inside Base. They show only the more general possibilities for design using a form editor: when you create a Web form you will need to use them.



Name	Form
URL	
Frame	
Submission encoding	URL
Type of submission	Get

Figure 189: Form Properties dialog (General tab)

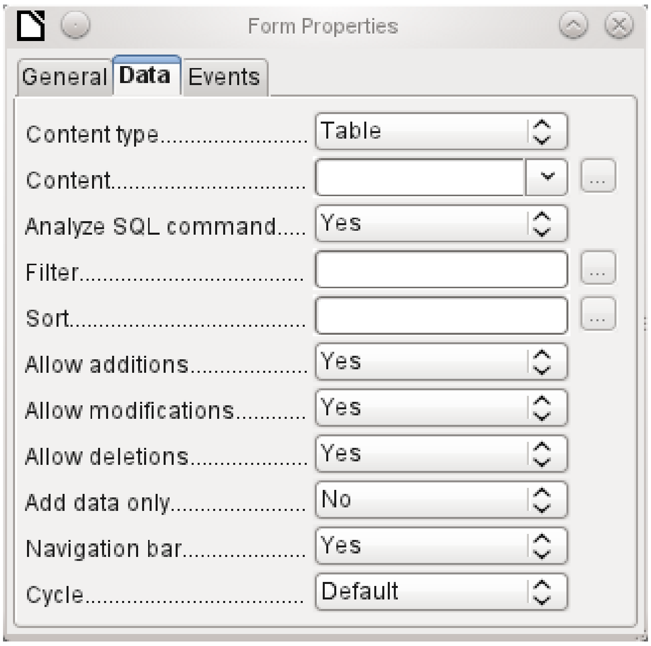
URL: Destination for the data.

Frame: Section of the destination website to be addressed where necessary.

Submission encoding: in addition to the normal character encoding for transmission to the URL, you can specify here text encoding and multipart coding (for example, for transfer of data).

Type of submission: GET (visible via the URL attached to the filename; you can see this often in the web if you use a search engine) or POST (not visible; suitable for large data volumes).

Data tab



Content type	Table
Content	
Analyze SQL command	Yes
Filter	
Sort	
Allow additions	Yes
Allow modifications	Yes
Allow deletions	Yes
Add data only	No
Navigation bar	Yes
Cycle	Default

Figure 190: Form Properties dialog (Data tab)

For creating internal forms in Base, this is the most important tab. Here you can set the following initial properties for the form:

Content type: Choose between **Table**, **Query**, and **SQL** command. While Table can always be used for data entry into a form, this is not always the case for Query (for more information, see Chapter 5, Queries) or direct entry of a SQL command. Here we are dealing with a query that is not visible in Base's query container but has in principle the same structure.

Content: According to whether Table or Query was chosen above, all available tables and queries are listed. If a SQL command is to be created, you can invoke the Query Editor by using the ellipsis (…) to the right of the Content field.

Analyze SQL command: If the analysis of SQL commands should not be permitted (because, for example, you are using code that the GUI cannot show correctly), you should choose **No** here. However this will prevent the form accessing the underlying data using a filter or a sort.

Filter: Here you can set a filter. To get help with this, click the button to the right of the field. See also Chapter 3, Tables.

Sort: Here you can set up a Sort for your data. To get help, click the button to the right of the field. See also Chapter 3, Tables.

Allow additions: Should the entry of new data be allowed? By default this is set to **Yes**.

Allow modifications: Should editing of the data be allowed? By default also **Yes**.

Allow deletions: The deletion of data is also allowed by default.

Add data only: If you choose this option and enter **No** for the other selections, an empty form will always be displayed. There will be no access to existing data, which can neither be edited nor viewed. However, If **Allow additions** and add data only are selected, data can be added to a field. But once the data has been saved, it is no longer visible and the data has been written to the tables.

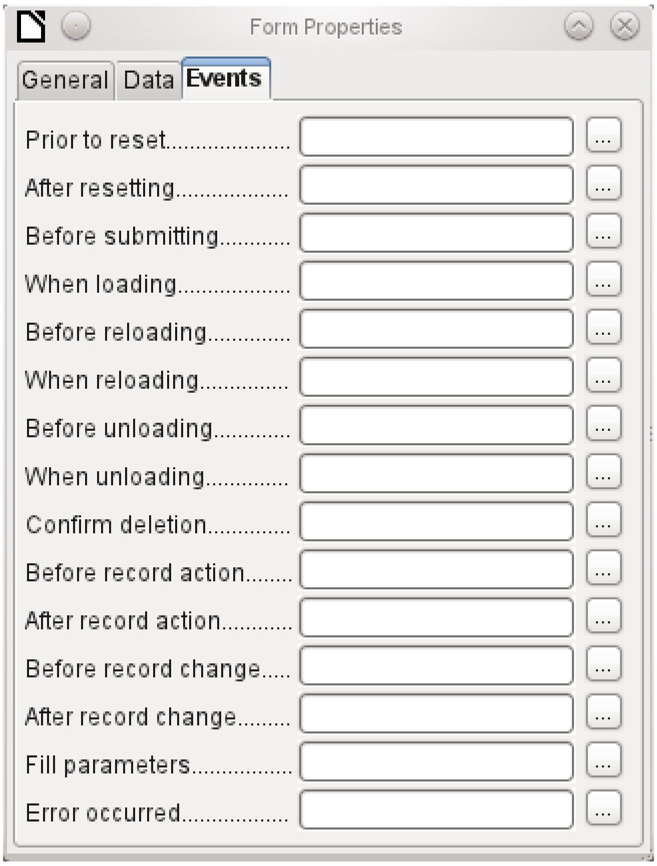
Tip

This property can be useful when there is data is to be added to the database but the person doing it is not allowed to only enter it but not to modify it in any way. Modification is to be left for another person to do with a different form which allows doing this.

Navigation bar: The appearance of the Navigation Bar at the bottom of the screen can be switched on or off. There is also a possibility, when you have a subform, always to show the Navigation Bar for the main form, so that activation of this toolbar affects the main form only. This setting for the Navigation Bar is not relevant to the internal navigation toolbar that can be added as a form control if required.

Cycle: The Default option for Base databases is that after entry into the last field in a form, the Tab key takes you to the first field of the next record – that is, a new record will be created. For databases, this has the same effect as *All records*. By contrast, if you choose *Active record*, the cursor will move only within the record; when it reaches the last field, it will jump back to the first field in that record. *Current page* refers particularly to HTML Forms. The cursor jumps from the end of a form to the next form on that page further down.

Events tab



Prior to reset....................

After resetting..................

Before submitting...........

When loading..................

Before reloading.............

When reloading..............

Before unloading............

When unloading.............

Confirm deletion.............

Before record action........

After record action...........

Before record change.....

After record change.........

Fill parameters................

Error occurred.................

Figure 191: Form Properties dialog (Events tab)

Events can trigger macros. A click on the button on the right (…) allows macros to be linked to the event.

Reset: The form is emptied of all new entries that have not yet been saved.

Before submitting: Before the form data are sent. This is only meaningful for Web forms.

When loading: Only when opening the form, not when loading a new record into the form.

Reloading: This takes place when the content of the form is refreshed, for example by using a button on the Navigation Bar.

Unloading: This option seems not to function. It would be expected to refer to the closing of the form.

Record action: This includes, for example, storage using a button. In tests, this action regularly duplicates itself; macros run twice in succession.

This is because here different functions ("implementations") are being carried out. Both have names: `org.openoffice.comp.svx.FormController` and `com.sun.star.comp.forms.ODatabaseForm`. If inside the macro that uses `oForm.ImplementationName`, the corresponding name is queried, the macro can be limited to one run.

Record change: The opening of a form counts as a record change. Whenever one record changes to another within the form, this action likewise occurs twice. Macros are therefore run twice in succession. Here too we can distinguish between the causes of this result.

Fill parameters: This macro will be run if a parameter query is to be invoked in a subform, but for some reason the parameter is not correctly transmitted from the main form. If this event is not caught, a parameter query will follow the loading of the form.

Error occurred: This event could not be reconstructed.

Properties of controls

Once a form has been created, it can be filled with visible controls. Some controls allow the content of the database to be displayed or data to be entered into the database. Other controls are used exclusively for navigation, for searching, and for carrying out commands (interaction). Some controls serve for additional graphical reworking of the form.

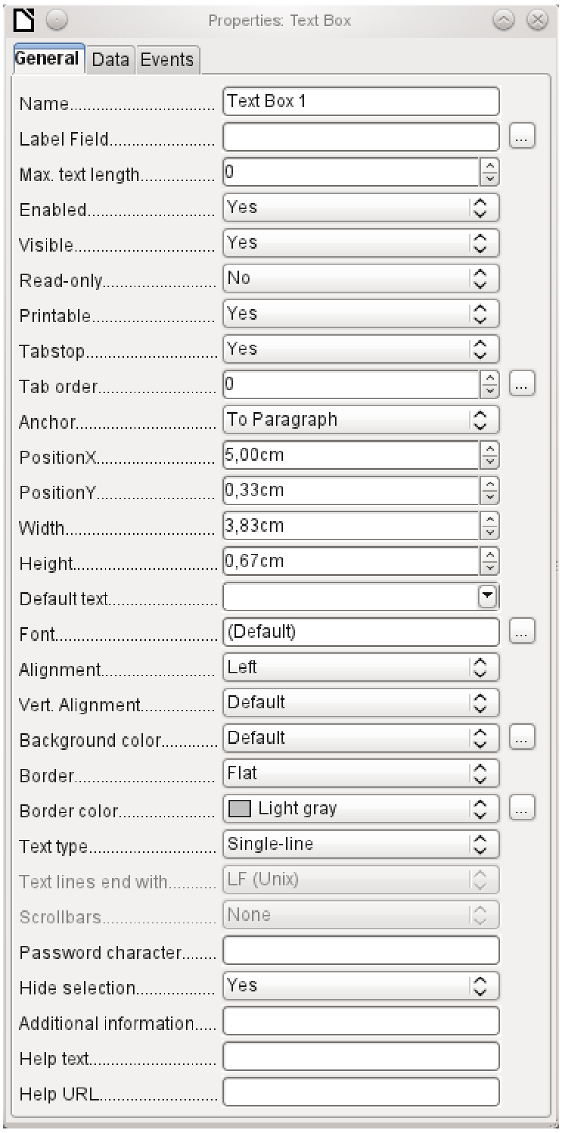
Table 5: Form controls

Data entry and Data display	
Control	***Use***
Text field	Text entry
Numeric field	Entering numbers
Date field	Entering dates
Time field	Entering times
Currency field	Numeric entry, preformated for currency
Formatted field	Display and entry with additional formatting, for example using measurement units
List box	Choosing between several different possibilities, also for transfer into the database of values other than those displayed
Combo box	Similar to a list field, but with only the displayed value transferred, or you can enter new values by hand
Check box	Yes/No Field
Options button	Radio button; allows you to choose from a small number of possibilities
Image control	Display of images from a database and entry of images into a database via a path selection
Pattern field	Entry into a preset mask; limits the entry possibilities to specific character combinations
Table control	Universal entry module, which can display a whole table. Integrated into this control are many of the above controls
Design	
Control	***Use***
Label field	Heading for the form, description of other controls
Group box	A frame around, for example, a set of option buttons
Interaction	
Control	***Use***
Button	Button with label
Image button	Like a button, but with an additional image (graphic) displayed on it
Navigation bar	Toolbar very similar to the one at the bottom edge of the screen

Interaction	
Control	***Use***
File selector	For selecting files, for example to upload in an HTML form—not further described
Spin box	Can only be used with a macro—not further described
Scrollbar	Can only be used with a macro—not further described
Hidden control	Here a value can be stored using macros and then read out again,

Default settings for many controls

As with forms, control properties are grouped into three categories: *General*, *Data*, and *Events*. General comprises everything that is visible to the user. The Data category specifies the binding to a field in the database. The Events category controls actions, which can be bound to some macro. In a database without macros, this category plays no role.



Name	Text Box 1
Label Field	
Max. text length	0
Enabled	Yes
Visible	Yes
Read-only	No
Printable	Yes
Tabstop	Yes
Tab order	0
Anchor	To Paragraph
PositionX	5,00cm
PositionY	0,33cm
Width	3,83cm
Height	0,67cm
Default text	
Font	(Default)
Alignment	Left
Vert. Alignment	Default
Background color	Default
Border	Flat
Border color	Light gray
Text type	Single-line
Text lines end with	LF (Unix)
Scrollbars	None
Password character	
Hide selection	Yes
Additional information	
Help text	
Help URL	

Figure 192: Properties dialog (General tab) for a text box

General tab

Name................................ Text Box 1

The name of a control must be unique within the form—used to access using macros.
[Name]

Label Field....................... ...

Does the field have a label? This groups field and label together.

A label allows the form field to be reached directly with a keyboard shortcut.
[LabelControl]

Enabled............................ Yes

Non-enabled fields cannot be used and are grayed out. Useful to control using macros. (Example: If Field 1 contains a value, Field 2 must not contain one; Field 2 is deactivated.)
[Enabled]

Visible............................... Yes

Usually Yes; invisible fields can be used as intermediate storage, for example in creating combination fields with macros. See Chapter 9, Macros.
[EnableVisible]

Read-only......................... No

Yes will exclude any modification of the value. This is useful, for example, for an automatically generated primary key.
[ReadOnly]

Printable........................... Yes

Sometimes it is useful to print a page from a form rather than a separate report. In this case, not all fields may be required to appear.
[Printable]

Tabstop............................. Yes

Within a form, the Tab key is normally used for navigation.
A field that is read-only does not need a tab stop; it can be skipped.
[Tabstop]

Tab order.......................... 0 ...

Does the field have a tab stop? Here the activation sequence within the form is specified.
[Tabindex]

Anchor............................... To Paragraph

Anchoring of graphics within a text field.

PositionX.......................... 5,00cm

Position from the top left corner relative to the left side of the form.
[PosSize.X]

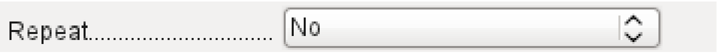
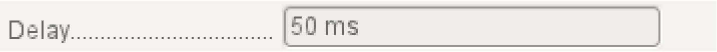
PositionY.......................... 0,33cm

Position from the top left corner relative to the top of the form.
[PosSize.Y]

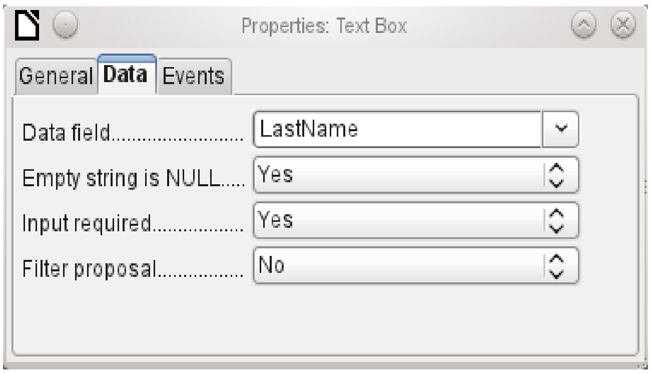
Width 3,83cm	Width of the field. [PosSize.Width]
Height 0,67cm	Height of the field. [PosSize.Height]
Font (Default)	Font, font size, and font effects can be set here. [Fontxxx]
Alignment Left	Alignment. Here text entry is left-justified. [Align]
Vert. Alignment Default	Vertical alignment: Standard | Top | Middle | Bottom. [VerticalAlign]
Background color Default	Background color of the text field. [BackgroundColor]
Border Flat	Framing: No frame | 3D-Look | Flat. [Border]
Border color Light gray	If there is a frame, its color can be set here only if *Flat* is selected as the border. [BorderColor]
Hide selection Yes	Highlighted text loses the highlight when the text field loses focus. [HideInactiveSelection]
Additional information	Used for information to be read by macros. See Chapter 9, Macros. [Tag]
Help text	Appears as a tooltip when the mouse is hovered over the text field. [HelpText]
Help URL	Points to a help file, useful mostly for HTML. Can be invoked using F1 when the focus is on the field. [HelpURL]

In addition, numeric, date fields, etc. have the following properties.

Strict format Yes	With testing enabled, only numbers and decimal points may be entered. [EnforceFormat]
Mouse wheel scroll Never	Never does not allow alterations using the mouse wheel; When selected allows such changes when the field is selected and the mouse is over the field; Always means whenever the mouse is over the field. [MouseWheelBehavior]
Spin Button No	A spin symbol is incorporated into the right side of the field. [Spin]

 Repeat.............................. No	If a spin arrow is pressed down and held, this determines if the entry in the box should be incremented beyond the next value. [Repeat]
 Delay................................. 50 ms	Determines the minimum delay after a mouse button press that triggers repetition. [RepeatDelay]

Data tab



Data field.......................... LastName

Empty string is NULL..... Yes

Input required................. Yes

Filter proposal................ No

Figure 193: Properties dialog (Data tab) for a text box

Data field: Here you create the binding with the table on which the form is based. [Model.DataField]

Empty string is NULL: Whether an empty string should be treated as (NULL) or the content simply deleted.

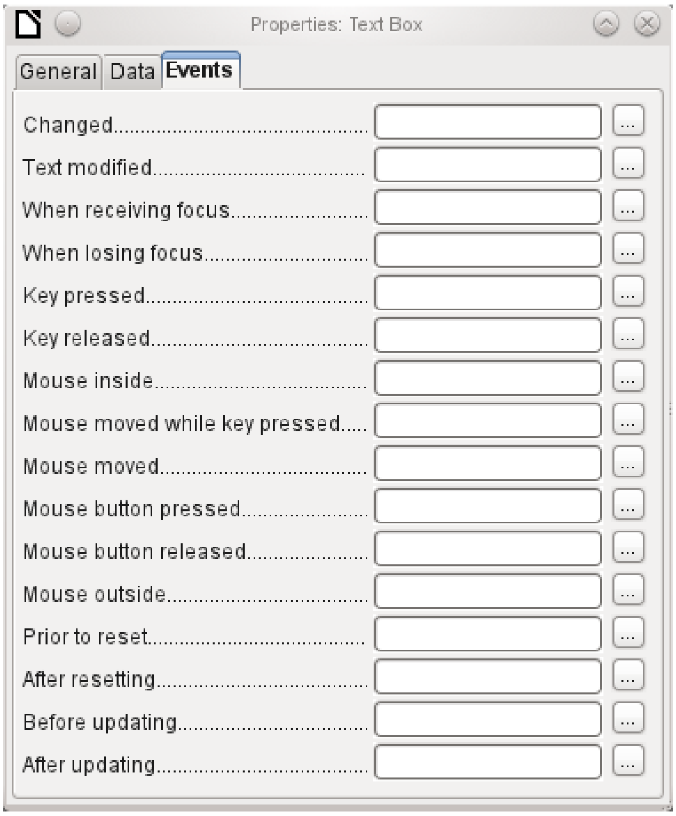
Entry required: This condition should match the one in the table. The GUI will prompt for entry if the user has not entered a value. [Model.InputRequired]

Filter proposal: When the data is to be filtered, the content of this field is temporarily stored as a suggestion. [Model.UserValueFilterProposal]

Caution

Caution – with large contents, this choice can use a lot of storage.

Events tab



Properties: Text Box

General | Data | Events

Changed ...

Text modified ...

When receiving focus ...

When losing focus ...

Key pressed ...

Key released ...

Mouse inside ...

Mouse moved while key pressed ...

Mouse moved ...

Mouse button pressed ...

Mouse button released ...

Mouse outside ...

Prior to reset ...

After resetting ...

Before updating ...

After updating ...

Figure 194: Properties dialog (Events tab) for a text box

Changed: This event takes place when a control is modified and afterwards loses the focus. The event is lost if you switch directly to another record. In these circumstances, a change is saved without being detected previously. [com.sun.star.lang.EventObject]

Text modified: Refers to the content, which can in fact be text, numeric, or whatever. Occurs after each additional character is entered. [com.sun.star.awt.TextEvent]

When receiving focus: The cursor enters the field.

Caution

Under no circumstances must the macro create a message dialog on the screen; clicking in such a dialog causes the form field to lose the focus and then recover it, triggering the macro again. A loop is created which can only be broken by using the keyboard.

When losing focus: The cursor leaves the field. This can lead to the same kind of interplay when the handling of the event causes it to recur.

Key: Refers to the keyboard. For example, a key is typed when you move through the form using the Tab key. This causes a field to receive the focus. Then the key is released.

The event is passed using the *keyCode* or *KeyChar* of the released key (letter, number, special key). [com.sun.star.awt.KeyEvent]

Mouse: Self-explanatory; these events only take place if the mouse is or was already within the field ("outside" corresponds to the javascript onMouseOut). [com.sun.star.awt.MouseEvent]

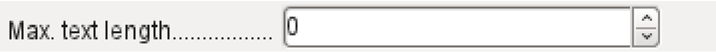
Reset: The form is emptied of all data (when creating a new record) or set back to its original state (when editing an existing record). For a form field, this event is triggered only when data entry is undone using the button in the navigation bar. [com.sun.star.lang.EventObject] When a form is first loaded, the two events *Prior to reset* and *After resetting* occur in succession, before the form is available for input.

Updating: If the event is bound to a form control, update takes place when the focus is lost and jumps to another form control, after altering the content of the field. Changes in the form are accepted and displayed. When a form is closed, the two events *Before updating* and *After updating* occur in succession. [com.sun.star.lang.EventObject]

Text field

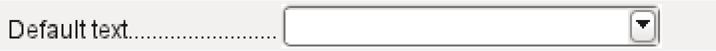
As well as the properties set out in section "Default settings for many controls" above, text fields can have the following additional properties:

General tab



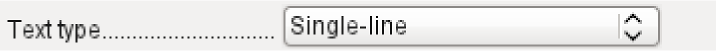
Max. text length 0

When this value is 0, entry is not permitted. Usually the length of the database field to which the text field corresponds is used here.
[MaxTextLen]



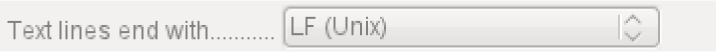
Default text

Should default text be put into an empty field? This text must be deleted if any other entry is to be made successfully.
[DefaultText]



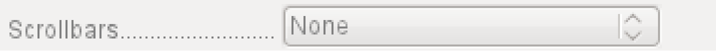
Text type Single-line

Possible types: Single-line | Multi-line | Multi-line with formatting (the last two differ in tabbing behavior and, in addition, a pattern field can not be bound to a database). The vertical alignment is not active for multi-line fields.
[MultiLine]



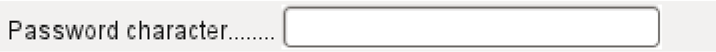
Text lines end with LF (Unix)

Unix or Windows? This mainly affects line endings. Internally Windows lines end with two control characters (CR and LF).
[LineEndFormat]



Scrollbars None

Only for multi-line fields: Horizontal | Vertical | Both.
[HScroll], [VScroll]



Password character

Active only for single-line fields. Changes characters to see only points.
[EchoChar]

Data tab

Nothing of significance.

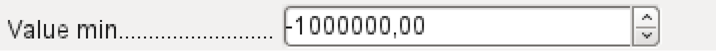
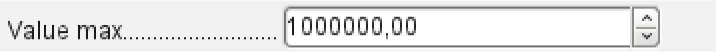
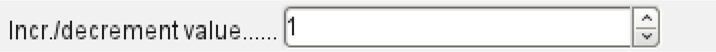
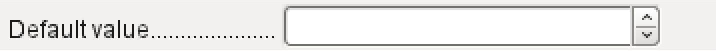
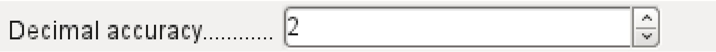
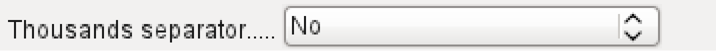
Events tab

Nothing of significance.

Numeric field

In addition to properties already described, the following properties exist:

General tab

 Value min -1000000,00	Minimum value for the field. Should agree with the minimum value defined in the table. [ValueMin]
 Value max 1000000,00	Maximum value. [ValueMax]
 Incr./decrement value 1	Scrolling increment when using the mouse wheel or within a spin box. [ValueStep]
 Default value	Value displayed when a new record is being created. [DefaultValue]
 Decimal accuracy 2	Number of decimal places, set to 0 for integers. [DecimalAccuracy]
 Thousands separator No	Separator for thousands, usually a comma. [ShowThousandsSeparator]

Data tab

There is no check on whether a field can be NULL. If there is no entry, the field will be NULL and not 0.

No filter proposal is offered.

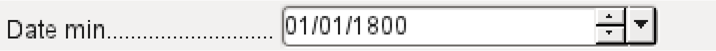
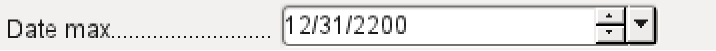
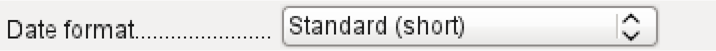
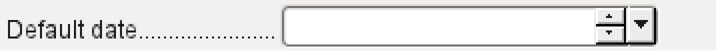
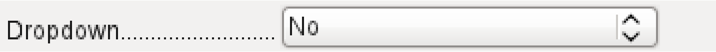
Events tab

The *Changed* event is absent. Changes must be handled using the *Text modified* event (the word *text* is not to be taken literally here).

Date field

As well as the properties described in section "Default settings for many controls" above, the following are to be noted.

General tab

 Date min 01/01/1800	Minimum value for the field, selectable using a drop-down calendar. [DateMin]
 Date max 12/31/2200	Maximum value. [DateMax]
 Date format Standard (short)	Short form as 10.02.12 or various forms using `'/'` instead of `'.'` or `'-'` in the American style. [DateFormat]
 Default date	Here you can enter a literal date but not (yet) the current date (Today) at the time the form is opened. [DefaultDate]
 Dropdown No	A month calendar for selecting dates can be included. [DropDown]

Data tab

There is no check on whether a field can be NULL. If there is no entry, the field will be NULL and not 0. No filter proposal is offered.

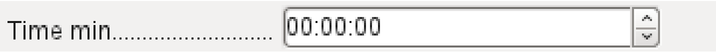
Events tab

The *Changed* event is absent. Changes must be handled using the *Text modified* event (the word *text* is not to be taken literally here).

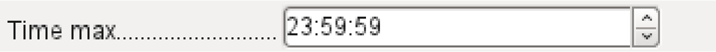
Time field

As well as the properties listed in section "Default settings for many controls" above, the following features are available.

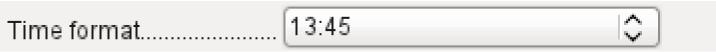
General tab


Time min 00:00:00

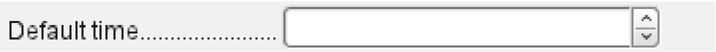
Minimum value for the field, by default set to 0.
[TimeMin]


Time max 23:59:59

Maximum value, by default set to 1 second before 24:00.
[TimeMax]


Time format 13:45

Short form without seconds, long form with seconds, and also 12-hour formats with AM and PM.
[TimeFormat]


Default time

You can set a fixed time but not (yet) the actual time of saving the form.
[DefaultTime]

Data tab

There is no check on whether a field can be NULL. If there is no entry, the field will be NULL and not 0.

No filter proposal is offered.

Events tab

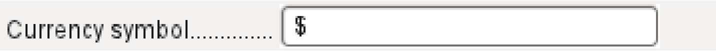
The *Changed* event is absent. Changes must be handled using the *Text modified* event (the word *text* is not to be taken literally here).

Currency field

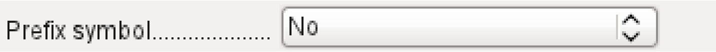
In addition to the properties already listed in section "Default settings for many controls" above, the following features are available:

General tab

Min. value, Max. value, Increment, Default value, Decimal places, and Thousands separator. correspond to the general properties listed in section "Numeric field" above. In addition to these, there is only:


Currency symbol $

The symbol is displayed but not stored in the table that underlies the form.
[CurrencySymbol]


Prefix symbol No

Should the symbol be placed before or after the number?
[PrependCurrencySymbol]

Data tab

There is no check on whether a field can be NULL. If there is no entry, the field will be NULL and not 0.

No filter proposal is offered.

Events tab

The *Changed* event is absent. Changes must be handled using the *Text modified* event (the word *text* is not to be taken literally here).

Formatted field

In addition to the properties listed in section "Default settings for many controls" above, the following features are offered:

General tab

Minimum and maximum values, and the default value, depend on the formatting. Behind the button for Formatting is a flexible field that makes most currency and numeric fields unnecessary. Unlike a simple currency field, a pattern field can show negative sums in red.

The button to the right with the three dots provides a choice of numeric formats, as you usually do in Calc. [FormatKey]

Among the numeric formats can be seen, alongside Date, Time, Currency or normal numeric format, possibilities for using fields with a measurement unit such as kg (see Figure 195). See also the general Help on numeric format codes.

A formatted field makes it possible to create and write into timestamp fields in tables using just one field. The Form Wizard uses a combination of a date and a time field for this.

If you want to enter data in the form minutes:seconds:hundredths of seconds into a timestamp field, you will need to use macros.

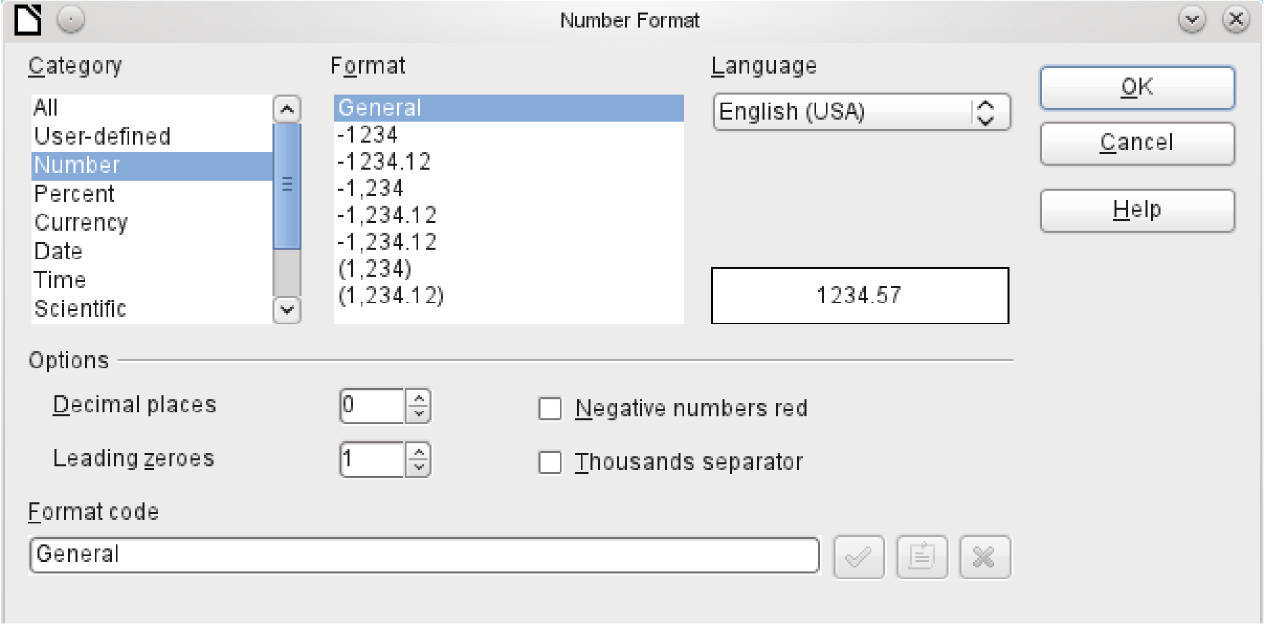


Figure 195: Number Format dialog with general number format selected

Data tab

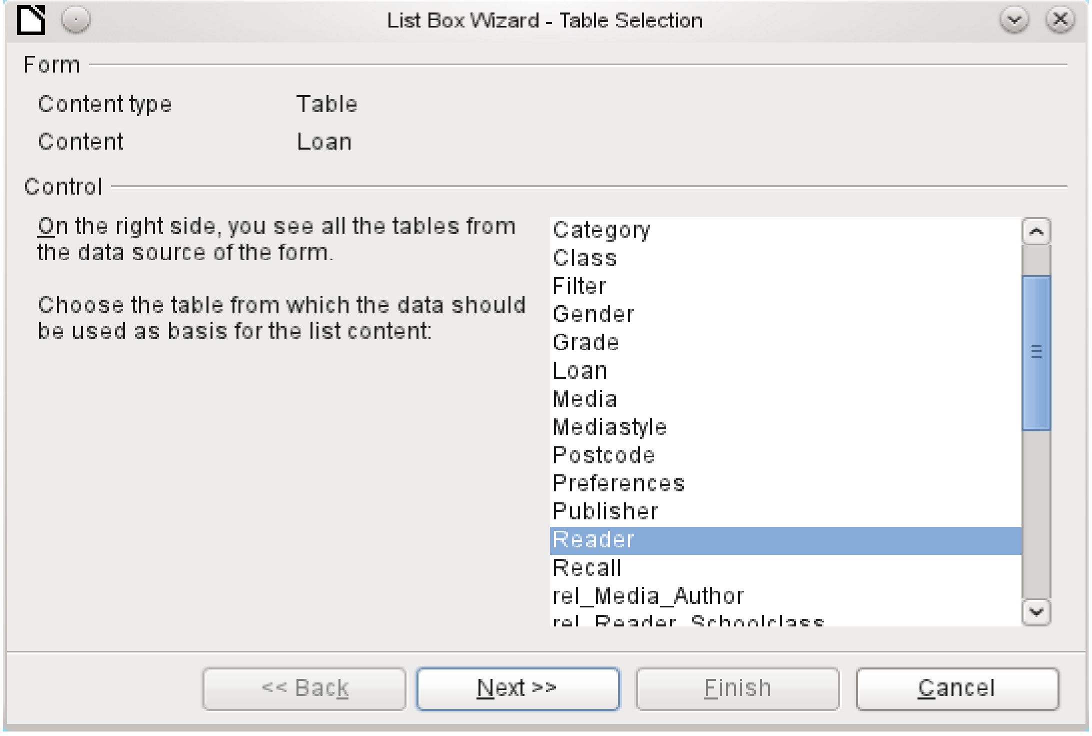
Nothing special to report.

Events tab

The *Changed* event is absent. Changes must be handled using the *Text modified* event (the word *text* is not to be taken literally here).List box

List Box

When a list box is created, the List Box Wizard appears by default. This automatic appearance can be switched off if required using the **Toggle Form Control Wizards** icon in the Form Controls toolbar (shown in Figure 186).



List Box Wizard - Table Selection

Form

Content type: Table

Content: Loan

Control

On the right side, you see all the tables from the data source of the form.

Choose the table from which the data should be used as basis for the list content:

Category, Class, Filter, Gender, Grade, Loan, Media, Mediastyle, Postcode, Preferences, Publisher, Reader (selected), Recall, rel_Media_Author, rel_Reader_Schoolclass

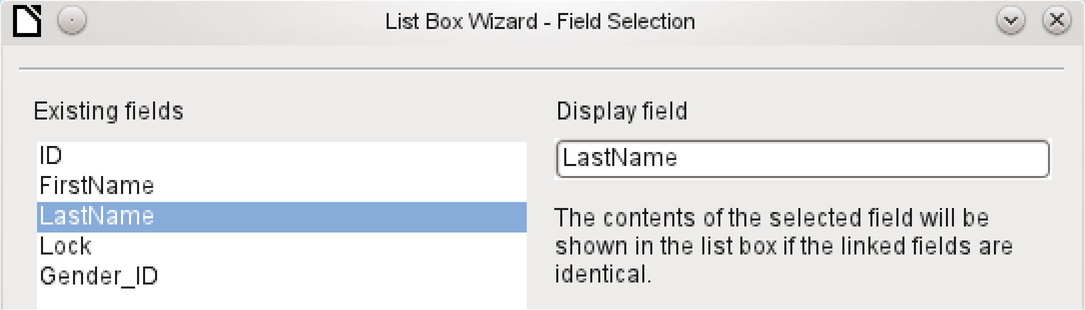
Figure 196: List Box Wizard, table selection

Wizard

The form is already defined. It is bound to a table named Loans. A list box shows the user different data from what is actually transmitted into the table. This data usually comes from another table in the database, and not from the table to which the form is bound.

The Loans table is supposed to show which Reader has borrowed which Media. However this table does not store the name of the reader but the corresponding primary key from the Reader table. It is therefore the Reader table that forms the basis for the list box.

The LastName field from the Reader table should be visible in the list box. This serves as the Display field.



List Box Wizard - Field Selection

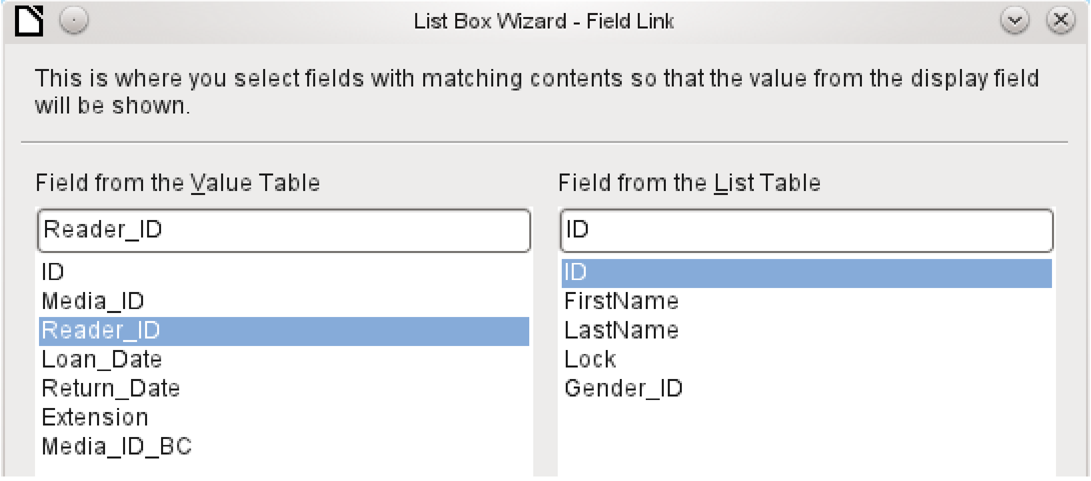
Existing fields: ID, FirstName, LastName (selected), Lock, Gender_ID

Display field: LastName

The contents of the selected field will be shown in the list box if the linked fields are identical.

Figure 197: List Box Wizard, field selection

The Reader_ID field occurs in the Loan table which underlies the form. This table is described here as the Value table. The primary key ID from the Reader table must be bound to this field. The Reader table is described here as the List table.



List Box Wizard - Field Link

This is where you select fields with matching contents so that the value from the display field will be shown.

Field from the Value Table: Reader_ID

- ID
- Media_ID
- Reader_ID
- Loan_Date
- Return_Date
- Extension
- Media_ID_BC

Field from the List Table: ID

- ID
- FirstName
- LastName
- Lock
- Gender_ID

Figure 198: List Box Wizard, linking fields in value and list tables

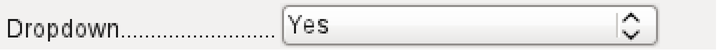
The list box has now been created complete with data and default configuration and is fully functional.

In addition to the properties listed in section "Default settings for many controls" above, the following features are available.

General tab

List entries

The list entries have already been set using the Wizard. Here you could add further entries that are not from any table in the database. List entries here mean the visible entries, not those that the form will transmit to the table.
[StringItemList]



Dropdown: Yes

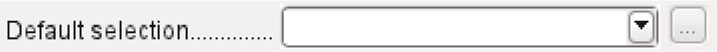
If the field is not specified as drop-down, scroll arrows will appear on the right side of the list box when the form is loaded. The list field then automatically becomes a multi-line field, in which the actual value selected is highlighted.
[Dropdown]

Line count: 20

If the field is drop-down, this property gives the maximum visible number of lines. If the content extends over more lines, a scrollbar appears when the list drops down.
[LineCount]

Multiselection: No

Can more than one value be selected? In the above example, this is not possible since a foreign key is being stored. Usually this function is not used for databases, since each field should only contain one value. If necessary, macros can help in the interpretation of multiple entries in the list field.
[MultiSelection]
[MultiSelectionSimpleMode]

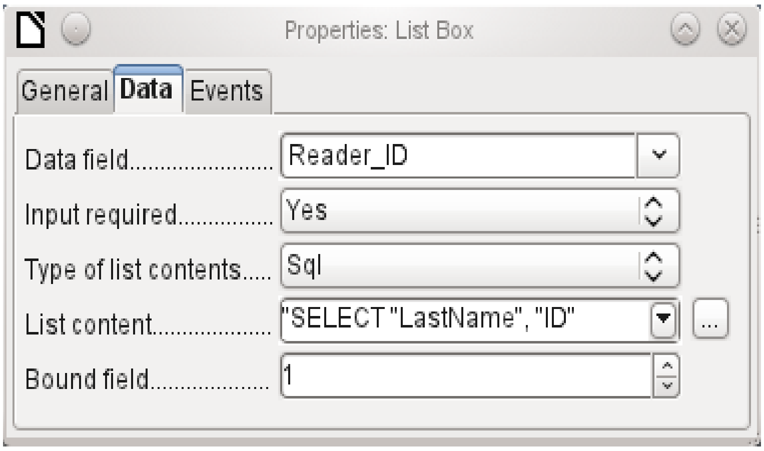


Default selection............. ...

As the deactivated button makes clear, a default selection makes little sense in the context of a binding with a database table, as created by the List Field Wizard. It could well be the case that the record corresponding to the default selection in the example table Readers is no longer present. [DefaultSelection]

Data tab

In addition to the usual data properties, Data field and Input required, there are significant properties which affect the binding between the displayed data and the data to be entered into the table that underlies the form.



Data field....................... Reader_ID

Input required............... Yes

Type of list contents..... Sql

List content.................. "SELECT "LastName", "ID"

Bound field.................. 1

Figure 199: Properties dialog (Data tab) for a list box

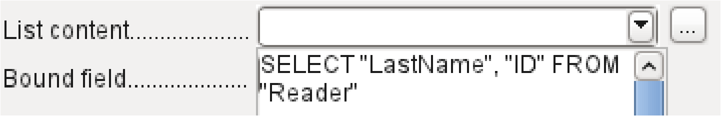
Type of list contents: Valuelist | Table | Query | SQL | SQL [Native] | Tablefields [ListSourceType]

List contents Valuelist: If list entries have been created under *General*, the corresponding values to be stored are entered here. The list contents are loaded with individual items separated by *Shift+Enter*. The List content field then shows them as *"Value1";"Value2";"Value3"* … The Bound Field property is inactive.

List contents Table: Here one of the database tables can be selected. However this is seldom possible as it requires the content of the table to be so structured that the first table field contains the values to be displayed in the list field, and one of the following fields contains the primary key which the table underlying the form uses as a foreign key. The position of this field within the table is specified in Bound Field, where the *Numbering begins with 0 for the first field of the database table*. But this 0 is reserved for the displayed value, in the above example the Surname, while the 1 refers to the ID field.

List contents Query: Here a query is first created separately and stored. The creation of such queries is described in Chapter 5, Queries. Using the query, it is possible to move the ID field from the first position in the underlying table to the second position, here represented by the bound field 1.

List contents SQL: The List Box Wizard fills this field. The query constructed by the Wizard looks like this:



List content..................

Bound field..................

SELECT "LastName", "ID" FROM "Reader"

Figure 200: Typical SQL code generated by the List Box Wizard

The query is the simplest possible. The Surname field occurs at position 0, the ID field at position 1. Both are read from the Reader table. As the bound field is Field 1, this SQL formula works. Here should be added *ORDER BY "LastName" ASC.* So you don't need to scroll too long through the list to find somebody. An additional problem might be that LastName could be the same for more than one reader. So FirstName must be added in the view of the list box. When there are readers with the same LastName and the same FirstName, the primary key ID must also be shown. See Chapter 5, Queries, for information on how this works.

List contents SQL [Native]: The SQL formula is entered directly, not using the Wizard. Base does not evaluate the query. This is suitable when the query contains functions that might perhaps not be understood by the Base GUI. In this case the query is not checked for errors. More about *direct SQL Mode* can be found in Chapter 5, Queries.

List contents tablefields: Here Field names from a table are listed, not their content. For the Reader table, the List contents would be ID, Given name, Surname, Lock, Gender_ID.

Note

If you want a time field that can handle time in milliseconds, you will need a timestamp field, as described in the section on “Time field”. The representation of milliseconds is incompatible with the way characters are assembled in list boxes. To get around this, you must convert the timestamp into text:

```
SELECT REPLACE(LEFT(RIGHT(CONVERT(”Required_service(??)”.
”Time”, VARCHAR),15),8),’.’,’,’) AS “Listcontent”, “ID” FROM
“Required_service”
```

This will give a display in minutes:seconds:hundredths.

Bound field: List fields show content that is not necessarily identical to what will be stored in the form. Usually a name or something similar is displayed and the corresponding primary key becomes the stored value of this field.

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

The ID field is stored in the underlying table as a foreign key, The field count in databases begins at zero so the field bound to it in the form has the number 1.

If instead you select “0”, then the content of the Name field is saved inside the form. In that case you can remove the ID field from the query.

It is even possible to choose the position “–1”. Then it is not the content of the query but the position of the entry that is stored in the list. The first record then has the position 1.

The above query yields the following result:

Table 6: Result of query selecting data from table

Name	***ID***
Anneliese	2
Dorothea	3
Sieglinde	1

In list fields, only the name can be selected. The list field is set to the name “Dorothea”. The following saved data are possible for this field:

Bound field=1 means “3” is saved, the content of the ID field.

Bound field=0 means “Dorothea” is saved, the content of the Name field.

Bound field=–1 means “2” is saved, because “Dorothea” comes second in the list.

Note

The change of the bound field to “0” or “–1” was introduced with LO version 4.1. Previously, only the selection of values >= 1 was possible.

Events tab

In addition to the standard events, the following events are available:

Execute action: If a value is chosen by the keyboard or the mouse, the list box executes this action.

Item status changed: This could be the change of the displayed content of a list box through the use of the drop-down button. It could also be a click on the drop-down button of the field.

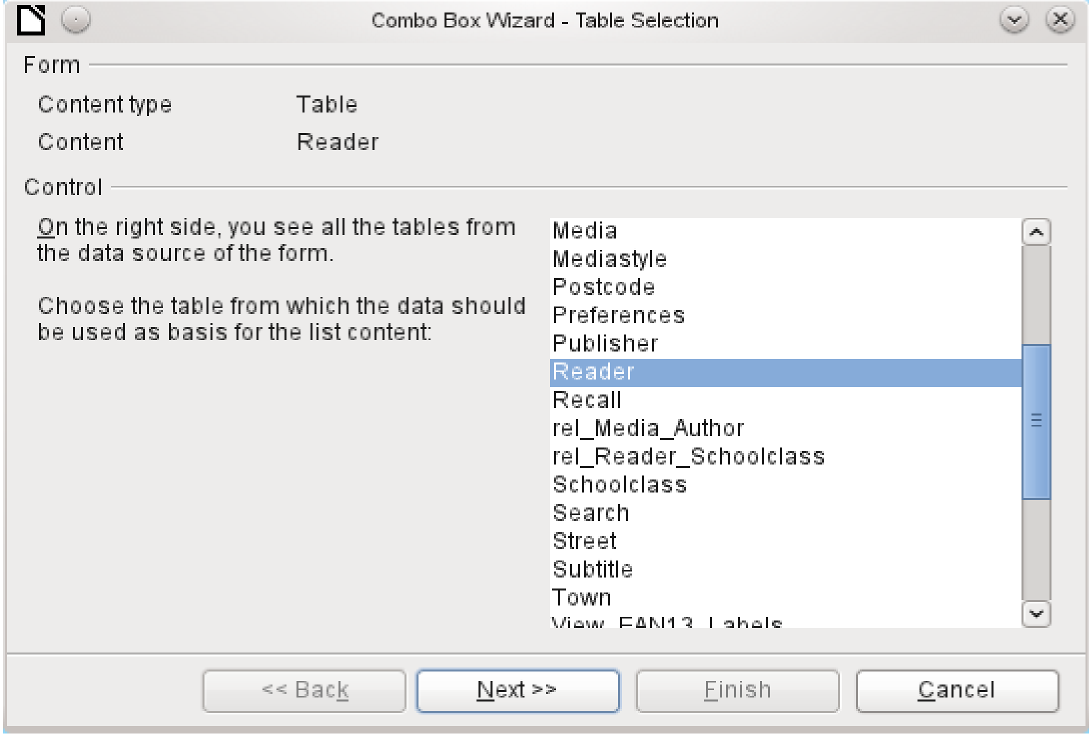
Error occurred: This event does not occur for list boxes.

Combo box

As soon as a combo box is created, a Wizard appears by default, just as with a list box. This automatic behavior can be switched off if necessary using the **Wizards On/Off** button.

Combo boxes write the selected text directly into the table underlying the form. Therefore the following example shows both the table linked to the form and the one selected for the control is the Reader table.

Wizard



Combo Box Wizard - Table Selection

Form

Content type Table

Content Reader

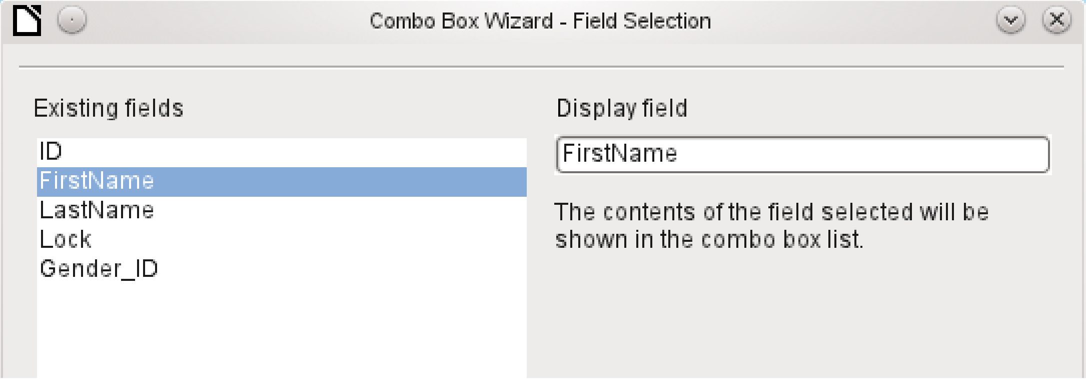
Control

On the right side, you see all the tables from the data source of the form.

Choose the table from which the data should be used as basis for the list content:

Figure 201: Combo Box Wizard, table selection

Again the form is predefined, this time with the Reader table. As the data to be displayed in the combo box is also to be stored in this table, the source selected for the data for the list is likewise the Reader table.



Combo Box Wizard - Field Selection

Existing fields

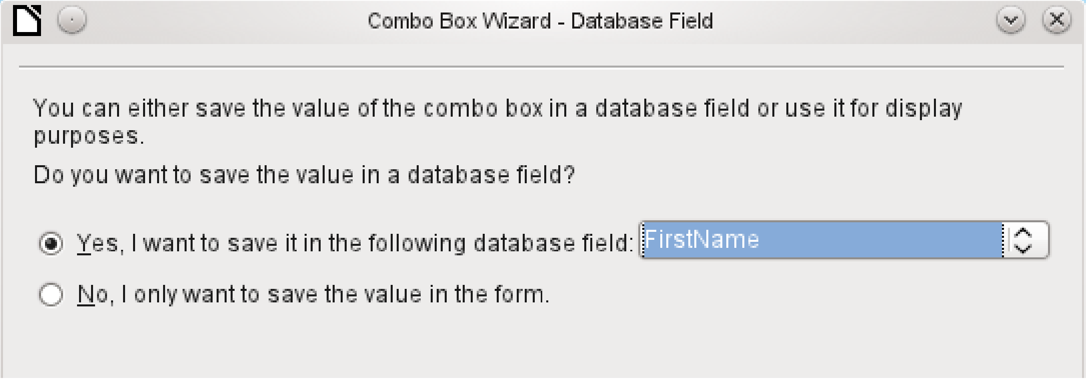
- ID
- FirstName
- LastName
- Lock
- Gender_ID

Display field: FirstName

The contents of the field selected will be shown in the combo box list.

Figure 202: Combo Box Wizard, field selection

In the Reader table the FirstName field occurs. This should be displayed in the combo box.



Combo Box Wizard - Database Field

You can either save the value of the combo box in a database field or use it for display purposes.

Do you want to save the value in a database field?

● Yes, I want to save it in the following database field: FirstName

○ No, I only want to save the value in the form.

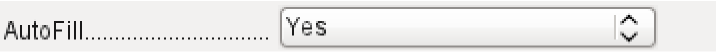
Figure 203: Combo Box Wizard, select whether to save value in a database field

In a database, there seems to be little point in not storing the value of a combo box within a field. We want to read given names from the Reader table, and also to make them available for new readers, so that new records do not need to be created for a given name that already exists in the database. The combo box shows the first name, and text input is not necessary.

If a new value does need to be entered, this can be done easily in a combo box, as the box shows exactly what is going into the underlying table for the form.

In addition to the properties shown in section "Default settings for many controls" above and described for list boxes, the following features are available.

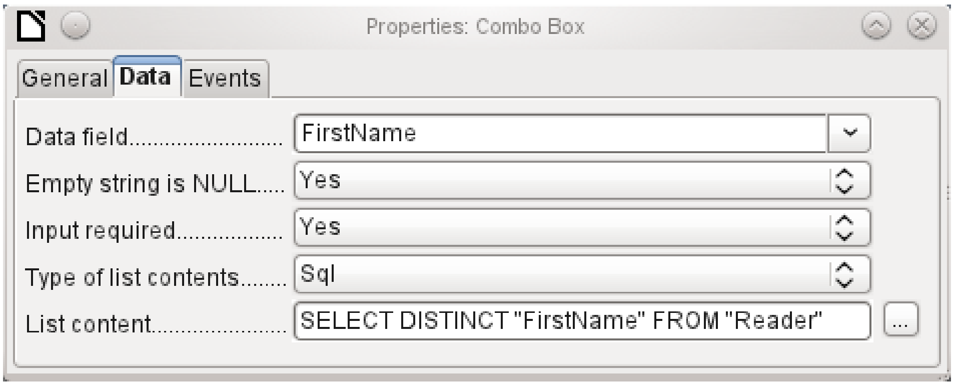
General tab



AutoFill: Yes

During entry of new values, a list of matching values (if any) is displayed for possible selection.
[AutoComplete]

Data tab



Data field: FirstName

Empty string is NULL: Yes

Input required: Yes

Type of list contents: Sql

List content: SELECT DISTINCT "FirstName" FROM "Reader"

Figure 204: Properties dialog (Data tab) for a combo box

The data fields conform to the existing default settings and the settings for a list box. The SQL command however shows a special feature:

```
SELECT DISTINCT "FirstName" FROM "Reader"
```

Adding the `DISTINCT` keyword ensures that duplicate given names are shown only once. However, creation using the Wizard once more makes it impossible for the content to be sorted.

Events tab

The events correspond to those for a list box.

Check box

The check box appears immediately as a combination of a check box field and a label for the box.

In addition to the properties described in section "Default settings for many controls" above, the following features are available.

General tab

Label: Check Box

The label for this box appears by default to the right of the box. In addition you can bind it to a separate label field. [Label]

Default status: Not Selected

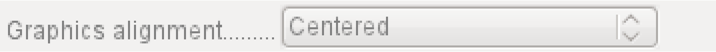
Here, dependent on the choice in the Tri-state field, up to three possibilities are available: Not selected | Selected | Not defined. Not defined corresponds to a NULL entry in the table underlying the form. [State]

Word break: No

By default, the label is not broken up. The label is truncated if the field is not big enough. [MultiLine]

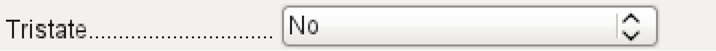
Graphics:

Here you can specify a graphic instead of or in addition to the label. You should note that these graphics are not bound into the *.odb document by default. For small graphics, it is useful to embed the graphic and not link it. [Graphic]



Graphics alignment......... Centered

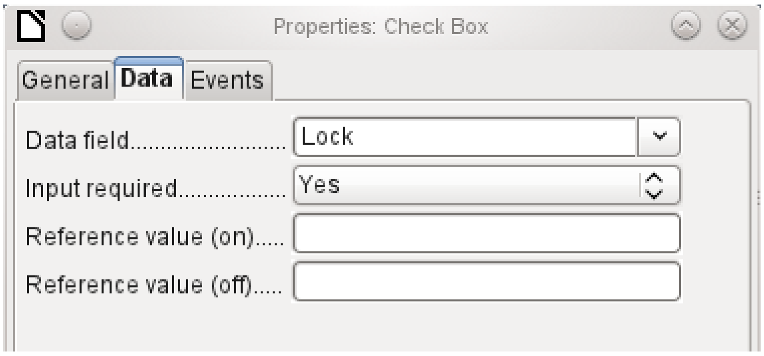
If you have chosen to use a graphic, its alignment with the label can be set here.
[ImagePosition]
(0=left | 1=centered | 2=right)



Tristate.............................. No

By default, checkboxes have only two states: Selected (Value: 1) and Not selected (Value: 0). With Tri-state, a definition of Empty field (NULL) is added.
[TriState]

Data tab



Data field......................... Lock

Input required................. Yes

Reference value (on).....

Reference value (off).....

Figure 205: Properties dialog (Data tab) for a check box

The check box can be given a reference value. However only the values of 1 (for On) or 0 (for Off) can be transferred to the underlying data field (check boxes act as fields for the choice of Yes and No).

Events tab

The fields *Changed*, *Text modified*, *Before updating*, and *After updating* are all absent.

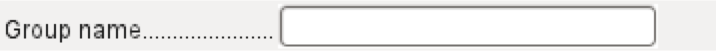
Additional fields for a check box are *Execute action* (see List box) and *Item status changed* (corresponds to *Changed*).

Option button

The option button is similar to the check box described above, except for its general properties and its external (round) form.

When several option buttons in the form are linked to the same table field, only one of the options can be selected.

General tab



Group name.....................

The option button is designed to be used in groups. One of several options can then be selected. That is why a Group name appears here, under which the options can be addressed.
[GroupName]

Data tab

See under Check box. Here, however, reference values that are entered are actually transferred to the data field.

Events tab

See under Check box.

Image control

A graphical (image) control manages the input and display of images for the database. The underlying data field must be a binary field if it is to store the image directly. It can also be a text field storing the relative path to the image. In that case you must take care that the path to images remains valid if the database is copied.

Caution

Pictures should in any case be reduced in size when stored in the database. With 3MB photographs in an internal HSQLDB database, you will very quickly get Java errors (`NullpointerException`) that make it impossible to store the records. This can lead to the loss of all the records that were input in the current session.

Entry into an image control takes place either by a double-click with the mouse to open a file selection dialog, or a right-click to choose whether an existing graphic is to be deleted or replaced.

A graphical control by default has no Tab stop.

In addition to the properties described in section "Default settings for many controls" above, the following features are available.

General tab

Graphics

The graphic selected here is only shown inside the control, while the form is being edited. It has no significance for later input.
[Graphic]

Scale: Keep Ratio

No: The image will not be fitted to the field. If it is too big, the field will show only apart of the image. The image is not distorted.
Keep ratio: The image is fitted to the control but not distorted (aspect ratio preserved).
Autom. Size: The image is fitted to the control and may be shown in a distorted form.
[ScaleImage] [ScaleMode]

Data tab

Nothing further to report.

Events tab

The events *Changed*, *Text modified*, *Before updating*, and *After updating* are missing.

Pattern field

An input mask is used to control input into the field. Characters are pre-specified for particular positions, determining the properties of entered characters. The preset characters are stored along with the entered ones.

In addition to the properties described in section "Default settings for many controls" above, the following features are available.

General tab

Edit mask

This determines what characters can be entered.
[EditMask]

Literal mask

This is what the form user sees.
[LiteralMask]

The length of the edit mask determines how many characters may be entered. If the user’s entry does not match the mask, the entry is rejected on leaving the control. The following characters are available for defining the edit mask.

Table 7: Characters available for defining an edit mask

Character	***Meaning***
L	A text constant. This position cannot be edited. The actual character is displayed at the corresponding position in the literal mask.
a	Represents any of the letters a–z/A–Z. Capital letters are not converted into lower case.
A	Represents any of the letters A–Z. If lower-case letters are entered, they will automatically be converted to upper case.
c	Represents any of the characters a–z/A–Z plus the digits 0–9. Capital letters are not converted into lower case.
C	Represents any of the letters A–Z plus the digits 0–9. If lower-case letters are entered, they will automatically be converted to upper case.
N	Only the digits 0–9 can be entered.
x	All printable characters are allowed.
X	All printable characters are allowed. If lower-case letters are entered, they will automatically be converted to upper case.

So, for example, you can define the literal mask as "__/__/2012" and the edit mask as "NNLNNLLLLL", to allow the user to enter four characters only for a date.

Data tab

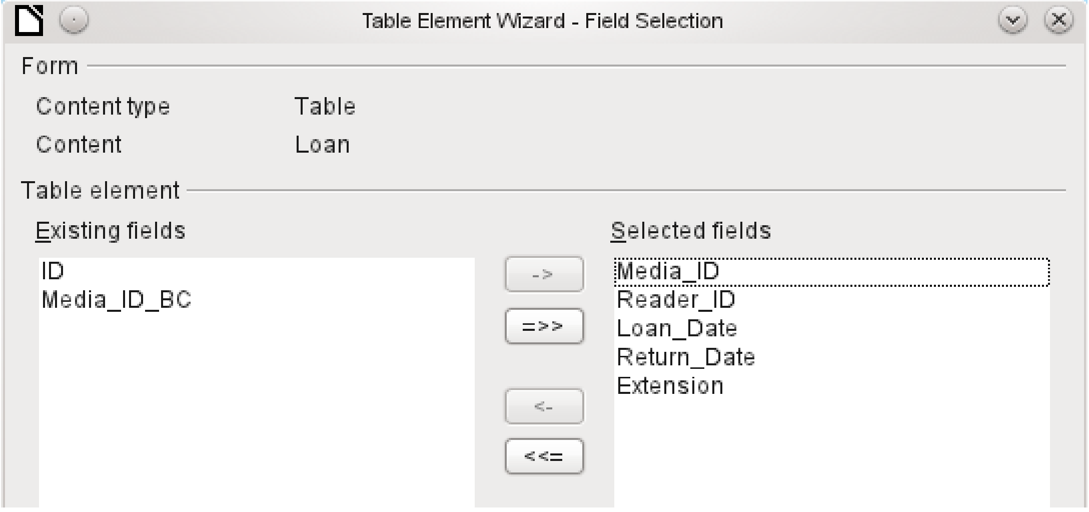
Nothing further to report.

Events tab

The *Changed* event is absent.

Table control

This is the most comprehensive control. It provides a table, which can then be provided with controls for individual columns. This not only allows the actual data to be viewed during input, but also the previously entered data, without the need to use the Navigation bar to scroll through the records.



Form

Content type Table

Content Loan

Table element

Existing fields

- ID
- Media_ID_BC

Selected fields

- Media_ID
- Reader_ID
- Loan_Date
- Return_Date
- Extension

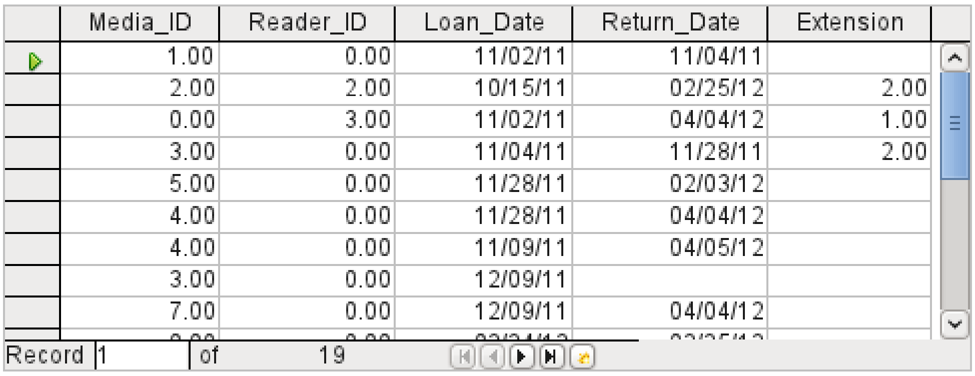
Figure 206: Table Element Wizard, field selection

Not every field that is possible in a form can be selected for a table control field. Push buttons, image buttons, and option buttons are not available.

The Table Control Wizard assembles in a window the fields that will appear afterwards in the table.

The Loans table is available for editing in the control. In addition to the ID (primary key) field and the Media_ID_BC field (entry of media using a bar-code scanner), all fields are to be used in the control.

The previously created table control must now be further developed, to allow entry into the Loans table. For fields such as Reader_ID or Media_ID, it would be more useful to be able to choose the reader or the media directly, rather than a number representing the reader or media. For this purpose, controls such as list boxes can be placed within the table control. This is declared later. The formatting of the Extension field with two decimal places was certainly not intended.



Media_ID	Reader_ID	Loan_Date	Return_Date	Extension
1.00	0.00	11/02/11	11/04/11	
2.00	2.00	10/15/11	02/25/12	2.00
0.00	3.00	11/02/11	04/04/12	1.00
3.00	0.00	11/04/11	11/28/11	2.00
5.00	0.00	11/28/11	02/03/12	
4.00	0.00	11/28/11	04/04/12	
4.00	0.00	11/09/11	04/05/12	
3.00	0.00	12/09/11		
7.00	0.00	12/09/11	04/04/12	

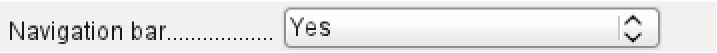
Record 1 of 19

Figure 207: Output of the Table Element Wizard

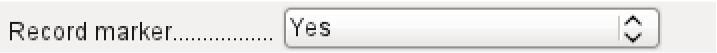
In addition to the properties listed in section "Default settings for many controls" above, the following features are available.

General tab

Height of individual lines. With no value here, the height is automatically adjusted to the font size. Multi-line text fields are then shown as single lines to be scrolled.
[RowHeight]


Navigation bar: Yes

As with tables, the lower edge of the control shows the record number and navigation aids. [HasNavigationBar]


Record marker: Yes

By default there is a record marker on the left edge of the control. It indicates the current record. You can use the record marker to access the delete function for the whole record. [HasRecordMarker]

Data tab

Since this is a field that contains no data itself but manages other fields, there are no data properties.

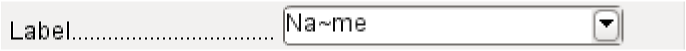
Events tab

The *Changed* and *Text modified* events are missing. The *Error occurred* event is added.

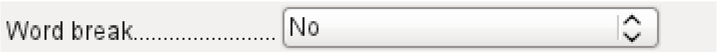
Label field

In addition to the properties described in section "Default settings for many controls" above, the following features are available.

General tab


Label: Na~me

The title acts as a description of another control. If the title is bound to a control, it can be used as a shortcut to it. An inserted “~” indicates a particular letter that becomes the shortcut “Na~me” defines “m” as this letter. When the cursor is anywhere in the form, *Alt+m* goes straight to the field.


Word break: No

By default a label is not wrapped. If it is too long for the field, it is truncated. Caution: word wrapping does not recognize spaces, so if the field is too small, a break can occur within a word. [MultiLine]

Data tab

None.

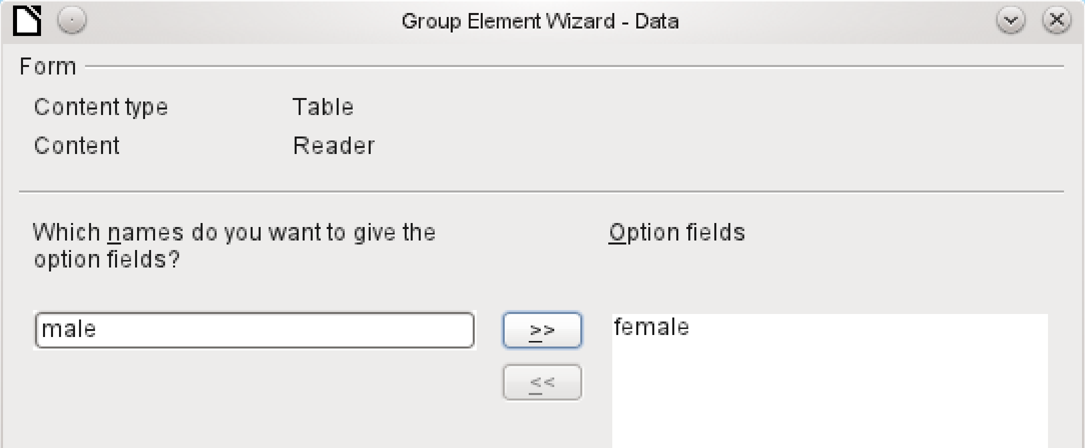
Events tab

The label field reacts only to events that are connected with the mouse, a key, or the focus.

Group box

A group box graphically groups several controls and provides them with a collective label.

If a group box is created with Wizards active, the Wizard proceeds from the assumption that several option buttons will occur together within this frame.



Group Element Wizard - Data

Form

Content type Table

Content Reader

Which names do you want to give the option fields?

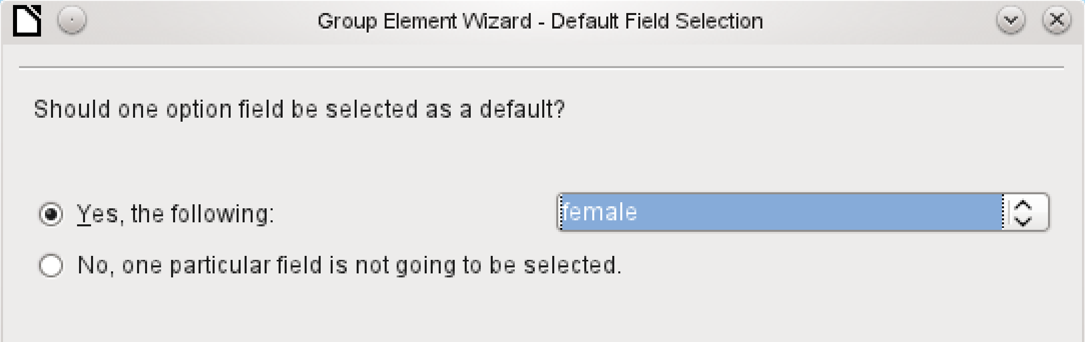
male

Option fields

female

Figure 208: Group Element Wizard, option field selection

This form is based on the Reader table. We are dealing with the choice of gender. The entries are the labels of the option buttons.



Group Element Wizard - Default Field Selection

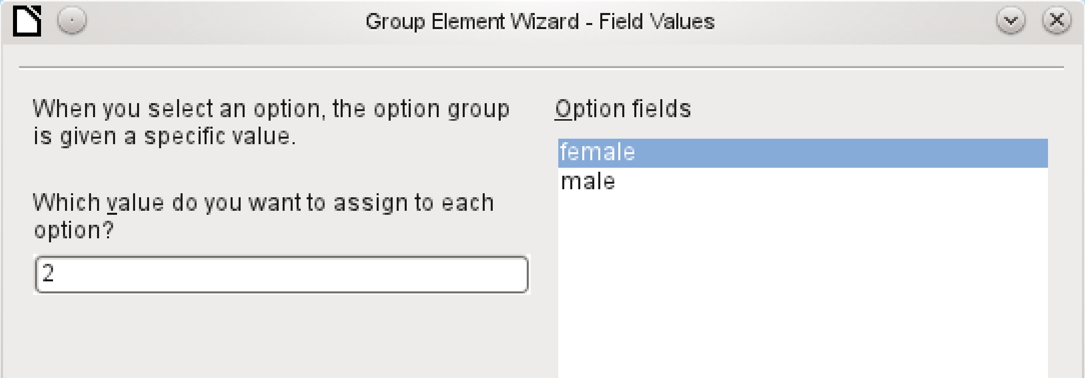
Should one option field be selected as a default?

● Yes, the following: female

○ No, one particular field is not going to be selected.

Figure 209: Group Element Wizard, default option field selection

Here the default option is "female". If there is to be no default field, the default entry in the underlying table is NULL.



Group Element Wizard - Field Values

When you select an option, the option group is given a specific value.

Which value do you want to assign to each option?

2

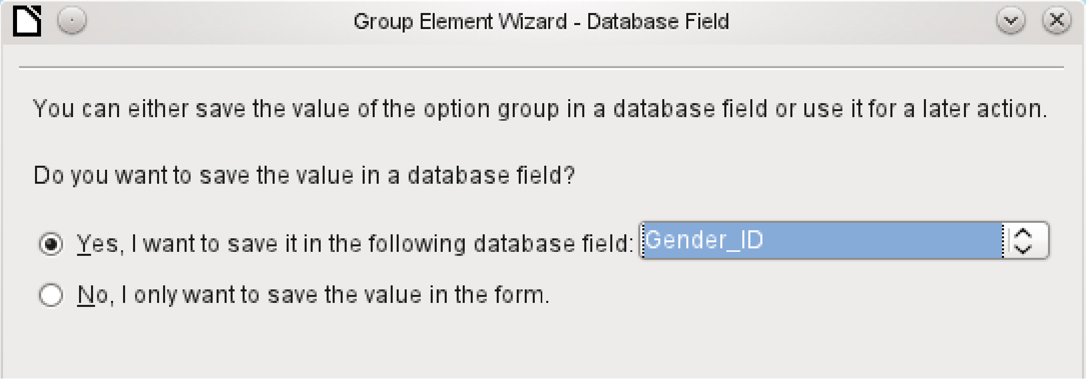
Option fields

female
male

Figure 210: Group Element Wizard, option field value assignment

The Wizard gives the option buttons separate values by default, here 1 for female and 2 for male. These values correspond to the examples of primary key fields in the Gender table.

The value selected by clicking an option button is transferred to the Gender_ID field of the form's underlying table Readers. In this way the Readers table is provided with the corresponding foreign key from the Gender table by using the option button.



Group Element Wizard - Database Field

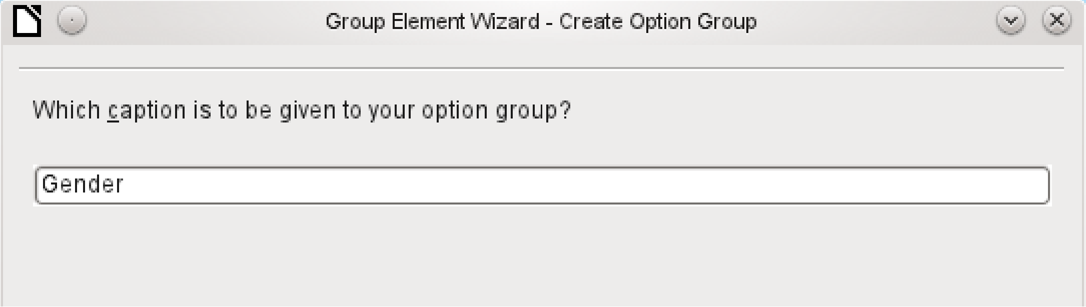
You can either save the value of the option group in a database field or use it for a later action.

Do you want to save the value in a database field?

● Yes, I want to save it in the following database field: Gender_ID

○ No, I only want to save the value in the form.

Figure 211: Group Element Wizard, select whether to save value in a database field



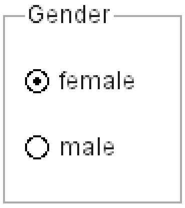
Group Element Wizard - Create Option Group

Which caption is to be given to your option group?

Gender

Figure 212: Group Element Wizard, caption definition

The option button group is given a group box (frame) with the label Gender.



Gender

● female

○ male

Figure 213: Example of an option button group

If female is selected in the active form, male is deselected. This is a characteristic of option buttons that are bound to the same field in the underlying table. In the example shown above, the option buttons replace a two-element list box.

In addition to the properties described in section "Default settings for many controls" above, the following features are available.

General tab
The label can be changed from its default value. At present the frame properties (Line thickness, line color) cannot be changed but you can change the font formatting.

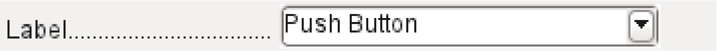
Data tab
None, since this control serves only for visual grouping of fields.

Events tab
The group box reacts to events involving the mouse, a key, or the focus.

Push button

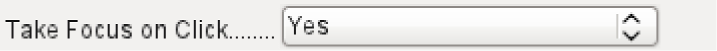
In addition to the properties described in section "Default settings for many controls" above, the following features are available.

General tab


Label.................................. Push Button

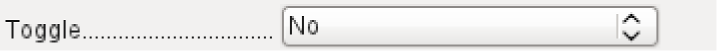
Label on the button.

By inserting “~”, a specific letter of the label can be turned into a shortcut. “Push ~Button” defines “b” as the shortcut. When the cursor is anywhere in the form, *Alt* + *b* will take you to the button.
[Label]

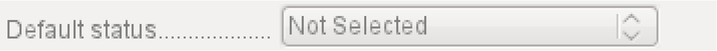

Take Focus on Click........ Yes

The button receives the focus when it is clicked.

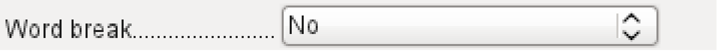
This might not always be desirable. For example, you might want a click on the button to put some content into a different form field, in which case the cursor should remain in that field when you click the button.
[FocusOnClick]


Toggle................................ No

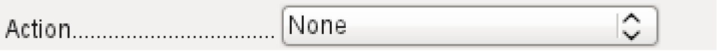
If Yes, the button can be shown pressed in. The button state is shown as for a switch. When you press it a second time, it shows an unpressed button.
[Toggle]


Default status.................. Not Selected

Active, when Toggle is set to Yes. *Selected* corresponds to the pressed-in button.
[DefaultState]

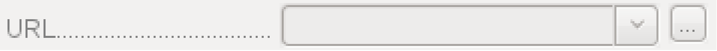

Word break....................... No

Word wrapping if the button is too narrow.
[MultiLine]


Action.................................. None

A variety of actions similar to those for the navigation bar are available.

If the action is to open a document or website, put the URL in the next field.

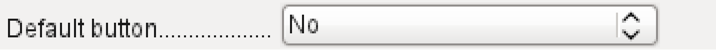

URL....................................

HTML: File to be called up with this button. Here you must choose the resource which is to be opened by ‘Open document/website’ under ‘Action’.

As well as HTML, ‘Action’ can be used to open other LO modules. So if you enter here **.uno:RecSearch**, the search function of the navigator will be assigned to the button.

You can also open Writer files so that pressing the button carries out a mail merge using records from the database.

The commands available here can be determined using the macro recorder.
[TargetURL]

Frame	Only for HTML forms: The target-frame (frame arrangement for different HTML pages) in which the file should be opened. [TargetFrame]
 Default button: No	The default button is framed when this is set to Yes. When there are several alternative buttons on a form, the one most often used should have this characteristic. It is activated by pressing the Enter key, when no other action needs to depend on this key. Only one button on the form can be the default button. [DefaultButton]
Graphics	Should a graphic appear on the button? [Graphic] [ImageURL]
Graphics alignment: Centered	Only active when a graphic has been selected. Specifies the alignment of the graphic to the text. [ImagePosition] [ImageAlign]

Data tab

None. A button only carries out actions.

Events tab

Approve action, *Execute action*, and *Item status changed*.

Image button

In addition to the properties already described in section "Default settings for many controls" above, the following features are available.

General tab

Similar to a normal button. However this button has no text and the button itself is not visible. You see only a frame around the graphic.

By default, an image button has no tab stop.

Caution: at the time of writing, hardly any actions work with this button. It is practically only usable with macros.

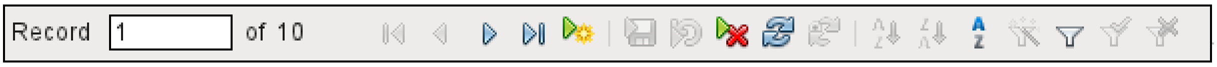
Data tab

None; this control only carries out actions.

Events tab

Approve action and all events involving the mouse, a key, or the focus.

Navigation bar



Record 1 of 10

Figure 214: Navigation bar control

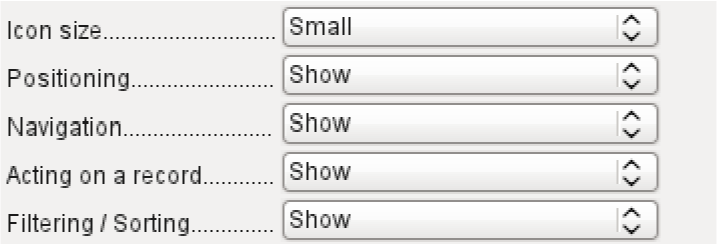
The standard Form Navigation bar is inserted into forms at the lower edge of the screen. The insertion of this toolbar can cause a brief rightward shift of the form as it builds up on the screen. This can be distracting in cases where the navigation bar is switched off again for some parts of the visible form, for example when there are subforms or more than one form in the visible form.

By contrast, a navigation bar control that is part of the form, separate from the corresponding items, makes it clear through which items you navigate with the toolbar. The form for Loans, for example, needs to search first through the readers and then show the media loaned to the reader. The navigation bar control is positioned near the reader, so the user notices that the navigation bar is used for the reader and not for the media loaned to the reader.

The standard Form Navigation bar makes available the buttons shown in Figure 215. The navigation bar control shows the same buttons except those for Find Record, Form-Based Filters, and Data source as Table.

In addition to the properties listed in section "Default settings for many controls" above, the following features are available for the Navigator Bar control.

General tab



Icon size	Small
Positioning	Show
Navigation	Show
Acting on a record	Show
Filtering / Sorting	Show

The icon size is adjustable. In addition you can choose which groups are displayed. These are shown in Figure 214 from left to right using a vertical line as a group separator: Positioning, Navigation, Acting on a record, and groups of commands for Filtering and Sorting.
[IconSize]
[ShowPosition]
[ShowNavigation]
[ShowRecordActions]
[ShowFilterSort]

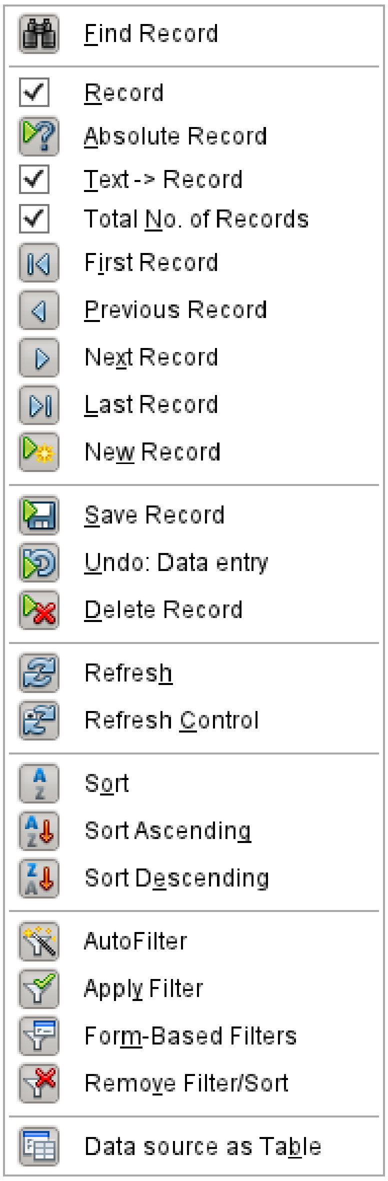


Figure 215: Default allocation of icons on the Form Navigation toolbar

Data tab

None, as this control only carries out actions.

Events tab

All events that involve the Mouse, a key, or the focus.

Independent of this form control, the *insertable navigation bar* naturally continues to exist with the same items as the above figure.

This insertable navigation bar provides additionally the general record search, the form-based filter and the display of the form's underlying data source in table view above the form.

If you are working not just with a form but with subforms and ancillary forms, you must be careful that this insertable navigation bar does not disappear as you switch forms. That creates a disturbing effect on the screen.

Spin buttons and scrollbars

Neither of these fields has a direct connection to data in the database. They can only be utilized by means of macros, where the spin button is integrated into, for example, a numeric field.

A scrollbar differs from a spin button in having slider in addition to buttons for incrementing and decrementing the value within a fixed range.

When the form is opened, the scrollbar must be told the current value of the field to which it is linked. This macro is bound to **Form Properties > Events > After record change**.

```
SUB scrollbar_form(oEvent AS OBJECT)
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   oForm = oEvent.Source
   oField = oForm.getByName("scrollbar")
   oFeld.ScrollValue = oForm.getInt(oForm.findColumn("numeric_value"))
END SUB
```

First the variable is declared. The event that it will handle is specified on the form to which the macro will eventually be bound. The scrollbar will be found by its name on the form. Its current value is set to the number stored in numeric_field in the underlying table.

If the value is altered using the scrollbar, this change must be transmitted to the underlying field. This macro is bound to **Properties: Scrollbar > Events > While adjusting**.

```
SUB Scrollbar_change(oEvent AS OBJECT)
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM inValue AS INTEGER
  oField = oEvent.Source.Model
   inValue = oEvent.Value
   oForm = oField.Parent
   oForm.updateInt(oForm.findColumn("numeric_value"),inValue)
END SUB
```

First the variables are declared. The selected value is read as an integer from the field. The corresponding field in the underlying table numeric_value is then updated to this value.

A detailed explanation of this kind of code is given in Chapter 9, Macros.

Hidden control

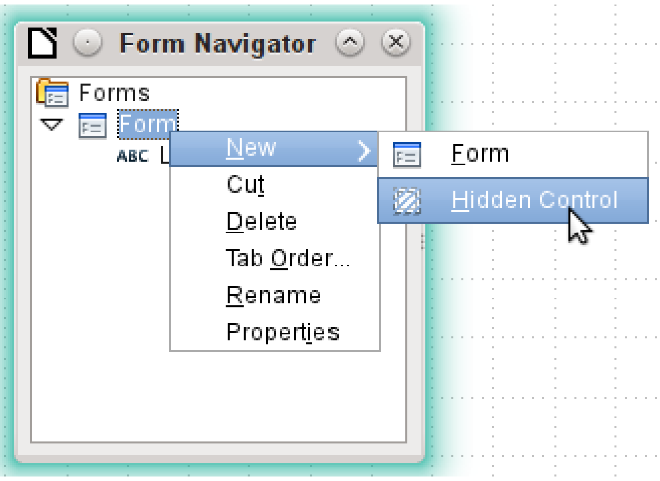


Figure 216: Creating a hidden control using the Form Navigator

One type of control that cannot be inserted from the controls toolbar is the Hidden Control. It must be created within the form navigator, using the form's context menu.

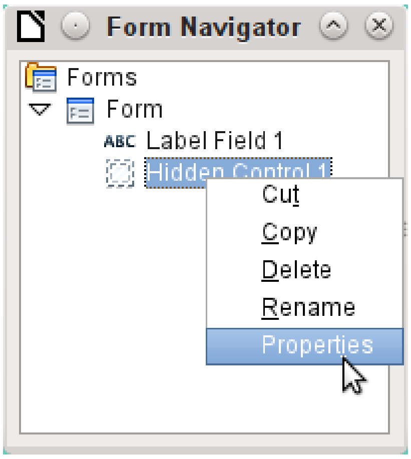
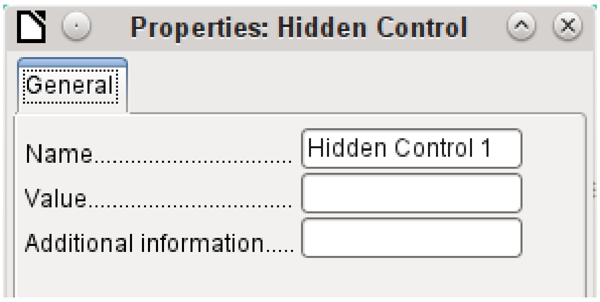


Figure 217: Accessing the Properties dialog for a hidden control

Like any other control, a hidden control is part of the form. However it is not visible in the user interface.

A hidden control has only a few properties. The Name and the Additional information have the same meaning as for other controls. In addition, the control can be assigned a value [Hidden Value].

There is no point in a hidden control if you are not using macros. With macros however, it is often useful to be able to store intermediate values somewhere on the form, to be accessed later. An example of this way of using macros is described in Chapter 9, Macros, in the section "Hierarchical Listboxes".


Properties: Hidden Control

General

Name: Hidden Control 1

Value:

Additional information:

Figure 218: Properties dialog for a hidden control

Multiple selection

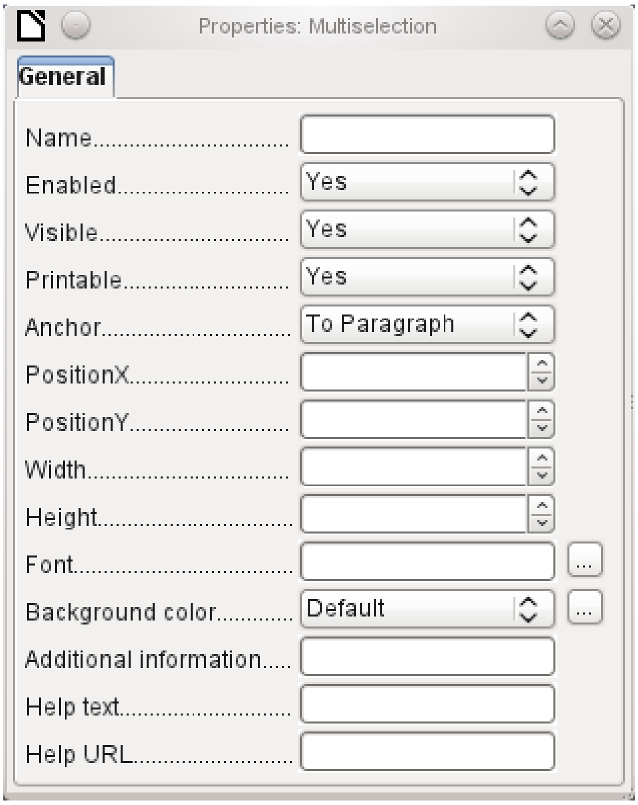
If you use the **Select** button to select a large region or several elements of a form, the following modifications may be carried out.

You should not alter the name. That would cause all the selected elements suddenly to acquire the same name. It would make finding individual elements using the Form Navigator difficult, and management of the form by using named controls in macros impossible.

Multiple selection is more useful for changing the font, the height or the background color of controls. Note that changing the background color affects the labels as well.

If you want to alter only the labels, hold down the *Control* key and click these controls directly or in the Navigator, or right-click on a field to call up the control properties. Now the choice of properties that you can change is greater as you are dealing only with similar fields. You can change anything here that is available in a label field.

The possibilities of multiple selection depend therefore on the choice of fields. You can simultaneously change controls of the same kind that have all the properties that exist for a single instance.



Name	
Enabled	Yes
Visible	Yes
Printable	Yes
Anchor	To Paragraph
PositionX	
PositionY	
Width	
Height	
Font	
Background color	Default
Additional information	
Help text	
Help URL	

Figure 219: Properties dialog for a multiple selection

A simple form completed

A simple form has form controls for writing or reading records from a single table or query. Its construction is shown by the following example.

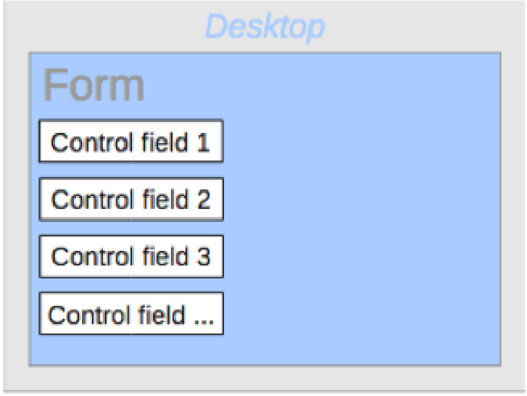


Figure 220: Schematic representation of a simple form

The example of a simple form for library loans is shown here using several variants. The quick way to use the Form Wizard is described in Chapter 8, Getting Started with Base, in the *Getting Started Guide*. Here we describe the creation in Design View.

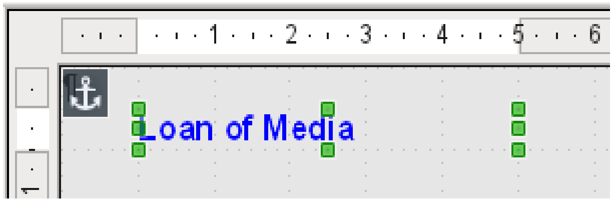


Figure 221: Inserting a label to provide a heading for the form

The heading for the form was created using a label field. The font was changed. The label field is anchored to a paragraph in the top left corner of the document. Using the context menu of the label field, a form was created that was linked to the Loans table (see “Form properties” above). The page has also been given a uniformly colored background.

Adding groups of fields

A quick variant for direct entry of fields with labels is to use the Add Field function.

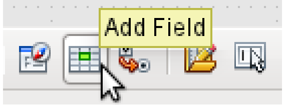
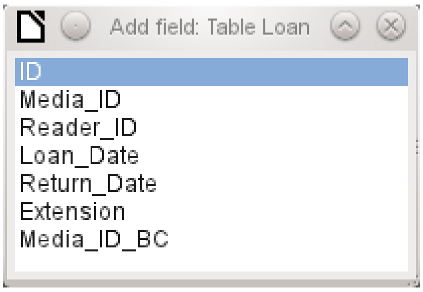


Figure 222: Add Field icon on Formula Design toolbar

This function, available on the Form Design toolbar (see Figure 184), allows all fields of the underlying table to be selected.



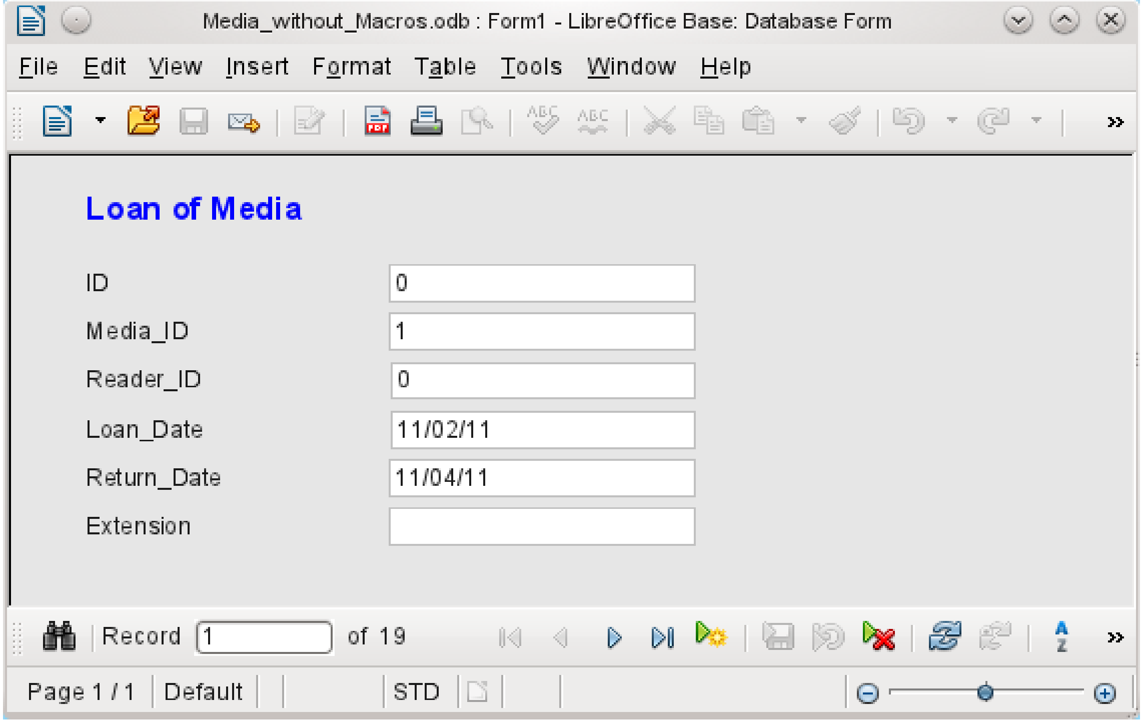
Add field: Table Loan

ID
Media_ID
Reader_ID
Loan_Date
Return_Date
Extension
Media_ID_BC

Figure 223: Using the Add field dialog to select which field of the underlying table is to be added

Double-click on the fields to insert them into the form as a group with labels, initially all stacked at the same location. The group needs to be separated out so that the form eventually looks like that shown in Figure 224. For a better view, all unnecessary toolbars have been removed from the window, which has also been compressed so that not all elements of the Navigation bar are visible.

All fields have been selected except Media_ID_BC, which is designed to be used only with a barcode scanner.


Media_without_Macros.odb : Form1 - LibreOffice Base: Database Form
File Edit View Insert Format Table Tools Window Help
Record 1 of 19
Page 1 / 1 Default STD

Loan of Media

ID	0
Media_ID	1
Reader_ID	0
Loan_Date	11/02/11
Return_Date	11/04/11
Extension	

Figure 224: Simple form created using the Add Field function

For each table field, an appropriate form control has been automatically selected. Numbers are in numeric fields and are declared as integers without decimal places. Date fields are represented correctly as date controls. All fields have been given the same width. If a graphical control had been included, it would have been given a square field.

Adjusting field proportions

We can now do some creative things, including adjusting the length of the fields and making the dates into drop-down fields. More important still is for the Media_ID and the Reader_ID fields to be made more user-friendly, unless every library user has a library ID card and every medium is supplied with an ID on accession. That will not be assumed in what follows.

To adjust individual fields, we must edit the group. This can be done with a right-click on the group and then following the context menu (Figure 226). For future work, it will be clearer if we use the Form Navigator.

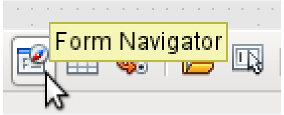


Figure 225: Form Navigator icon on Form Design toolbar

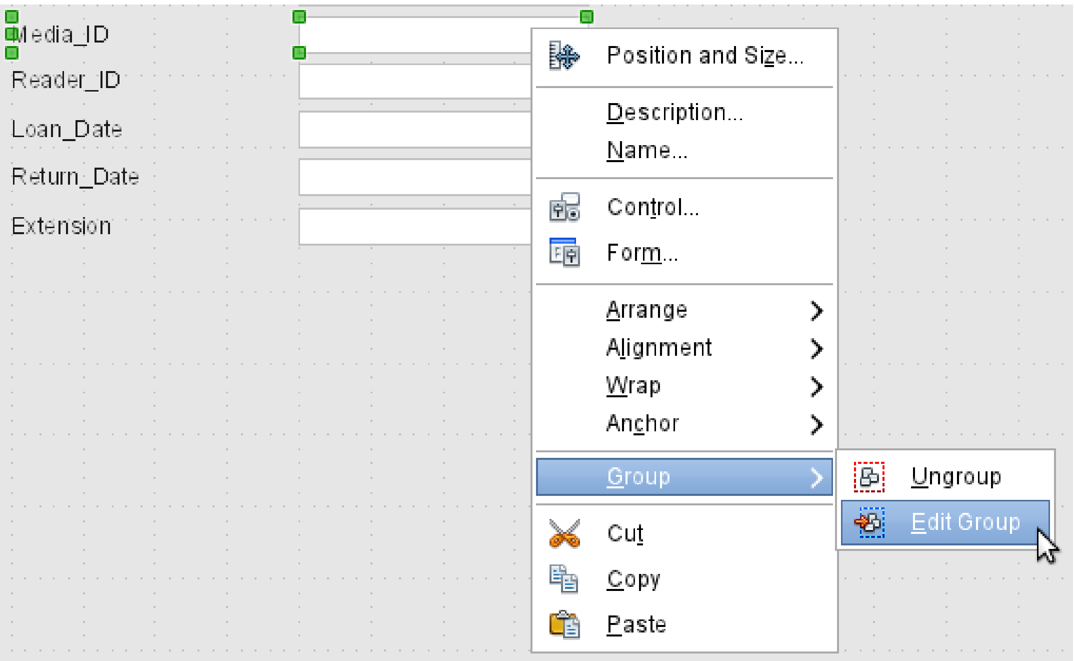


Figure 226: Using the context menu to ungroup or enter / exit a group of form controls

The Form Navigator displays all the elements of the form with their labels. For controls, the names are taken directly from the names of the fields in the underlying table. The names of the labels have the suffix Label.

A click on Media_ID selects this field (Figure 227). Right-click to replace the selected field with a different type of field, using the context menu (Figure 228).

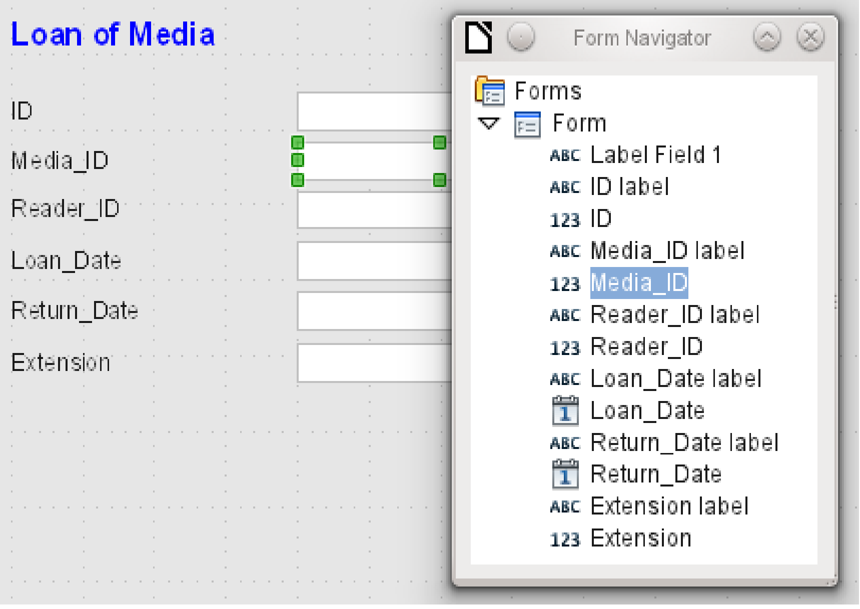


Figure 227: Selecting form controls directly using the Form Navigator

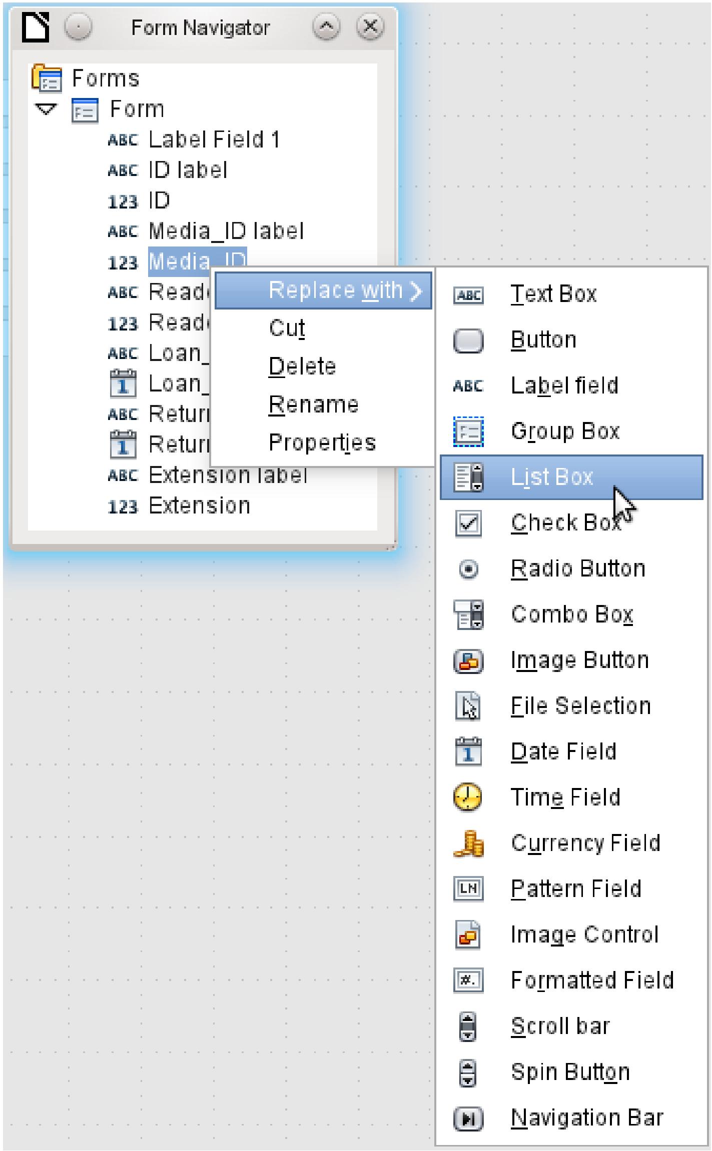


Figure 228: Replacing one control type with another using the Form Navigator

This replacement is carried out for the Media_ID and Reader_ID controls.

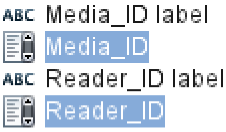
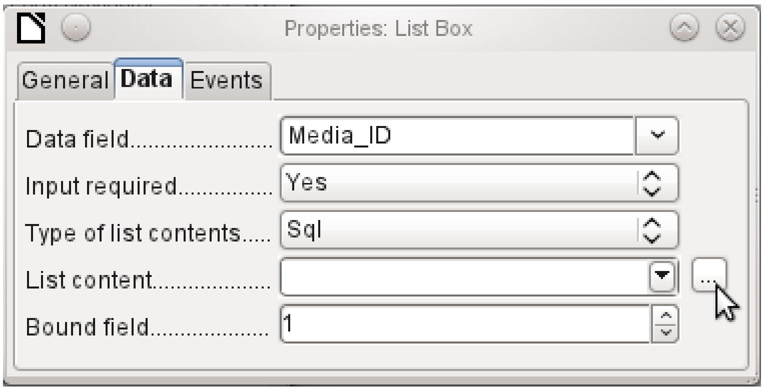


Figure 229: Media_ID and Reader_ID replaced with list boxes

The change is made visible in the Form Navigator by the change in the accompanying icon.



Data field: Media_ID
Input required: Yes
Type of list contents: Sql
List content:
Bound field: 1

Figure 230: Properties dialog (Data tab) for a list box

The SQL query for the list field can now be created through the graphical user interface by clicking on the button at the right. This is carried out automatically when a list box is created directly, but not when it is formed by conversion from another type of control. For the SQL command, see Chapter 5, Queries.

Since the list boxes are to be made drop-down, the following defects can be corrected at the same time:

- The labels for the list boxes should be Media instead of Media_ID and Reader instead of Reader_ID.
- The ID control should be declared as read-only.
- Any fields which are not absolutely necessary for issuing loans for a new medium do not need a tab stop. Without it, the form can be traversed much faster. If necessary, the tab stop can also be adjusted using the activation sequence (see "Default settings for many controls" above). Only the Media, Reader, and Loan_date fields must be accessible in all cases using the Tab key.
- If the form is intended for carrying out loans, it is unnecessary and also confusing for returned media to be displayed. Media with a return date should be filtered out. In addition, the display order could be sorted by Reader, so that media on loan to the same person are displayed successively. See the note on “Form properties” above. However, there is a problem here in that readers can be sorted only by ID, not alphabetically, because the table underlying the form only contains the ID.

Adding single fields

The addition of single fields is a bit more complicated. The fields must be selected, dragged onto the form surface, and the appropriate field from the underlying table specified. In addition, the type of field must be correctly chosen; for example, numeric fields have two decimal places by default.

Only when creating list boxes does the Wizard come into play, making it easier for a novice to carry out the steps for creating the correct fields—up to a point. Beyond that point, the Wizard ceases to meet requirements because:

- The entries are not automatically sorted.
- Combining several fields in the list box content is not possible.

Here again we need to make retrospective improvements, so that the required SQL code can be created quite quickly using the built-in query editor.

When adding single controls, the field and its label must be explicitly associated (see “Default settings for many controls” above). In practice it could be better if you do not associate fields with the labels, so you do not have to use Edit Group before changing the properties of a field.

Table control

The use of the Table Wizard to create a table control has already been described in section "Table control" above. It has however some defects which need to be improved:

- The Media_ID and Reader_ID fields must become list boxes.
- Numeric fields must be stripped of their decimal places, since the Wizard always specifies two decimal places for numbers.

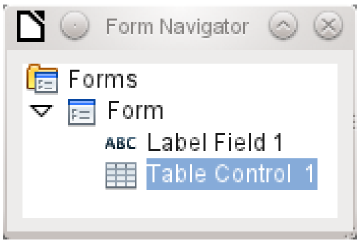


Figure 231: Table control shown in Form Navigator

Changing fields within the table control is not possible using the same method as described for other controls. In the Navigator, the description of fields ends with the table control. The Navigator knows nothing about the controls that lie within the table control, referring to fields in the underlying table. This equally applies later, when attempts are made to access the fields using macros. They cannot be accessed by name.

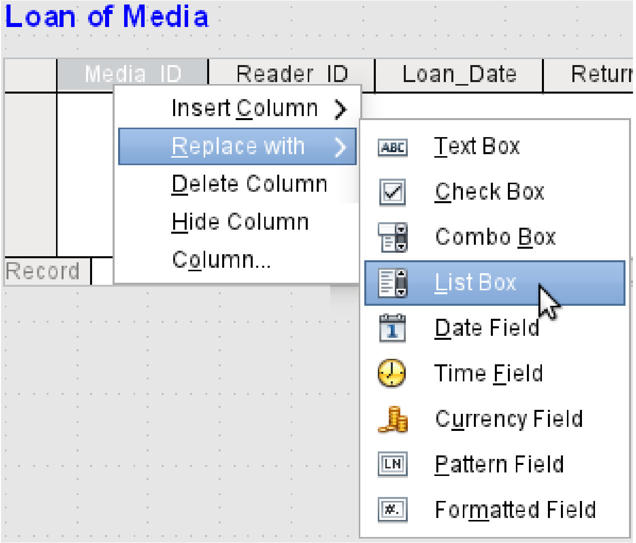


Figure 232: Replacing one control type with another for a column in a table control

The controls within the table control are called columns. Using the context menu, it is now possible to replace one type of field by another. However the whole range of types is not available. There are no push buttons, option boxes, or graphical controls.

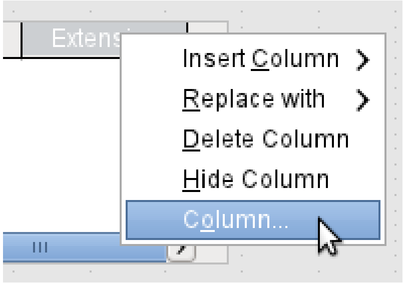


Figure 233: Accessing the Properties dialog for a column in a table control

The properties of the fields are hidden in the context menu behind the concept of columns. Here, for example, the numeric field Extension can be changed so that no decimal places are shown. Also the default minimum value of –1,000,000.00 hardly makes sense for a loan extension. The number should always remain small and positive.

As soon as the properties of a column are called up, you can select another column without shutting the properties dialog. In this way you can work on all the fields, one after another, without having to save in between.

Note

The value is saved in the dialog as soon as movement is made between properties. Such is also the case when moving from one column to the next.

End by saving the entire form, and finally the database itself.

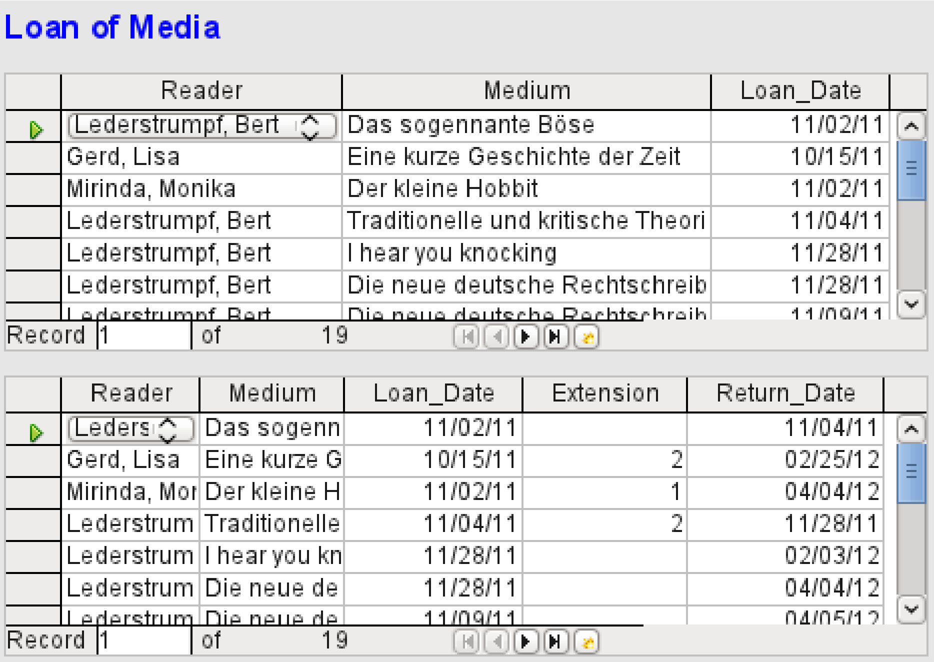
The properties of the fields built into a table control are not so comprehensive as for those outside. The font, for example, can be set only for the entire table control. In addition, you do not have the option of skipping individual columns by removing their tab stops.

Tip

You can move through a form using either the mouse or the Tab key. If you tab into a table control, the cursor will move one field to the right for each additional tab; at the end of the line, it will move back to the first field of the next record in the table control. To exit the table control, use Ctrl+Tab.

If only certain fields should be visible during use, you can use several different table controls in the form, as the Tab is captured by default by each table control.

The form shown in Figure 234 is for the loan of media. Only the fields immediately necessary are shown in the upper table control. The lower one shows all the fields, so that it is apparent which person and medium the return is for.



Loan of Media

	Reader	Medium	Loan_Date
▷	Lederstrumpf, Bert	Das sogennante Böse	11/02/11
	Gerd, Lisa	Eine kurze Geschichte der Zeit	10/15/11
	Mirinda, Monika	Der kleine Hobbit	11/02/11
	Lederstrumpf, Bert	Traditionelle und kritische Theori	11/04/11
	Lederstrumpf, Bert	I hear you knocking	11/28/11
	Lederstrumpf, Bert	Die neue deutsche Rechtschreib	11/28/11
	Lederstrumpf, Bert	Die neue deutsche Rechtschreib	11/09/11

Record 1 of 19

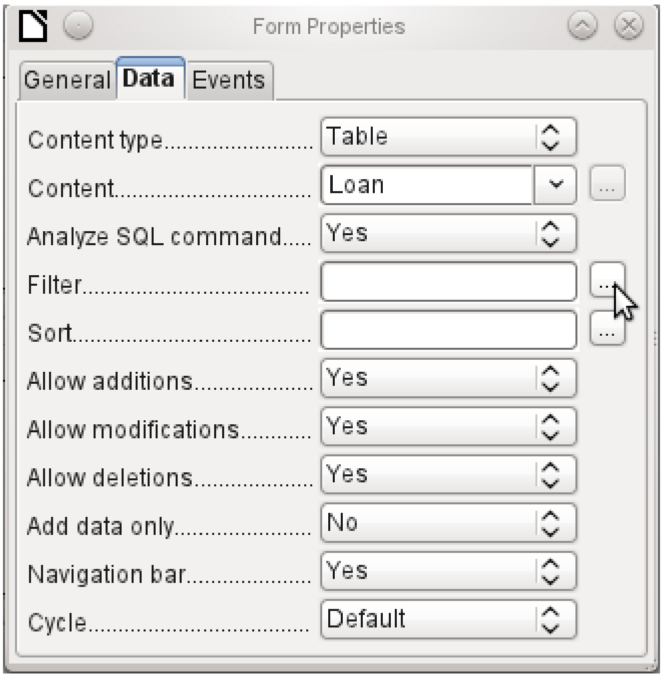
	Reader	Medium	Loan_Date	Extension	Return_Date
▷	Leders	Das sogenn	11/02/11		11/04/11
	Gerd, Lisa	Eine kurze G	10/15/11	2	02/25/12
	Mirinda, Mor	Der kleine H	11/02/11	1	04/04/12
	Lederstrum	Traditionelle	11/04/11	2	11/28/11
	Lederstrum	I hear you kn	11/28/11		02/03/12
	Lederstrum	Die neue de	11/28/11		04/04/12
	Lederstrum	Die neue de	11/09/11		04/05/12

Record 1 of 19

Figure 234: A form with multiple table controls

This figure shows an aesthetic failing that needs urgent attention. In the upper table control, the same medium sometimes occurs more than once. This happens because the table also shows media that have been returned earlier. Therefore the data needs to be filtered to show only the loans. Records with a return date should not appear.

This filtering is possible either by a query or directly using the form properties. If it is done using the form properties, the filter can be temporarily switched off during input. Filtering using a query is described in Chapter 5, Queries. Here we describe how to do it using form properties.



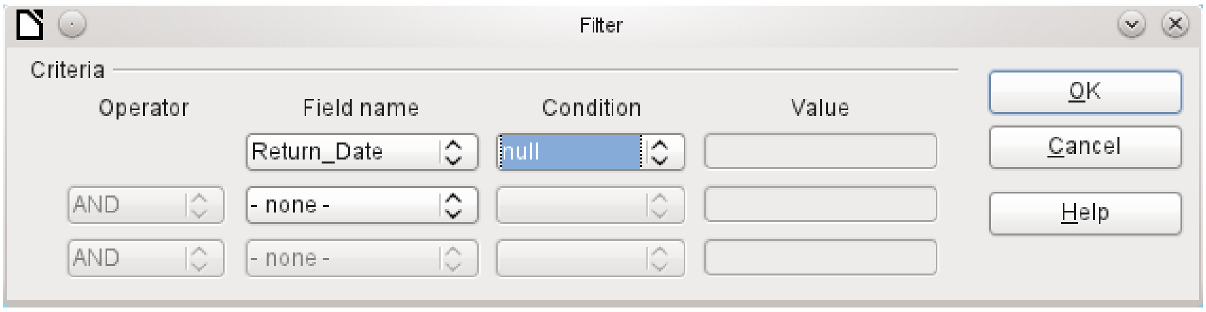
Form Properties

General | **Data** | Events

Content type	Table
Content	Loan
Analyze SQL command	Yes
Filter	
Sort	
Allow additions	Yes
Allow modifications	Yes
Allow deletions	Yes
Add data only	No
Navigation bar	Yes
Cycle	Default

Figure 235: Click the ellipsis icon adjacent to the Filter field on the Data tab of the Form Properties dialog

The filtering is carried out using the button with the three dots (an ellipse, which opens the dialog shown below. You can also enter the filter directly into the Filter text field if you know the SQL coding.



Operator	Field name	Condition	Value
	Return_Date	null	
AND	- none -		
AND	- none -		

Figure 236: Filter dialog

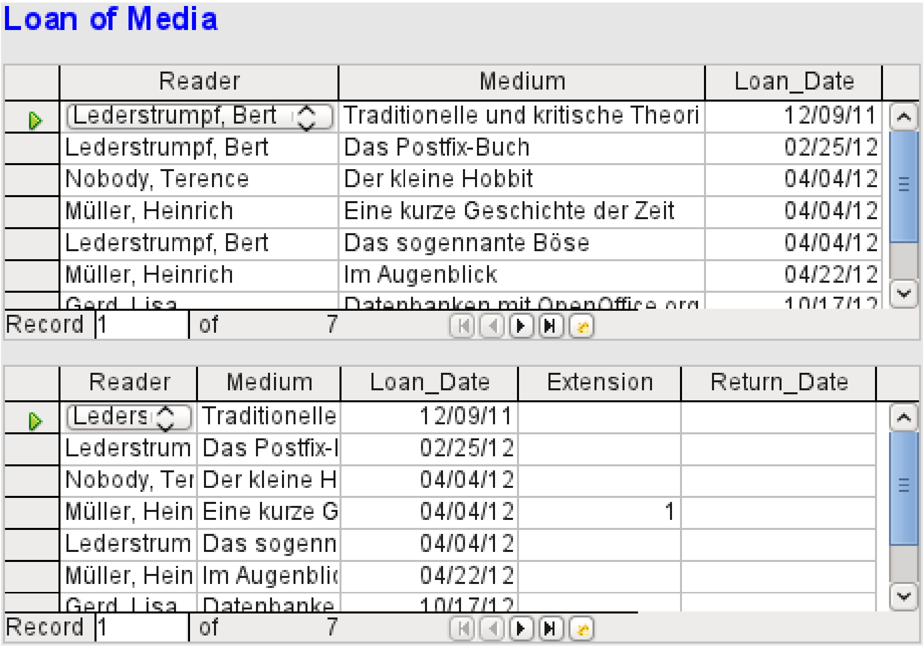
Using the GUI, you can now select the field named Return_Date. It will show only the records for which the field is empty, where "empty" stands for the SQL designation NULL.

The cleaned-up form (shown in Figure 237) now looks rather simpler.

Of course there is still room for improvement, but compared with the earlier form, this version has a clear advantage in that all the media are visible at a glance.

The processing of data using table controls is similar to using an actual table. A right-click on the record header of an existing record causes it to be deleted, and an entry can be canceled or saved in the case of new records.

When you leave a line, the record is automatically saved.


Loan of Media

Reader	Medium	Loan_Date
Lederstrumpf, Bert	Traditionelle und kritische Theori	12/09/11
Lederstrumpf, Bert	Das Postfix-Buch	02/25/12
Nobody, Terence	Der kleine Hobbit	04/04/12
Müller, Heinrich	Eine kurze Geschichte der Zeit	04/04/12
Lederstrumpf, Bert	Das sogennante Böse	04/04/12
Müller, Heinrich	Im Augenblick	04/22/12
Gerd, Lisa	Datenbanken mit OpenOffice.org	10/17/12

Record 1 of 7

Reader	Medium	Loan_Date	Extension	Return_Date
Lederst	Traditionelle	12/09/11		
Lederstrum	Das Postfix-I	02/25/12		
Nobody, Ter	Der kleine H	04/04/12		
Müller, Hein	Eine kurze G	04/04/12	1	
Lederstrum	Das sogenn	04/04/12		
Müller, Hein	Im Augenblic	04/22/12		
Gerd, Lisa	Datenbanke	10/17/12		

Record 1 of 7

Figure 237: Amended form with multiple table controls

We can still improve the Loan of Media form in a number of ways.

- It would be nice if selecting a reader in one part of the form caused the media on loan to this reader to be displayed in another.
- In the table shown above, you can see a lot of records that are not necessary because these media are already on loan. The table was created to allow loans to be made, so it would be better if only an empty page appeared, which could then be filled with the new loan.

Such solutions are available using further forms that are hierarchically arranged and make possible separate views of the data.

Tip

Table controls can be combined with other fields in the form. For example, the table control field shows only the titles of those media for which changes have been made in the underlying form fields.

When table controls are combined with other form fields, there is a minor problem, which can be easily worked around. If both types of the field are present together in a form, the cursor moves automatically from the other fields into the table control even though by default this type of field has the setting Tabstop > No. This can be fixed by switching the tabstop property to Yes and then back to No again. This will cause the “No” value to be registered.

Main forms and subforms

A subform lies within a form like a form control. Like a form control, it is bound to data from the main form. However its data source can be another table or a query (or a SQL command). The important thing for a subform is that its data source is somehow linked to the data source of the main form.

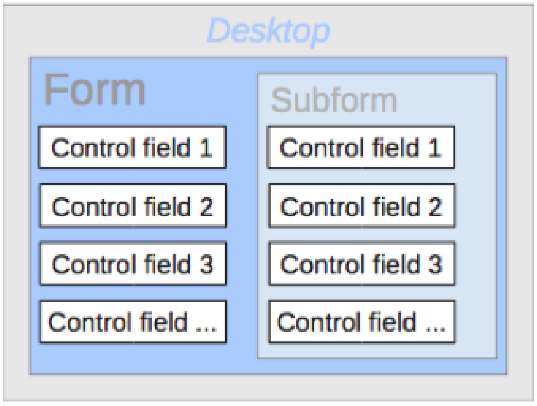
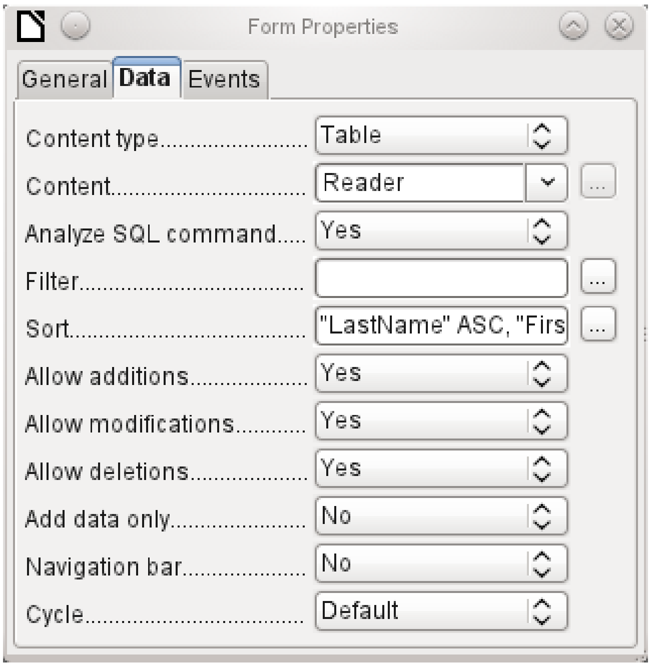


Figure 238: Schematic representation of a form with a subform

Typical table structures that lend themselves to use as subforms are tables with a one-to-many relationship (see Chapter 3, Tables). The main form shows a table with records to which many dependent records in the subform can be linked and displayed.

First we will use the relationship of the Reader table to the Loan table (see Chapter 3, Tables). The Reader table will form the basis for the main form and the Loan table will be reproduced in the sub-form.



Form Properties

General | **Data** | Events

Content type	Table
Content	Reader
Analyze SQL command	Yes
Filter	
Sort	"LastName" ASC, "Firs
Allow additions	Yes
Allow modifications	Yes
Allow deletions	Yes
Add data only	No
Navigation bar	No
Cycle	Default

Figure 239: Form Properties dialog (Data tab) for a form linked to the Reader table that is sorted alphabetically

Here the main form is linked to the Reader table. To speed up the search for readers, the table is sorted alphabetically. We will do without a navigation bar, since the content of the subform would come between the main form and the navigation bar. Instead we will use the built-in form control (Figure 214).

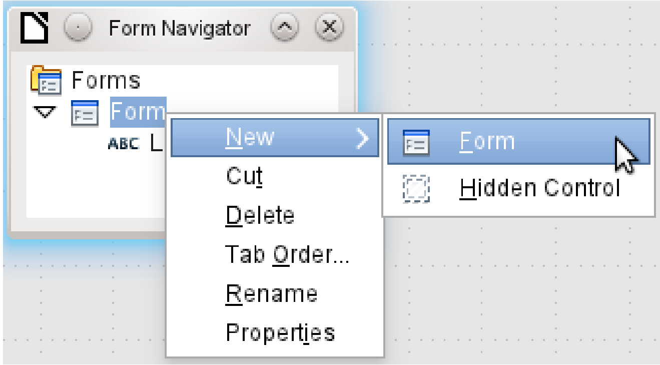


Figure 240: Creating a new subform using the Form Navigator

Right-click on the main form in the Form Navigator to use the context menu to create a new form. Once again this form has the default name of Form, but it is now an element in the subfolder of the main form.

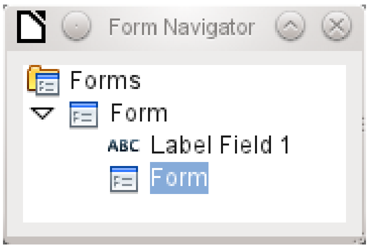
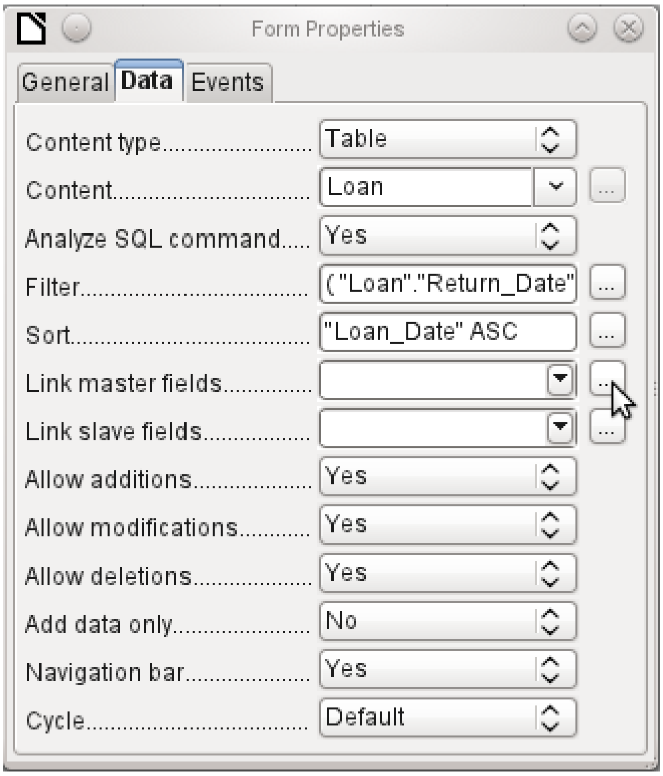


Figure 241: One form within the subfolder of another

The properties of the subform must now be set up to give it the right data source, in order to reproduce the data for the correct reader.

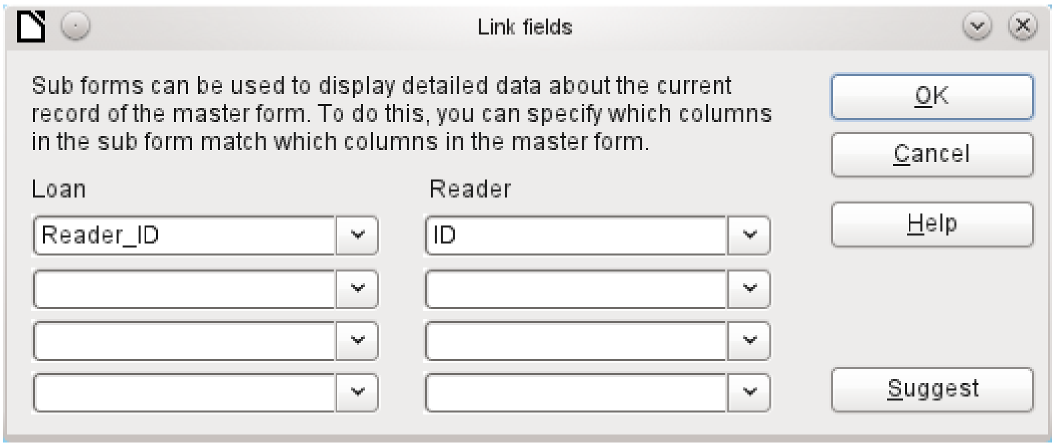


Content type	Table
Content	Loan
Analyze SQL command	Yes
Filter	("Loan"."Return_Date"
Sort	"Loan_Date" ASC
Link master fields	
Link slave fields	
Allow additions	Yes
Allow modifications	Yes
Allow deletions	Yes
Add data only	No
Navigation bar	Yes
Cycle	Default

Figure 242: Form Properties dialog (Data tab) for a subform

The Loans table is chosen for the subform. For the filter we specify that the Return_Date field should be empty ("Return_Date" IS NULL). This prevents any media that have already been returned from appearing. The records should be sorted by loan date. The ascending sort shows the medium on loan for the longest period at the top.

Link master fields and Link slave fields are used to create a linkage to the main form, in which the subform lies. The button with three dots shows once again that a helpful dialog is available for creating these.



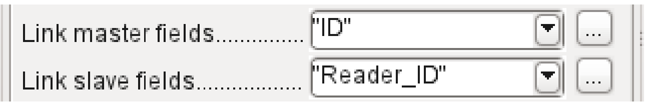
Sub forms can be used to display detailed data about the current record of the master form. To do this, you can specify which columns in the sub form match which columns in the master form.

Loan	Reader
Reader_ID	ID

Figure 243: Using the Link fields dialog to link columns in the master form and the subform

Under Loans, the fields in the Loans table are shown, under Readers those of the Reader table. The Reader_ID from Loans should be set as equivalent to the ID from the Reader table.

Although this linkage has already been created in the database using Tools > Relationships (see Chapter 3, Tables), the function that lies behind the Suggest button in this dialog does not reference this and would suggest that the first foreign key in the Loan table, namely Media_ID, should be linked with ID from the Reader table. The Form Creation Wizard solves this better by reading the relation from the relationship of the database.



Link master fields	"ID"
Link slave fields	"Reader_ID"

Figure 244: ID field in Reader table linked to Reader_ID field in Loan table

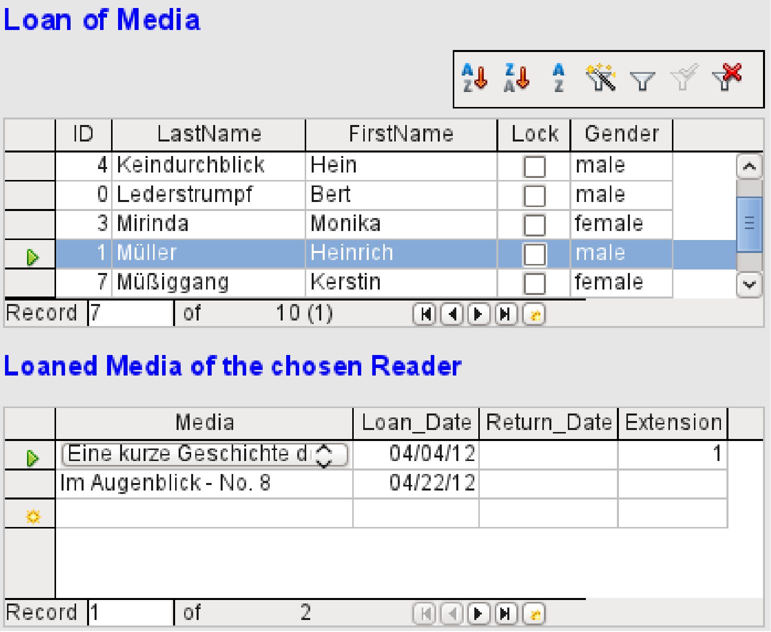
The chosen link between the table for the subform and the table for the main form is now specified in terms of fields from the tables.

To create a table control for the main form, we must now select the main form in the Form Navigator. Then, if the Table Control Wizard is enabled, it will show the fields available in the main form. We deal with the subform in a similar way.

Once the table controls have been set up, we need to carry out the modifications already discussed when creating the simpler form:

- Replacing the numeric field Media_ID in the subform with a list box.
- Renaming the Media_ID field to Media.
- Modifying the numeric fields to a format without decimal places.
- Limiting the minimum and maximum values.
- Renaming other fields, to save space or to add non-ASCII characters which should not be used in field names in database tables.

Sort and filter functions are supplemented for the main form by adding a navigation bar. The other fields on the navigation bar are not needed, as they are mostly available from the table control (record display, record navigation) or else carried out by movement through the table control (data storage). The final form might look like Figure 245.



Loan of Media

	ID	LastName	FirstName	Lock	Gender
	4	Keindurchblick	Hein	□	male
	0	Lederstrumpf	Bert	□	male
	3	Mirinda	Monika	□	female
▷	1	Müller	Heinrich	□	male
	7	Müßiggang	Kerstin	□	female

Record 7 of 10 (1)

Loaned Media of the chosen Reader

	Media	Loan_Date	Return_Date	Extension
▷	Eine kurze Geschichte d	04/04/12		1
	Im Augenblick - No. 8	04/22/12		
✱				

Record 1 of 2

Figure 245: Form consisting of a main form (above) and a subform (below)

If a reader is now selected in the main form, the subform will show the media on loan to that reader. When an item is returned, it continues to appear on the form until the form itself is refreshed. This occurs automatically when another record is loaded into the main form. If the original reader is selected again, returned media are no longer displayed.

This delayed updating is actually desirable in this case, as it allows one to inspect the media currently lying on the library counter and see at once whether these have been registered.

This form structure is significantly easier to use than the previous one with only a single form. However there are still details that can be improved:

- Media and loan dates might be changed when the media is to be loaned out for longer. Changing the media date might make it impossible to trace which item is still available in the library and which is on loan. Changing the loan date could lead to errors. Recall notices could not be verified.
- If a reader record is not selected by clicking on the record header at the left, only the little green arrow on the header shows which record is currently active. It is quite possible that the active record will be scrolled right out of the table control window.
- Instead of the text “Loaned Media of the chosen Reader”, it would be better to have the reader’s name.
- It is possible to loan out the same medium twice without it having been returned.
- It is possible to delete the record for an item on loan quite easily.
- Data can be changed or deleted in the main form. This can be useful for small libraries with little public traffic. However when things become hectic at the loans counter, editing of user data should not take place at the same time as issuing loans. It would be better if new users could be registered but existing user data left untouched. For libraries, this applies equally to deletions or complete name changes.

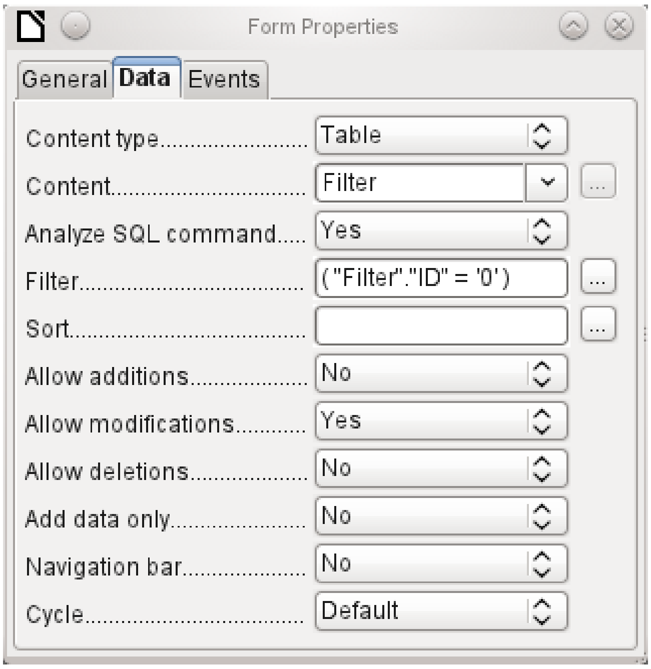
First let us improve the selection of readers. This should protect us from changes to the loan records. A simple solution would be not to allow any modification except the entry of new records. This still requires a search function when a reader wishes to borrow an item. It would be better to use a list box to find the reader and carry out the issue and return operations in separate table controls.

For the main form we need a table, into which the list box can write a value linked to this table. The table requires an integer field and a primary key. It will always contain only one record, so the primary key field ID can safely be declared as Tiny Integer. The following table named Filter should therefore be created.

Table 8: Definition of Filter table

Table name: Filter	
Field name	***Field type***
ID	Tiny Integer, Primary key
Integer	Integer

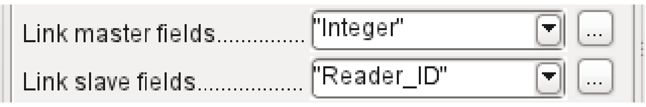
The table is given a primary key with the value 0. This record will be repeatedly read and rewritten by the main form.



Content type	Table
Content	Filter
Analyze SQL command	Yes
Filter	("Filter"."ID" = '0')
Sort	
Allow additions	No
Allow modifications	Yes
Allow deletions	No
Add data only	No
Navigation bar	No
Cycle	Default

Figure 246: Form Properties dialog (Data tab) for main form

The main form is based on the Filter table. It will just read the value from the table which is associated with the primary key (ID) of 0. No data will be added; the current record will just be repeatedly rewritten. As only edits of a single record are allowed, a navigation bar would be superfluous.

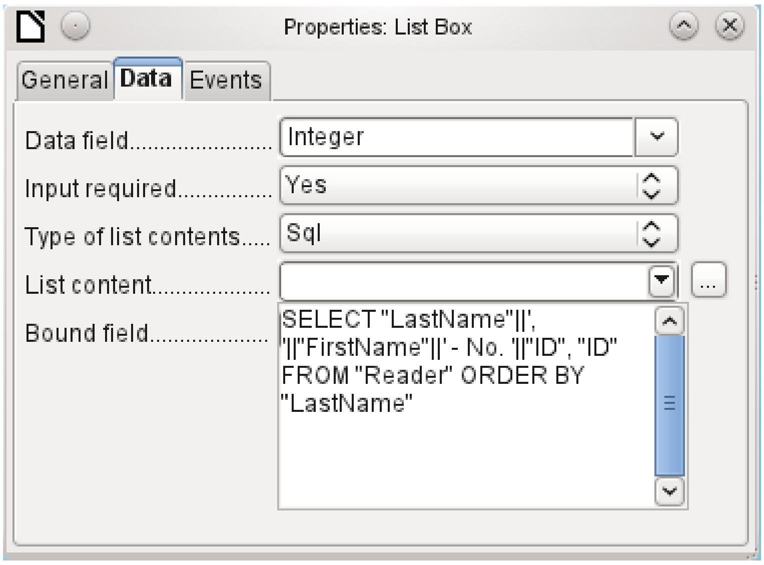


Link master fields	"Integer"
Link slave fields	"Reader_ID"

Figure 247: Integer field in Filter table linked to Reader_ID field in Loan table

This main form is linked to the subform in such a way that the value of the Integer field in the Filter table is the same as the value of the Reader_ID field in the Loan Table. The subform’s properties are unchanged from the version shown above.

Before we create a list box in the main form, we must switch off the wizards. The list box Wizard only allows you to create a box that shows the content of a single field; it would be impossible to have surname and given name and an additional number in the display area of a list box. As in the simpler form, we now enter for the list box contents Surname, Given name – ID Nr. The list box transmits the ID to the underlying table.



Data field: Integer

Input required: Yes

Type of list contents: Sql

List content:

Bound field:

SELECT "LastName"||', '||"FirstName"||' - No. '||"ID", "ID" FROM "Reader" ORDER BY "LastName"

Figure 248: Properties dialog for the list box showing the contents of the List content field on the Data tab

Next to the list box, a button is created. This button is actually part of the subform. It takes over two functions: saving the record in the main form and updating the table in the subform. It is good enough to entrust the update to the button in the subform. The save process for the modified main form is then carried out automatically.

The button can simply be labeled OK. The action assigned to it in the general properties of the button is Refresh Form.

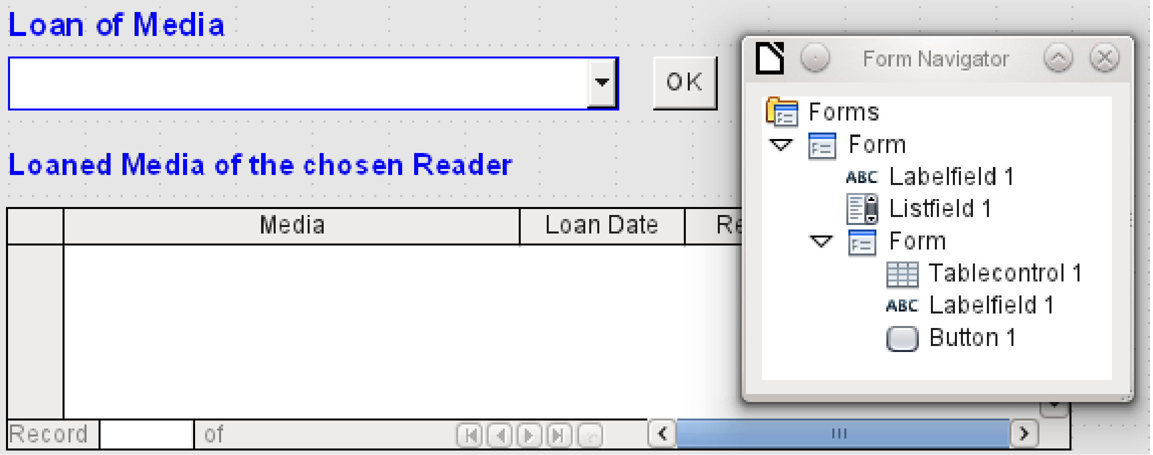


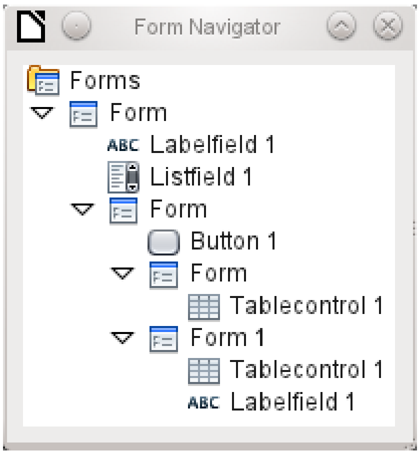
Figure 249: Main form as a filter for a subform

The main form consists only of the heading and the list box; the subform contains another heading, the table control from the previous version and the button.

The form now functions better in that:

- No reader can now be edited, altered or deleted, and
- Readers can be found more quickly by typing into the control than by using a filter.

For a greater degree of functionality (returns without alteration of previous data) a second subform must be created, linked to the same Loans table. To ensure the functionality of the list box in Figure 249, both subforms must be placed one level further down, as subforms of a subform. Data is updated hierarchically from the main form down through the subforms. The button in the previously described form must now be placed in the first subform and not in the two subforms that come under it.



```
Forms
  Form
    Labelfield 1
    Listfield 1
    Form
      Button 1
      Form
        Tablecontrol 1
      Form 1
        Tablecontrol 1
        Labelfield 1
```

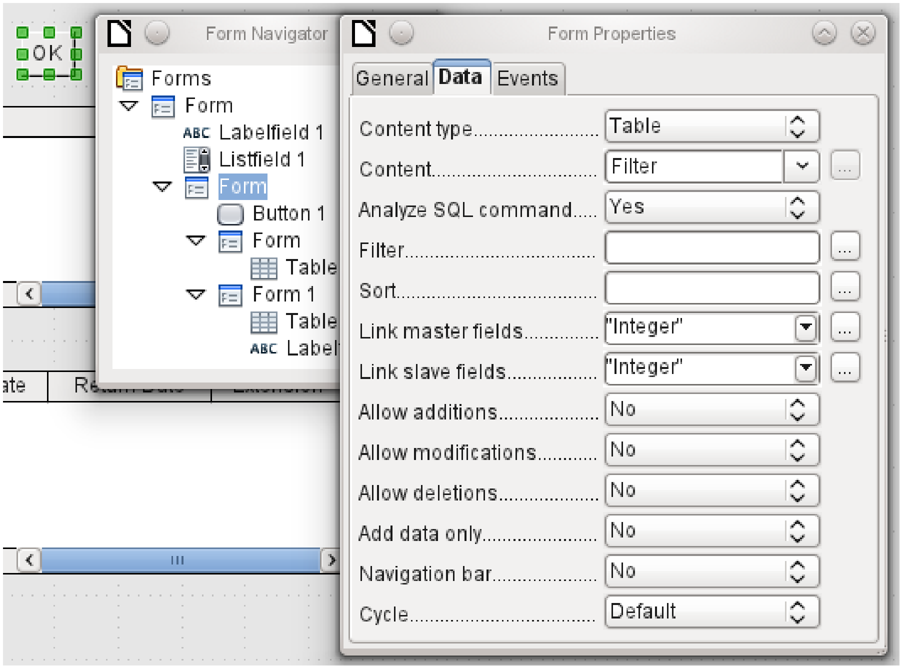
Figure 250: Hierarchy of form and subforms in Form Navigator

Here the Form Navigator is used to show the different levels. In the main form we have the text field for the form title and the list box for finding the reader. The list box appears at the bottom of the form, as it is declared after the subform. This display sequence cannot be altered. The subform has only one button, for updating its contents and at the same time saving the main form. One level further down are the two additional subforms. These are given different names when created so that no confusion arises in any level of the tree.

Note

Basically the names of forms and controls are without significance. However if these names are to be accessed by macros, they must be distinguishable. You cannot distinguish identical names at the same level.

Naturally it makes sense, when creating larger form structures to have meaningful names for forms and their controls. Otherwise finding the right field could quickly become problematic.

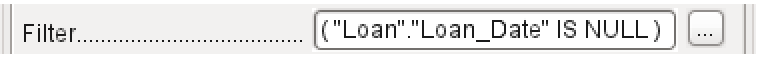


```
Forms
  Form
    Labelfield 1
    Listfield 1
    Form
      Button 1
      Form
        Table
      Form 1
        Table
        Label
```

Content type	Table
Content	Filter
Analyze SQL command	Yes
Filter	
Sort	
Link master fields	"Integer"
Link slave fields	"Integer"
Allow additions	No
Allow modifications	No
Allow deletions	No
Add data only	No
Navigation bar	No
Cycle	Default

Figure 251: Form Properties dialog (Data tab) for a subform, showing many fields set to No

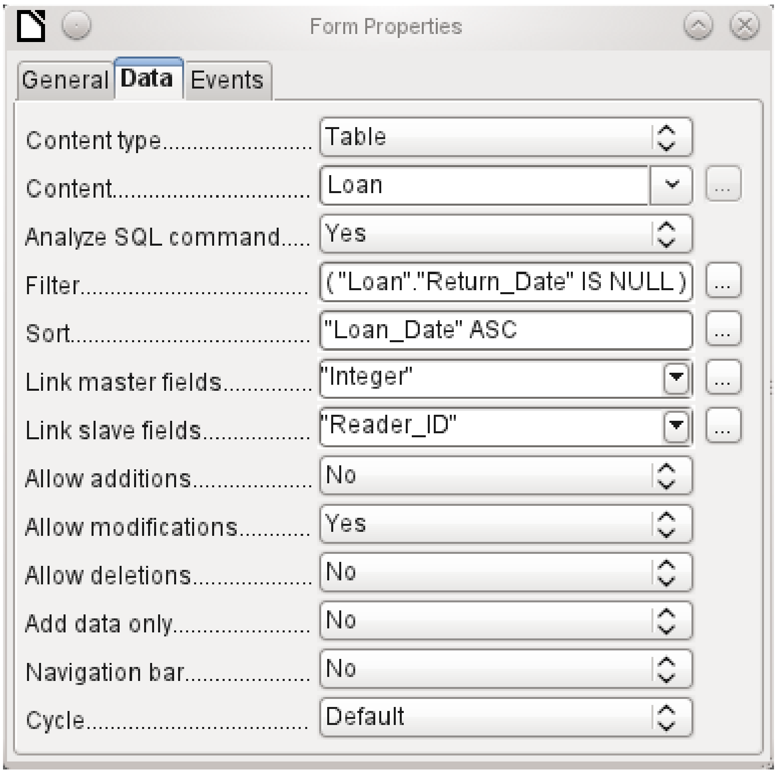
The main form and the subform use the same table. In the subform, no data are entered. That is why all the fields for this form are set to No. The main form and the subform are linked through the field, whose value is to be transmitted to the sub-subforms: the Integer field in the Filter table.



Filter ("Loan"."Loan_Date" IS NULL)

Figure 252: Filter condition for value transmitted to subform

In the first sub-subform, no existing data are displayed; it is used only for creating new data. For this, the suggested filter is adequate. Only records matching the Reader_ID and with an empty loan date field ("Loan_Date" IS NULL) will be displayed. In practice, this means an empty table control. As the table control is not continuously updated, newly loaned media will remain within it until the OK update button is used either to select a new name or to transfer the data into the second sub-subform.



Form Properties

Content type	Table
Content	Loan
Analyze SQL command	Yes
Filter	("Loan"."Return_Date" IS NULL)
Sort	"Loan_Date" ASC
Link master fields	"Integer"
Link slave fields	"Reader_ID"
Allow additions	No
Allow modifications	Yes
Allow deletions	No
Add data only	No
Navigation bar	No
Cycle	Default

Figure 253: Form Properties dialog (Data tab) for second subform

The second sub-subform requires more settings. This form too contains data from the "Loans" table. Here the data is filtered for an empty return date. ("Return_Date" IS NULL). The data are sorted as in the previous form, so that the media on loan for the longest time are immediately visible.

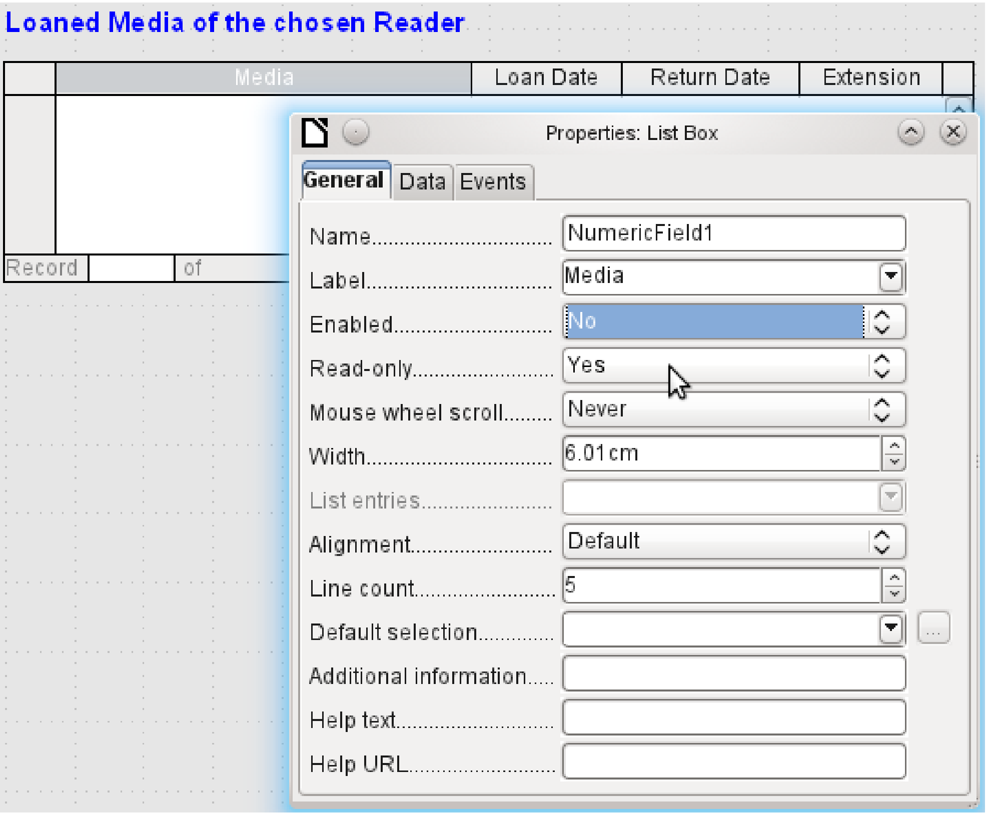
The following points are also important. Old records can be changed, but no new records can be added. Deletion is also impossible. This is the first necessary step to prevent loan records from being simply deleted later on. But it would still be possible to change the medium and the loan date. Therefore the properties of the columns will require adjustment. Eventually the medium and the loan date should be displayed but protected from modification.

The table control is simply duplicated after the creation of the form. This is done by selecting it, copying, deselecting, and then pasting it in from the clipboard. The copy will be at the same position as the original, and will therefore need to be dragged away. After that, both table controls can be edited separately. The table control for the media return can be left practically the same. Only the write access for the Media and Loan date columns need to be changed.

While for the Loan_Date it is only necessary to choose Read only, this is not sufficient for list boxes. This setting does not prevent the list box from being used to make changes. However if Enabled is set to No, a choice cannot be made there. A list box contained within the table control is then displayed as a read-only text field.

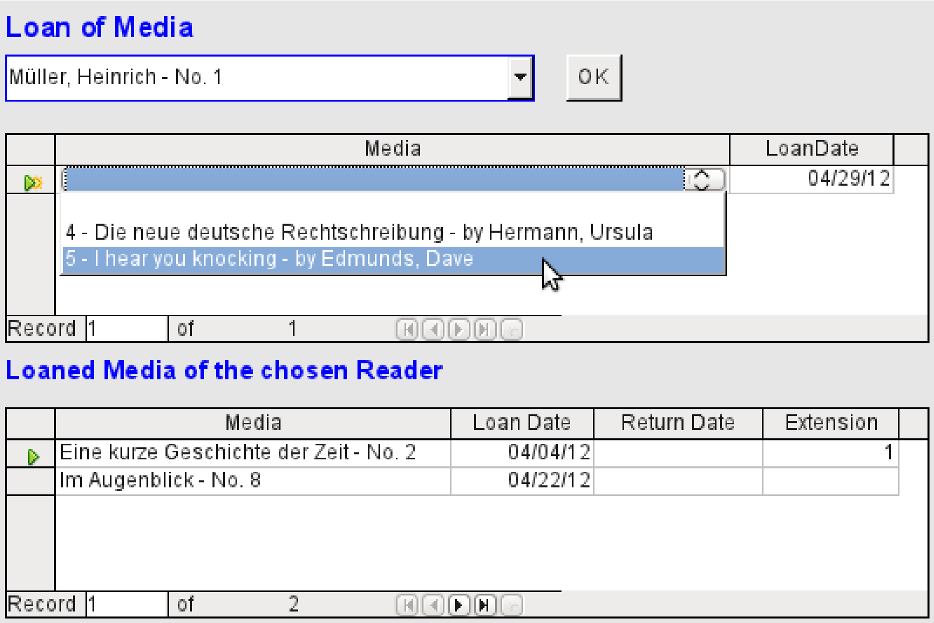
In the above table control, all fields that have nothing to do with the loan are removed. Only the medium as a selection field and the loan date Loan_Date remain.

If finally the query for the list box in the upper table control is selected, only those media are displayed which can actually be loaned out. More about this is in Chapter 5, Queries.



Name	NumericField1
Label	Media
Enabled	No
Read-only	Yes
Mouse wheel scroll	Never
Width	6.01cm
List entries	
Alignment	Default
Line count	5
Default selection	
Additional information	
Help text	
Help URL	

Figure 254: Properties dialog (General tab) for Media column showing Enabled set to No



Loan of Media

Müller, Heinrich - No. 1

OK

	Media	LoanDate
		04/29/12

4 - Die neue deutsche Rechtschreibung - by Hermann, Ursula
5 - I hear you knocking - by Edmunds, Dave

Record 1 of 1

Loaned Media of the chosen Reader

	Media	Loan Date	Return Date	Extension
▷	Eine kurze Geschichte der Zeit - No. 2	04/04/12		1
	Im Augenblick - No. 8	04/22/12		

Record 1 of 2

Figure 255: The selection field in the upper subform shows only media items that are not on loan

The media loan form is already significantly more useful. When a reader arrives at the loan counter, his or her name is searched. The media to be loaned can be selected using the list box and the loan date entered. The Tab key then takes you to the next record.

A final improvement is also desirable: at present the loan date must be selected each time. Imagine a day in the library with perhaps 200 loan transactions, perhaps just one person, who has to loan out about 10 media each time. That would require the same entry for the date field over and over again. There must be a way to simplify this.

Our main form is linked to the Filter table. The main form works only with the record that has as its primary key the "ID" 0. But additional fields can be built into the Filter table. At present there is no field that can store a date, but we can easily create a new field with the field name Date and the field type Date. In the Filter table we now have stored not only the Reader_ID ("Filter"."Integer") but also the Loan_Date ("Filter"."Date").

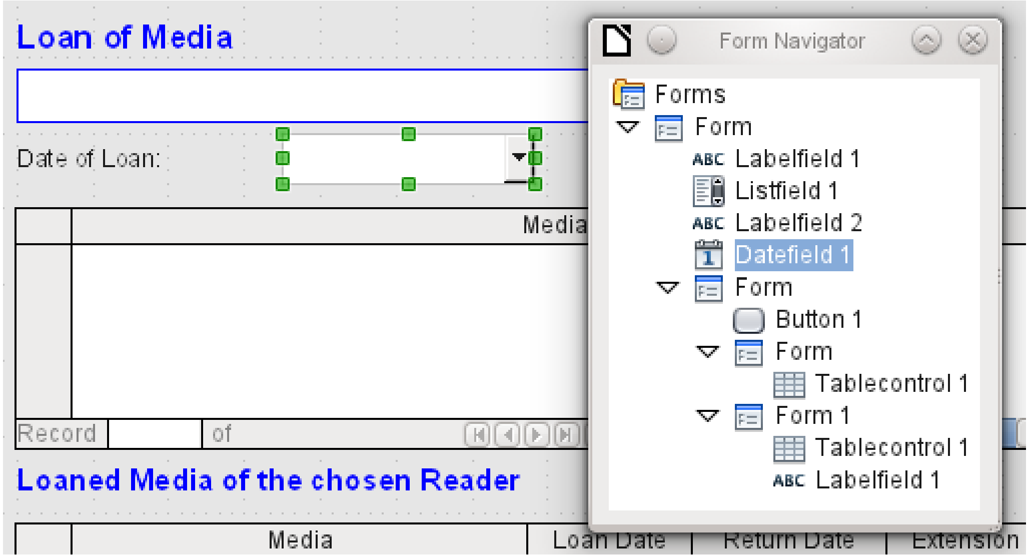
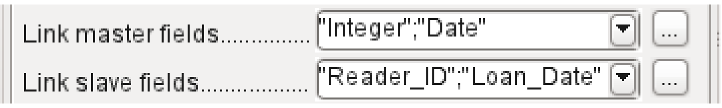


Figure 256: Additional date field in main form

In the main form, an additional date field appears, along with a label referring to its content. The value from the date field is stored in the Filter table and transferred by the linkages from subform to sub-subform.

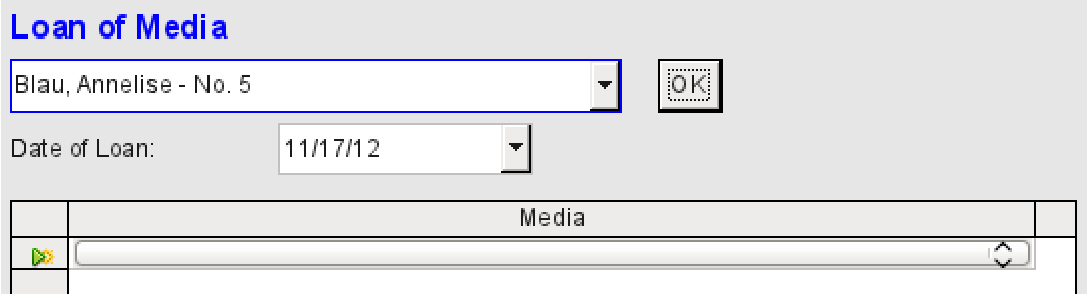


Link master fields............... "Integer";"Date"

Link slave fields................. "Reader_ID";"Loan_Date"

Figure 257: Integer and Date fields in Filter table linked to Reader_ID and Loan_Date fields in Loan table

The linkage between the two forms now refers to two fields. The Integer field is bound to the Reader_ID field of the sub-subform. The Date field is bound to the Loan_Date field. This ensures that the Loan_Date is automatically transferred from the Filter table to the Loans table when the loan is made.



Loan of Media

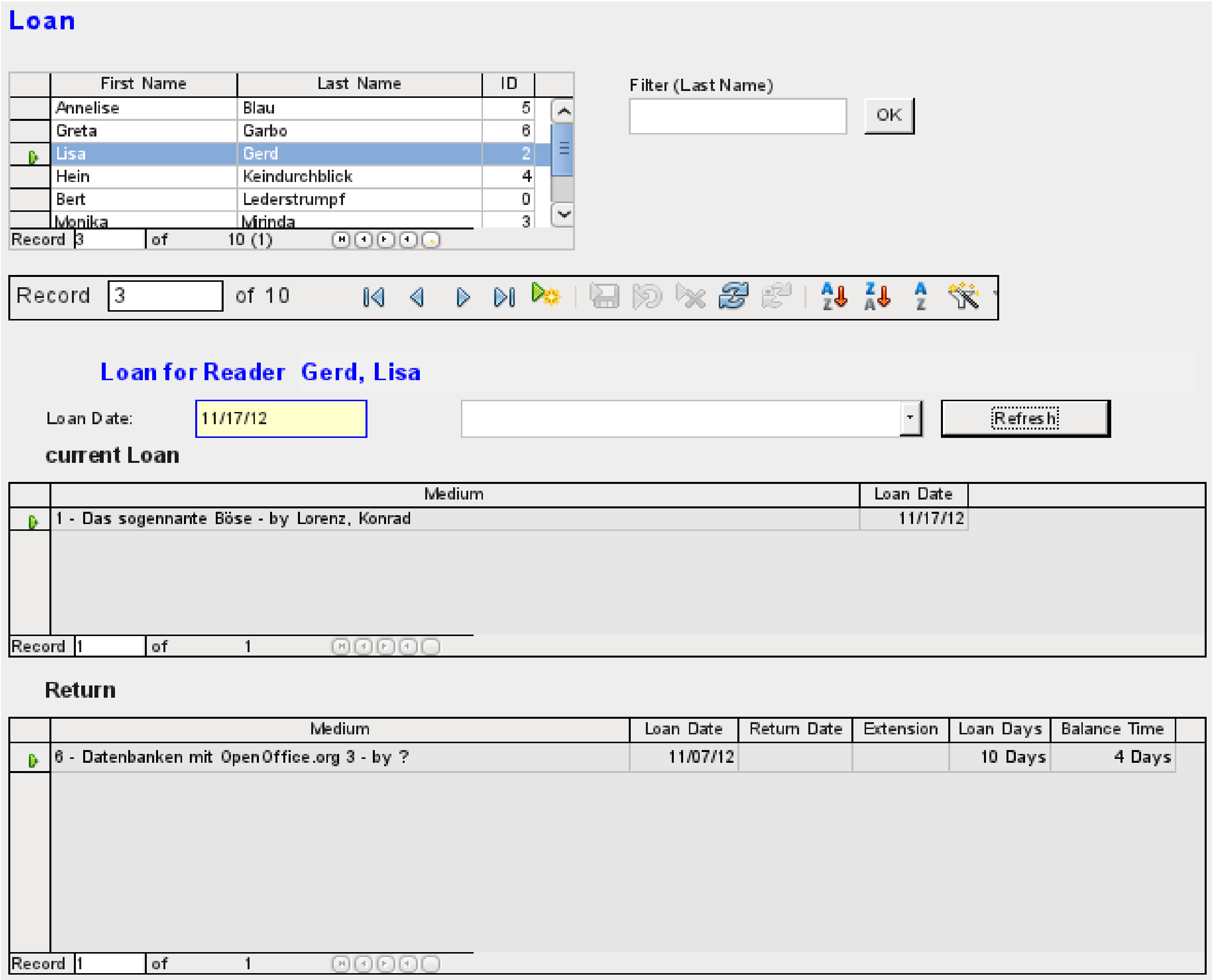
Blau, Annelise - No. 5

Date of Loan: 11/17/12

Figure 258: The date of the loan is entered only once; when the reader changes, it must be entered again

The date field is now removed from the table control, so that the latter contains only one search field. This would be the ideal requirement for speeding up even more the work of the library. For in fact each medium will have a printed identification number, so why does that have to be searched? You can just enter the number directly. Or, better still, it could be scanned in with a barcode reader. Then media can be loaned as rapidly as the borrower can put them in his bag.

This is illustrated in the example database. The above example should suffice for understanding the initial form design but, as the example database, Media_without_Macros.odb, develops the form further, the extra refinements are briefly presented below.



Loan

First Name	Last Name	ID
Annelise	Blau	5
Greta	Garbo	6
Lisa	Gerd	2
Hein	Keindurchblick	4
Bert	Lederstrumpf	0
Monika	Mirinda	3

Record 3 of 10 (1)

Filter (Last Name)

Record 3 of 10

Loan for Reader Gerd, Lisa

Loan Date: 11/17/12

current Loan

Medium	Loan Date
1 - Das sogennante Böse - by Lorenz, Konrad	11/17/12

Record 1 of 1

Return

Medium	Loan Date	Return Date	Extension	Loan Days	Balance Time
6 - Datenbanken mit OpenOffice.org 3 - by ?	11/07/12			10 Days	4 Days

Record 1 of 1

Figure 259: The Loan form

The Loan form shows the following properties:

- Readers are displayed in a table control. Here you can also enter new readers.
- Using a filter, linked to the Filter table, names can be filtered using their initial letter. So, if you enter A, only people whose surname begins with A will be displayed. This filtering is case-independent.
- The subtitle shows again the name of the person to whom the loan is to be made. If a lock has been placed on this borrower, this is also displayed.
- The loan date is set to the current date. This is done in the filter table using SQL such that, when no date is entered, the default value to be stored is the current date.
- Loanable media are selected using a list box. When the Update button is pressed, the loan is transferred to the table control below.
- The table control in the middle serves only to display the actual date of loan for the media. Here it is also possible to correct an error retrospectively by deleting the line.
- In the lower table control, as in the above example, alteration of media and loan dates is not possible. Nor is it possible to delete records.
- Apart from the entry of the return date or, if appropriate, an extension of the loan, this table also displays the number of days for which the medium can be loaned and how many days remain of the current loan period.

- If this remaining time becomes negative, the medium must be returned immediately. The issue is then locked. It becomes possible again only when the medium is returned. After the return, the Update button need only be pressed once.

This form, made by using queries, is significantly more complex in its structure than the previously shown version. You can learn more about the essentials of this in Chapter 5, Queries.

One view – many forms

While the example for the Loans form only involves entries into one table (the Loans table) and additionally allows entry into the subform for new readers, the entry procedure for media is significantly more extensive. In its final form, the media table is surrounded by a total of eight additional tables (see Chapter 3, Tables).

The Subtitle and rel_Media_author tables are linked to subforms of the Media form through an n:1 relationship. By contrast, tables with a 1:n relationship to the Media table should be represented by forms that lie above the Media table. As there are several such tables, their values are entered into the main form using list boxes.

The table of a main form and the table of a subform are in a 1:n relationship, or in some exceptional cases 1:1. Therefore, after long use of the database, the main form usually manages a table which has significantly fewer records in it than the table belonging to the subform.

Multiple main forms cannot include the same subform. Therefore it is not possible to create a form arrangement for many simultaneous 1:n relationships in which the subform has the same content. When there is a 1:n relationship for the table belonging to a form, you can use a list box. Here there are only a few terms available from another table, those whose foreign keys can be entered into the main form's table in this way.

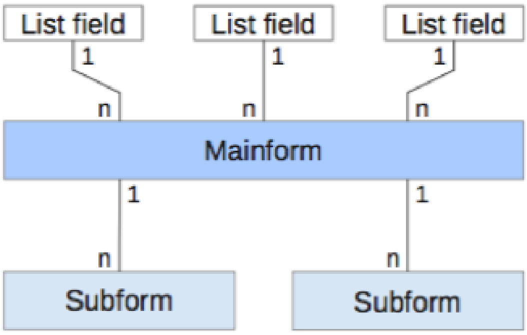


Figure 260: Schematic representation indicating how 1:n and n:1 relationships are handled

Using list boxes, the main form based on the Media table can be assigned values from the Category, Town or Publisher tables. Subforms are used to link the rel_Media_Author and Subtitle tables with the main form and through it with the Media table.

The subform for the rel_Media_Author table again consists of two list boxes so that the foreign keys from the Authors and Author_Add_ID (additions might be for example editor, photographer, and so on) do not have to be entered directly as numbers.

For the media entry form, the list boxes usually have to be filled up gradually during the entry process. For this purpose, further forms are built in alongside the main form. They exist independently of the main form.

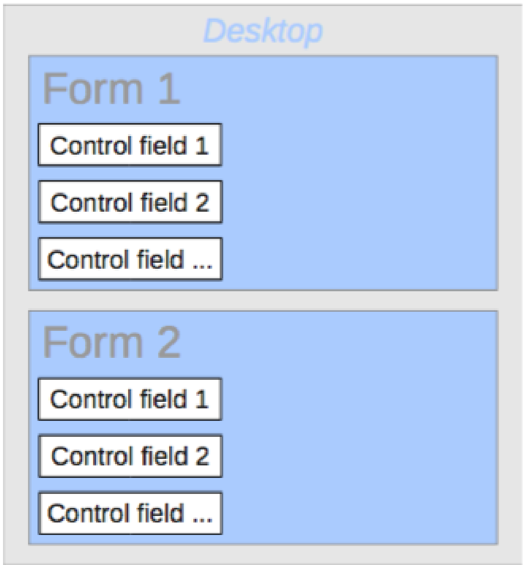
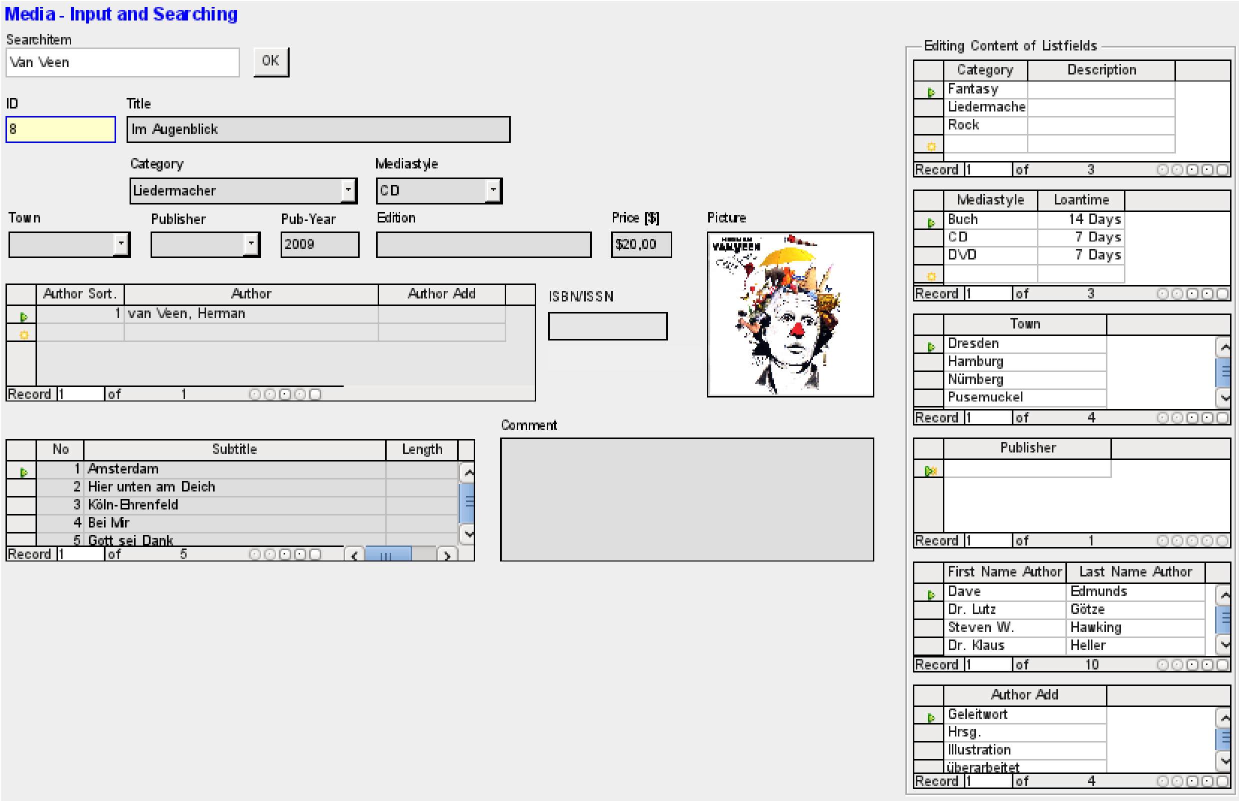


Figure 261: Schematic representation of a multiple form arrangement

The overall form for entering media looks like this:



Media - Input and Searching

Searchitem: Van Veen

ID: 8

Title: Im Augenblick

Category: Liedermacher

Mediastyle: CD

Town:

Publisher:

Pub-Year: 2009

Edition:

Price [$]: $20,00

Picture

	Author Sort.	Author	Author Add
	1	van Veen, Herman	

Record 1 of 1

ISBN/ISSN:

	No	Subtitle	Length
	1	Amsterdam	
	2	Hier unten am Deich	
	3	Köln-Ehrenfeld	
	4	Bei Mir	
	5	Gott sei Dank	

Record 1 of 5

Comment:

Editing Content of Listfields

	Category	Description
	Fantasy	
	Liedermache	
	Rock	

Record 1 of 3

	Mediastyle	Loantime
	Buch	14 Days
	CD	7 Days
	DVD	7 Days

Record 1 of 3

	Town
	Dresden
	Hamburg
	Nürnberg
	Pusemuckel

Record 1 of 4

	Publisher

Record 1 of 1

	First Name Author	Last Name Author
	Dave	Edmunds
	Dr. Lutz	Götze
	Steven W.	Hawking
	Dr. Klaus	Heller

Record 1 of 10

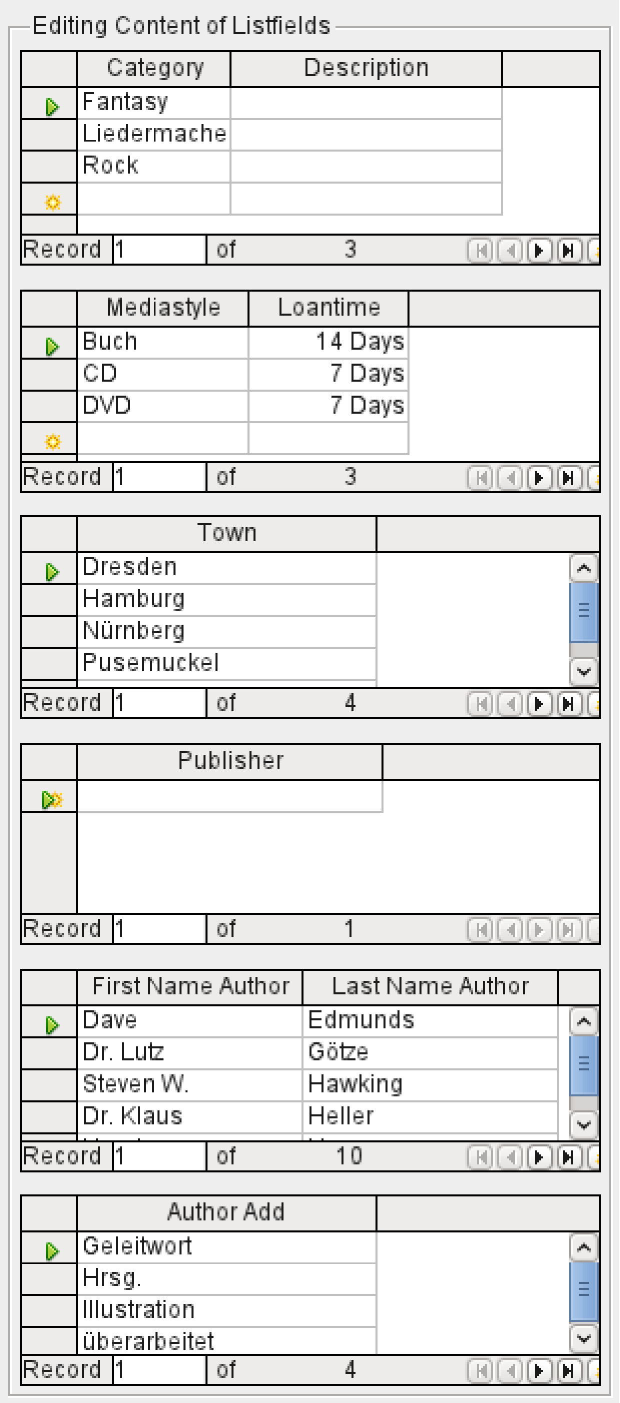
	Author Add
	Geleitwort
	Hrsg.
	Illustration
	überarbeitet

Record 1 of 4

Figure 262: The Media form

On the left side is the main form with a view to searching and entry of new media. On the right side is a group box with the label Editing Contents of List box, providing a separate area intended for filling up the list boxes in the main form. If the database has not existed for long, it will often be necessary to make entries into these fields. However, the more entries that are available for the list boxes of the main form, the less often will access to the table controls in the group box be necessary.

The following table controls are all subordinated as individual side forms to the main form, the entry form.



Editing Content of Listfields

Category	Description
Fantasy	
Liedermache	
Rock	

Record 1 of 3

Mediastyle	Loantime
Buch	14 Days
CD	7 Days
DVD	7 Days

Record 1 of 3

Town
Dresden
Hamburg
Nürnberg
Pusemuckel

Record 1 of 4

Publisher

Record 1 of 1

First Name Author	Last Name Author
Dave	Edmunds
Dr. Lutz	Götze
Steven W.	Hawking
Dr. Klaus	Heller

Record 1 of 10

Author Add
Geleitwort
Hrsg.
Illustration
überarbeitet

Record 1 of 4

Figure 263: Table controls on the right-hand side of the Media form

Here in each case the complete data for a table are entered. In the early stages, it is often necessary to have recourse to these side forms, since not many authors are yet stored in the corresponding table.

When a new record is stored in one of the table controls, it is necessary to find the corresponding list box in the main form and use the Update control (see Navigation bar) to read in the new values.

The Form Navigator shows a correspondingly large list of forms.

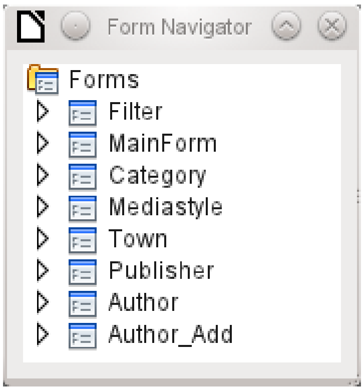
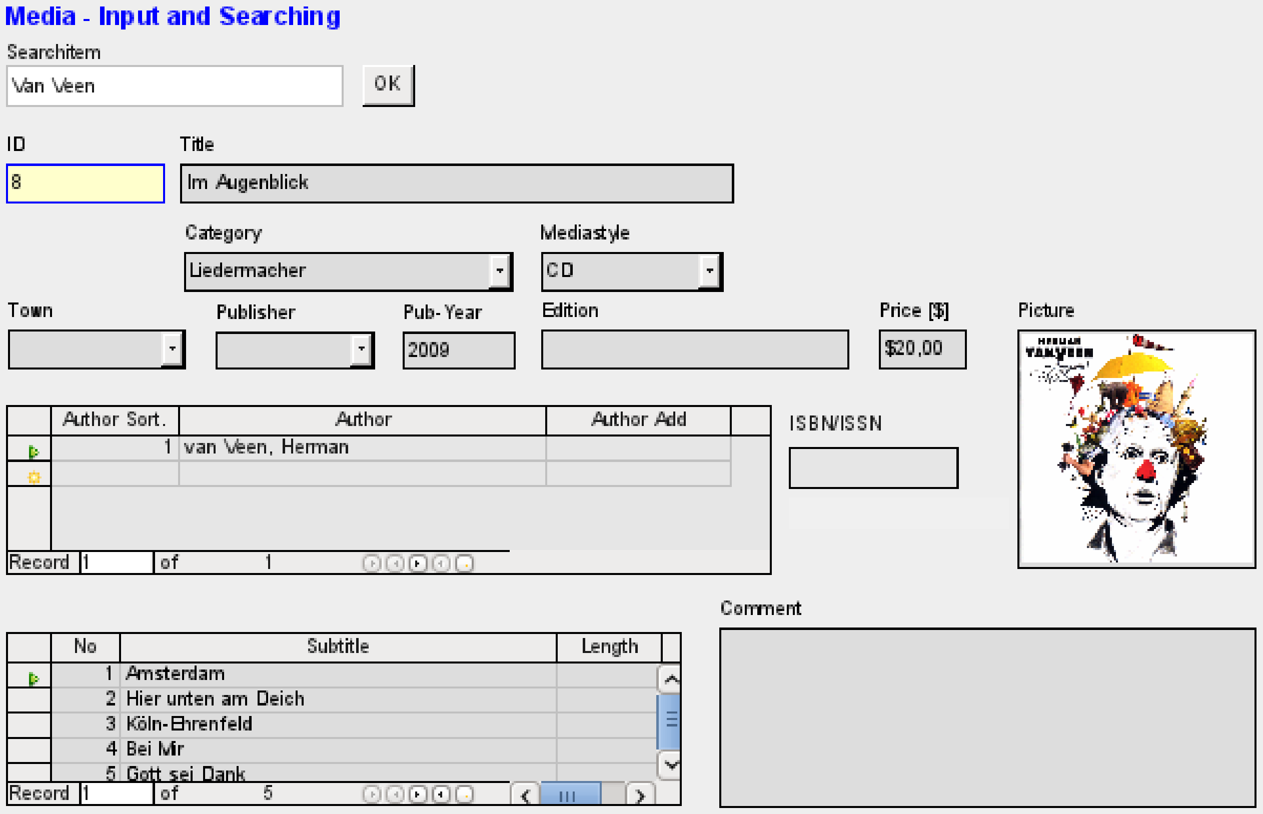


Figure 264: Form Navigator showing the top-level components of the Media form

The forms have been individually named to aid recognition. Only the main form still has the name of MainForm given to it by the Wizard. Altogether there are eight parallel forms. The Filter form hosts a search function while the MainForm form is the main input interface. All the other forms relate to one or another of the table controls shown above.

Without the table controls, the main form looks somewhat simpler.



Media - Input and Searching

Searchitem: Van Veen OK

ID: 8

Title: Im Augenblick

Category: Liedermacher

Mediastyle: CD

Town:

Publisher:

Pub-Year: 2009

Edition:

Price [$]: $20,00

Picture

	Author Sort.	Author	Author Add
▷	1	van Veen, Herman	
☼			

Record 1 of 1

ISBN/ISSN:

Comment:

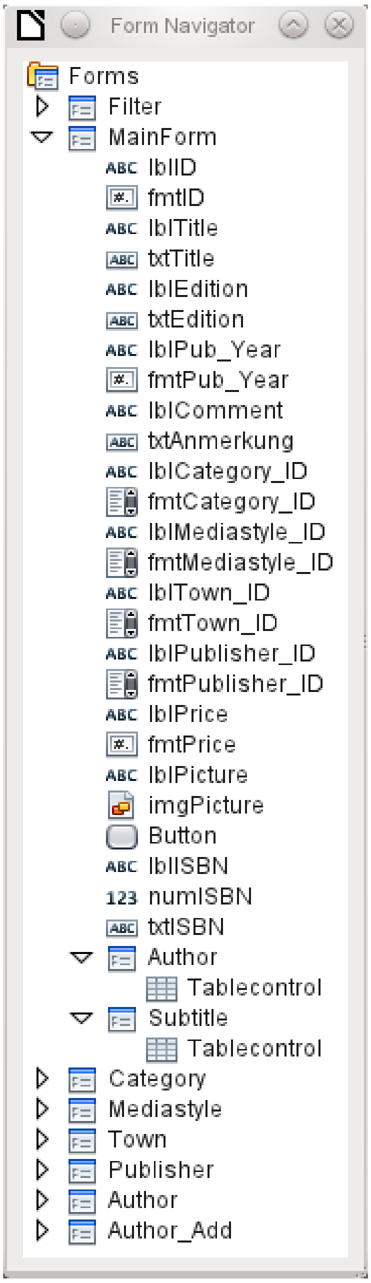
	No	Subtitle	Length
▷	1	Amsterdam	
	2	Hier unten am Deich	
	3	Köln-Ehrenfeld	
	4	Bei Mir	
	5	Gott sei Dank	

Record 1 of 5

Figure 265: Media form without the six table controls situated on the right

The field for the search term lies in the Filter form, the two table controls (for the author and the subtitle) lie in the subform of the Media Entry main form.

In the Form Navigator, this form looks much more complex, as all the controls and labels appear there. In the previous form, most of the fields were actually columns within the table controls and were thus invisible to the Form Navigator.



```
Forms
  Filter
  MainForm
    lblID
    fmtID
    lblTitle
    txtTitle
    lblEdition
    txtEdition
    lblPub_Year
    fmtPub_Year
    lblComment
    txtAnmerkung
    lblCategory_ID
    fmtCategory_ID
    lblMediastyle_ID
    fmtMediastyle_ID
    lblTown_ID
    fmtTown_ID
    lblPublisher_ID
    fmtPublisher_ID
    lblPrice
    fmtPrice
    lblPicture
    imgPicture
    Button
    lblISBN
    numISBN
    txtISBN
    Author
      Tablecontrol
    Subtitle
      Tablecontrol
  Category
  Mediastyle
  Town
  Publisher
  Author
  Author_Add
```

Figure 266: Form Navigator showing the structure of the MainForm form

The sequence within the Form Navigator cannot easily be changed. So, for example, it would seem more sensible to place the subforms Subtitle and Author as branches of MainForm right at the beginning. But within the Form Navigator, individual controls and subforms are simply shown in the order in which they were created.

The Form Wizard provides the controls with particular abbreviations which appear next to the icons and indicate what type of control this is. Labels begin with 'lbl', text fields with 'txt' and so on. Altogether labels provide only secondary information for data entry. They can also be placed directly above text frames and then do not appear in the Form Navigator.

The sequence of the elements in the navigator has nothing to do with the tab sequence. That is determined by the Activation Order.

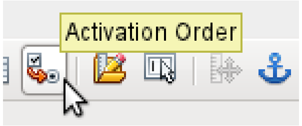
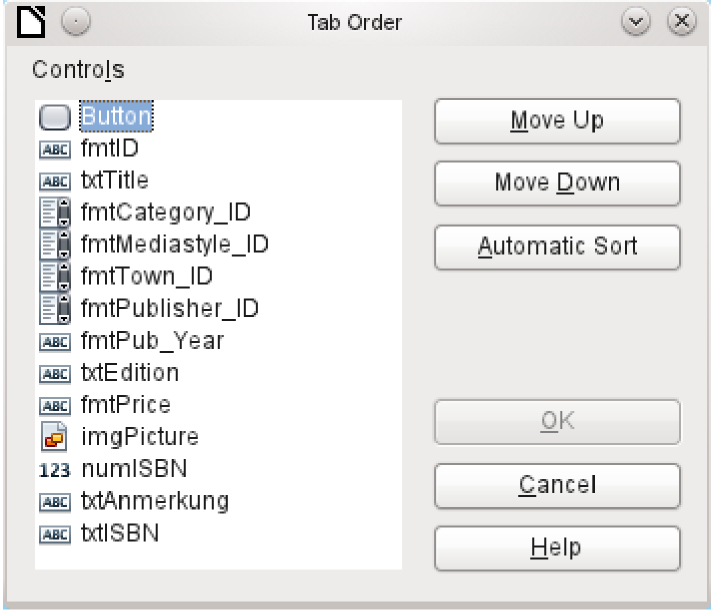


Figure 267: Activation Order icon on Form Design toolbar

The activation sequence for a particular form is invoked when an element of the form is selected and then the Activation sequence button is clicked. For a simple form, it is not necessary to do things in this order, but where there are many parallel forms, the function needs to know which form is intended. Otherwise, by default, it is the first form in the Form Navigator. In the above example, this contains only a single text field.



Button
fmtID
txtTitle
fmtCategory_ID
fmtMediastyle_ID
fmtTown_ID
fmtPublisher_ID
fmtPub_Year
txtEdition
fmtPrice
imgPicture
numISBN
txtAnmerkung
txtISBN

Figure 268: Tab Order dialog

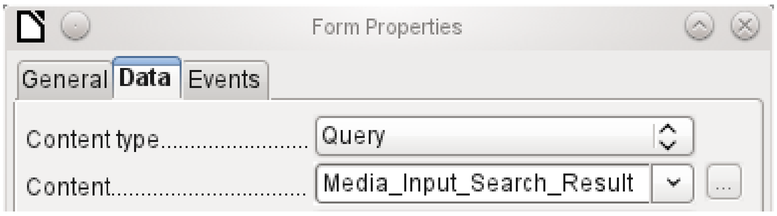
The activation sequence allows all the elements that transmit data to the underlying table of the form or can carry out actions, to be put in order. This corresponds to setting the activation sequence in the control properties listed in section "Default settings for many controls" above.

Note that in the activation sequence, some controls appear for which the tab stop is actually switched off. They are included in the list, but are not in fact accessed through the keyboard when working with the form.

Fields can be automatically sorted in the order in which they appear on the form background. The higher up a field lies, the earlier it comes in the sequence when you use Automatic Sort. For fields at the same height, the leftmost field comes first. This sorting functions without error only when the elements were exactly positioned using the grid when the form was created. Otherwise you will need to adjust them. Simply select a control and use Move Up or Move Down to move it higher or lower in the sequence.

If there is a subform, Automatic Sort jumps directly into the subform after completing the main form. In the case of a table control, this causes the cursor during keyboard input to be trapped within this subform; you can only free it by using the mouse or by pressing Ctrl+Tab.

The Automatic Sort functions only once for a table control. A subsequent subform with a table control will not be included. Parallel forms are not taken into account either. An Automatic Sort cannot be made retrospectively for a subform with a table control. The subform must be completely removed (temporarily moved to another form).



Content type: Query

Content: Media_Input_Search_Result

Figure 269: Form Properties dialog (Data tab) for the MainForm form, showing a query content type

The data substrate for the form for media entry is in this case not a table but a query. This is necessary as the form is to be used not just for entering records but also for searching. The form also contains a text field which shows, after saving, whether the ISBN number entered was correct. This too is only possible using a comprehensive query. To understand the background of this, it is therefore necessary to discuss the fundamentals of queries, covered in Chapter 5.

Error messages during input to forms

Some error messages that commonly occur when a form is first designed are briefly explained here.

Attempt to insert null into a non-nullable column: column: ID table: Tablename in statement …

There are fields which are not allowed to be empty. If they are so defined in the table (*NOT NULL*), you get this error message. If it has also been declared non-null in the form as well, you also see a native language message with the exact name of the field that must be filled in.

The above message appears very often when the primary key, in this case *ID*, has not been imported into the form. It was designed as an automatically incrementing field, but unfortunately the designer forgot to define it as AutoValue. So you must either correct this or the field must appear in the form, so that you can enter a value.

Defining a field retrospectively as AutoValue is sometimes problematical, if the table has relationships with other tables or when the table is accessed using a view. All links must be removed to make the primary key field of the table editable. The content of the SQL command that was used to create the view can be stored temporarily in a query.

Integrity constraint violation - no parent SYS_FK_95 table: Tablename in statement …

Here we have a link between the table underlying the form and another table. This table is supposed to supply its primary key for use in a foreign key field. Normally this is done using a listbox which carries out the correct query for retrieving the key. If you are not using a listbox, or if the listbox was incorrectly constructed, the foreign key field can acquire an incorrect value which is not actually present in the source table as a primary key. That is what is meant by “*Integrity constraint violation"*. *‘no parent SYS_FK_95’* means that in the second table, the source *“Parents“* table, the corresponding index value with the name *'SYS_FK_95'*is not present.

.Errors when entering new records
Errors in running functions

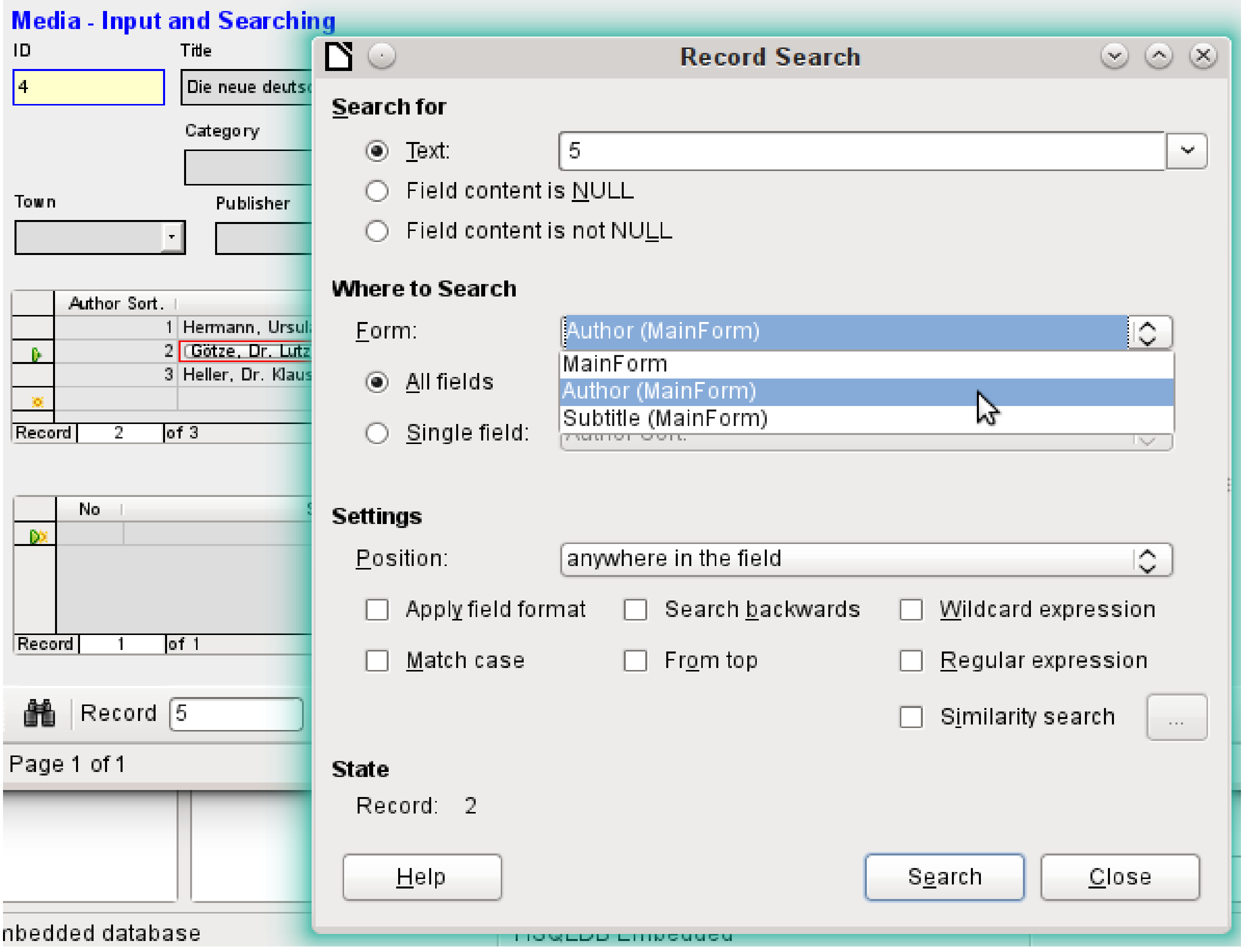
Suppose a form supplies data to a subform. Base does not see this as a change in a field value. The graphical user interface offers to save the value but the record appears empty. The answer is to include a simple field, for example a yes/no field, in the underlying table. Now, before you save, remove this field and the record can be saved without any problem.

Values passed on via other forms are apparently only entered into the corresponding fields when there is at least one additional action in the form.

Searching and filtering in forms using the navigation bar

The form’s navigation bar offers various possibilities for searching and filtering. Individual elements on the navigation bar have already been listed above.

Record searching using parameters



Media - Input and Searching

ID: 4

Title: Die neue deuts

Record Search

Search for

● Text: 5

○ Field content is NULL

○ Field content is not NULL

Where to Search

Form: Author (MainForm)

● All fields

○ Single field:

Settings

Position: anywhere in the field

□ Apply field format □ Search backwards □ Wildcard expression

□ Match case □ From top □ Regular expression

□ Similarity search

State

Record: 2

Figure 270: Searching the Author subform of the MainForm form using the Record Search dialog

A simple record search with parameters is described in Chapter 3, Tables. It offers a considerable number of settings. A characteristic of this type of search is that basically all records are sifted rather than limiting the records displayed to those containing the search parameters.

Record searches in forms allow you to search both the form and any subforms, but it is not possible to search all the forms at once.

The illustration shows access to the subform Form_author in the main form MainForm. This is a listfield containing the names of the authors. However, what is searched is not the text in the listfield but the values that are entered as foreign keys when the field is used.

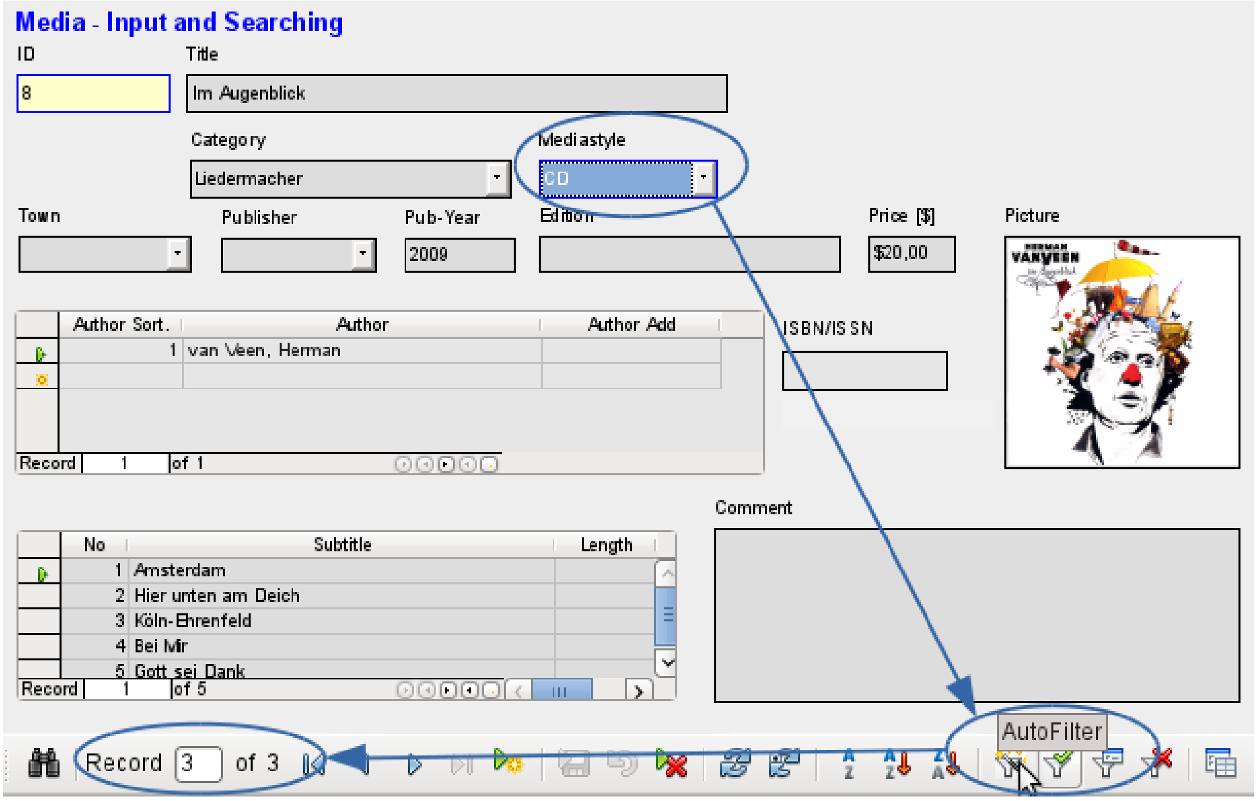
This type of search has the following shortcomings:

- The parameter search is too complicated for normal use.
- The search function itself is very slow as it does not use the query functions of the database.
- The search only works for one form at a time, although for all fields. Subforms must be separately searched.

- We are searching a subform that belongs to the current main form. Entries into other subforms will not be revealed. So, for example, it is not possible to find a subtitle of any medium other than the one currently displayed.
- The search applied to listfield functions on the basis of key fields (foreign keys) stored in the table. The displayed data cannot be used.

Filtering with the autofilter

The Autofilter can be directly selected in the navigation bar. Click on a field in the main form and it will filter on the content of that field. The autofilter functions with list fields too, and has an obvious advantage over the entry of foreign keys by hand.



Media - Input and Searching

ID: 8

Title: Im Augenblick

Category: Liedermacher

Mediastyle: CD

Town:

Publisher:

Pub-Year: 2009

Edition:

Price [$]: $20,00

Picture

Author Sort.	Author	Author Add
1	van Veen, Herman	

Record 1 of 1

ISBN/ISSN:

Comment:

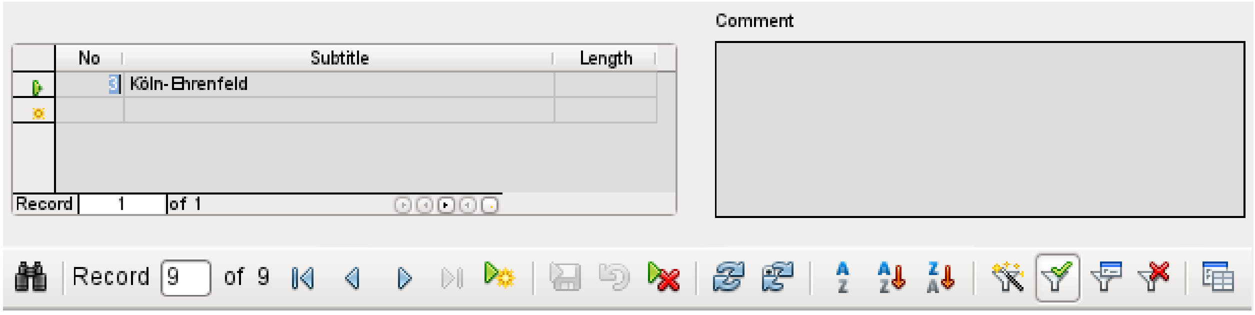
No	Subtitle	Length
1	Amsterdam	
2	Hier unten am Deich	
3	Köln-Ehrenfeld	
4	Bei Mir	
5	Gott sei Dank	

Record 1 of 5

Record 3 of 3

Figure 271: Using the autofilter

In the example shown above, a record is retrieved that has ‘CD’ in the Mediastyle field. Then the autofilter button is toggled. Of the original 9 records in the Media table, only 3 records now show in the record counter.



No	Subtitle	Length
3	Köln-Ehrenfeld	

Record 1 of 1

Comment:

Record 9 of 9

Figure 272: Output of the autofilter

Although only two records have a ‘3’ in the No. field, the main forms for all the other records continue to be displayed. In the subform, only records that match ‘3’ are displayed.

However using the autofilter cannot solve the following problems:

- If you search in a subform, only the specified value will be shown in it, but the main form is shown for all the records even if they don’t contain that value. So if, for example, you are looking for a medium with a second author, you will still see all media that have only one author. In these records, the author field will be empty.
- The value used for the filter must be specific. There is no possibility of a similarity search to find related materials.

Filtering with the form-based filter

The form-based filter offers a form for inserting actual values to be used for filtering rather than just a filter button. Here you can filter for several values at the same time. These values can either be combined as a joint condition (AND) or as alternatives (OR).

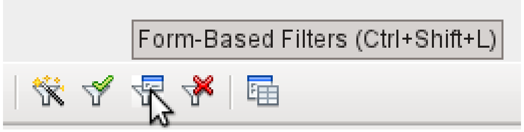
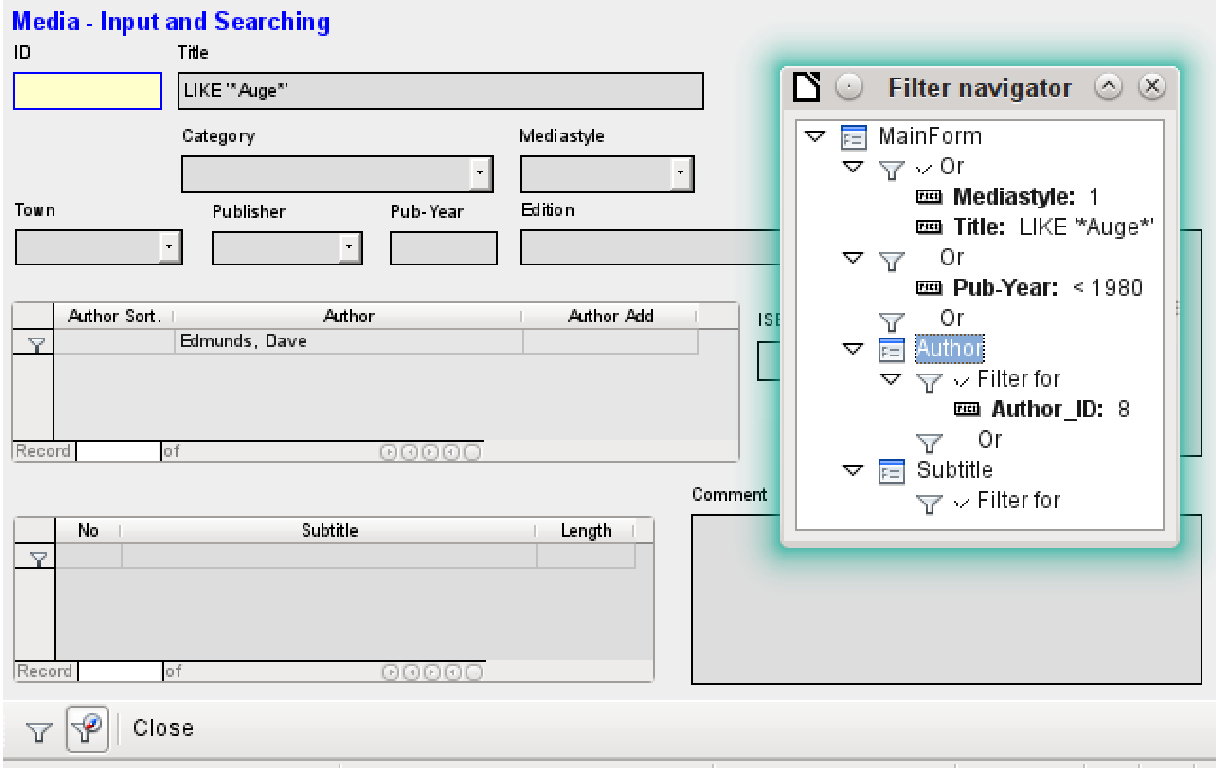


Figure 273: Form-Based Filters icon on Form Navigation toolbar

The filter values entered need not be unique. Here therefore you can formulate conditions like those described in the filtering of tables. So `LIKE'%eye%'` entered in the media title field will yield all records for which the title contains the word ‘eye’.

Filtering of listbox fields is carried out by direct selection of displayed list field contents. You do not need to input the underlying foreign keys.



Title: LIKE '*Auge*'

Author: Edmunds, Dave

```
MainForm
  Or
    Mediastyle: 1
    Title: LIKE '*Auge*'
  Or
    Pub-Year: < 1980
  Or
  Author
    Filter for
      Author_ID: 8
    Or
  Subtitle
    Filter for
```

Figure 274: Setting up a form-based filter

Here we search for all records with the mediastyle ‘CD’ (corresponding to a foreign key value of ‘1’) and also have somewhere in the title the character sequence ‘eye’. When you enter `LIKE '*eye*'`, the GUI replaces the usual SQL wildcard symbols “%“ with the “*” character, a more familiar wildcard symbol for most users.

We also add the condition that all records be displayed for which the year of publication is “1980”.

The author ‘Edmunds, Dave’ is selected from the listbox in the subform. The corresponding foreign key value is ‘8’.

During entry, the form-based filter shows only the fields that show a tick before OR. In this way several conditions can be set for the field. However, it causes the filter to suppress the display of the Mediastyle list field contents, whereas this works in the table control of the subform.

During the creation of the filter, you do not see the navigation bar; instead the choices for the form-based filter are displayed. You filter according to your requirements, insert the filter values, and then simply close the display.

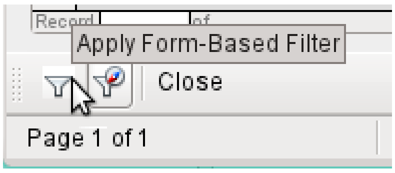
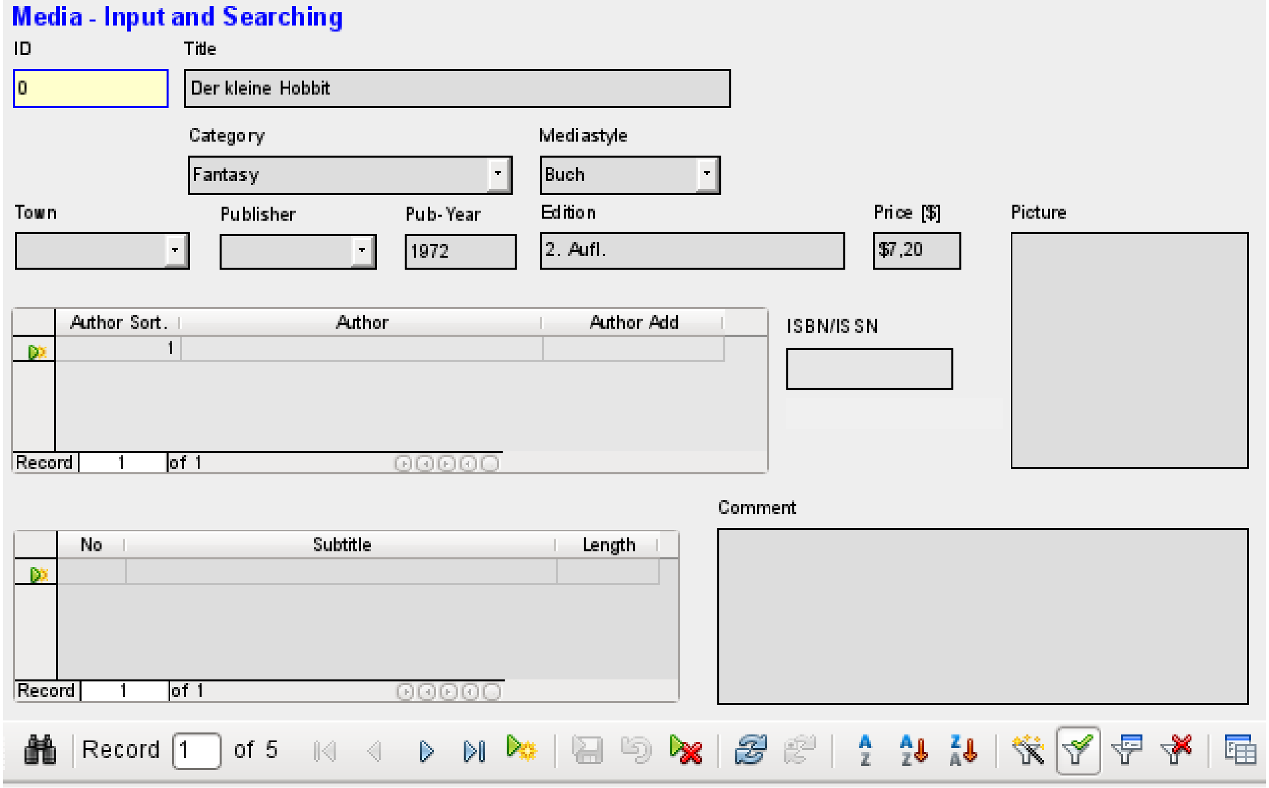


Figure 275: Apply Form-Based Filter icon on Form Filter toolbar

The filter is applied and the form is filtered according to your specification.


Media - Input and Searching

ID: 0

Title: Der kleine Hobbit

Category: Fantasy

Mediastyle: Buch

Town:

Publisher:

Pub-Year: 1972

Edition: 2. Aufl.

Price [$]: $7,20

Picture

Author Sort.	Author	Author Add
1		

Record 1 of 1

ISBN/ISSN:

No	Subtitle	Length

Record 1 of 1

Comment:

Record 1 of 5

Figure 276: Results of the form-based filter

When filtering is on, the number of records is limited to those that match. In this case there are five matches. In the illustration, neither the title nor the media type match; the condition that the year of publication be less than 1980 provides the match.

The subform shows something interesting. As this too is to be filtered, the author of the book “The Little Hobbit” is not shown. The filter value is linked to the author Dave Edmunds. This again shows the logic behind the filtering: it uses the filter properties of the current form. Subforms are separately filtered and a filtering of the subform does not affect the main form – and vice versa.

This fairly successful filter still has the following disadvantages:

- The filtering again functions only in the main form. In the subform, as in the case of Autofilter, values that fail to match are not displayed.. This has no effect on the display of the main form. Therefore in the main form, content is shown which cannot generate any filtered content for the subform.
- You need detailed knowledge of how to set conditions if you are searching for something more complicated than complete fields. Most users, even if they are accustomed to using search engines, lack the necessary skills.

Filtering with the default filter

The default filter can be reached by requesting the display of the data source as a table. Then the procedures are the same as when using the default filter in a table.

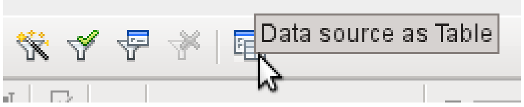
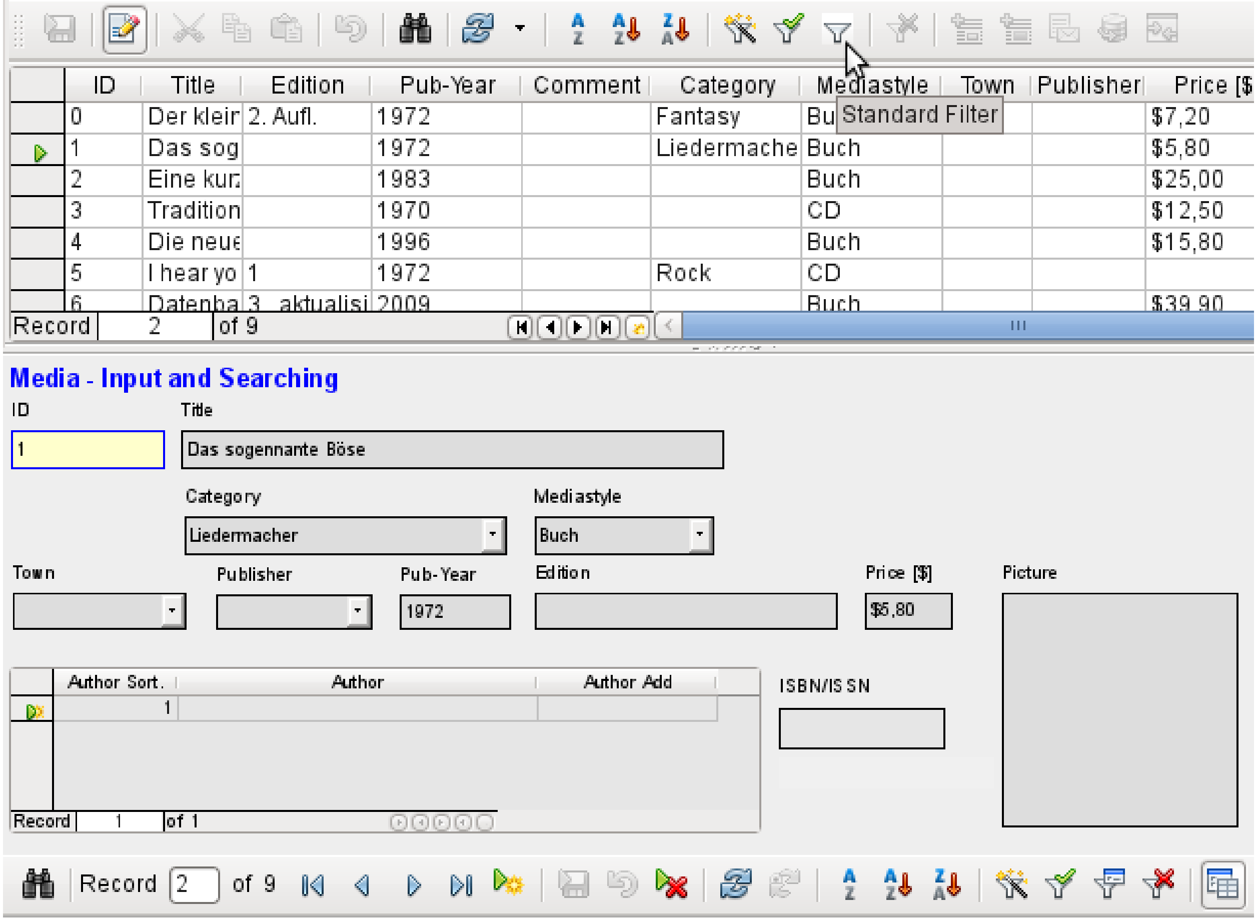


Figure 277: Data source as Table icon on Form Navigation toolbar



	ID	Title	Edition	Pub-Year	Comment	Category	Mediastyle	Town	Publisher	Price [$
	0	Der kleir	2. Aufl.	1972		Fantasy	Bu			$7,20
▷	1	Das sog		1972		Liedermache	Buch			$5,80
	2	Eine kur:		1983			Buch			$25,00
	3	Tradition		1970			CD			$12,50
	4	Die neue		1996			Buch			$15,80
	5	I hear yo	1	1972		Rock	CD			
	6	Datenba	3. aktualisi	2009			Buch			$39,90

Record 2 of 9

Media - Input and Searching

ID: 1

Title: Das sogennante Böse

Category: Liedermacher

Mediastyle: Buch

Town:

Publisher:

Pub-Year: 1972

Edition:

Price [$]: $5,80

Picture

	Author Sort.	Author	Author Add
▷	1		

Record 1 of 1

ISBN/ISSN:

Record 2 of 9

Figure 278: Applying a standard filter

At the right of the Form navigator, you can see a **Data source as Table** button. In table view, the default filter becomes available. Unlike the view of the actual Media table, foreign keys are shown, not with their actual values, but with the content that corresponds to it. A click on the Mediastyle field shows that this is the source of the list field that appears in the form.

Note

The above view highlights a bug because the 13-digit ISBN number is not correctly formatted. The corresponding field is given instead the minimum possible number of digits. If a number has more than 9 digits, take care: editing can lead to loss of data (Bug 82411).

This bug also shows up in table controls. They are shown correctly if you use a formatted field instead of a numeric field.

The table view of a form has the following disadvantages for this search method:

- The search only works within the one table or query that underlies the form and not in a subform that belongs to it.
- In the data view, list fields are shown but you can’t use them for filtering. Once again you need to know the actual foreign key values for those fields. So for example, if the media type is “Book” and this has the primary key value “1”, then the Media table has “1” stored in that field. You must search for this value, even though the composite view shows the value “Book“.

Summary

The search and filter functions are of limited use for forms. A search is much too slow and does not limit the total number of records to those that truly match. The filters only work within one form at a time. If you want to filter in a subform, only the subform is affected. You simply cannot get the correct number of matching records.

Therefore in the forms of the example database, a user-friendly search function has been integrated into the form, a procedure that needs to be worked out by hand and requires some knowledge of SQL.

Record input and navigation

If a form has been so designed that opening it activates the control focus, the cursor will appear in the first input field. Instead of using the mouse to move from field to field, you use the tab key to move to the next field in the sequence.

Sometimes a form allows you to jump to another field, either with the mouse or with a shortcut.

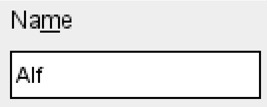


Figure 279: Label field showing underlined “m” character

Using **Properties: Text Box > Label field**, a label field is assigned. The label of this label field is changed from “Name” to “Na~me”. In the form, the letter “m” of the label is now shown underlined. Now you can jump to this text field instantly by using the shortcut *Alt+m.* This only works if the cursor is already in a field of the form.

In principle, any letter can be used as a shortcut, as the jump takes place entirely within the form. However if the cursor is not initially in a form control, the shortcuts activate the LibreOffice user interface instead. So, for example, the shortcut “a” would not jump to the Name field but would instead open the Table menu.

Field jumps only work within a single form, not structures that involve subforms. A jump from a subform back into the main form is not currently possible.

When the cursor is in a table control, it should jump out when you use *Ctrl+Tab*. Sometimes it happens that this key combination does not have the desired effect.

Note

See also the example database Example_cursorjump_subform_mainform.odb

So here is an alternative:

A button is created in the subform. Its title is “~New”. The tilde character is used, as described above, so that *Alt+N* can be used to activate the button. The Additional information field for the button names the field in the main form to which the cursor should be transferred on activation. The button is bound to the following macro using **Properties > Events > Execute action**:

```
SUB JumpToMainform(oEvent AS OBJECT)
   DIM oField AS OBJECT
   DIM oForm AS OBJECT
   DIM oDoc AS OBJECT
   DIM oController AS OBJECT
   DIM oView AS OBJECT
   DIM stShortcut AS STRING
   oField = oEvent.Source.Model
   stShortcut = Mid(oField.Label, InStr(oField.Label, "~") + 1, 1)
   oForm = oField.Parent.Parent
   SELECT CASE stShortcut
      CASE "n"
         oForm.MoveToInsertRow()
      CASE "a"
         IF oForm.isLast() THEN
            oForm.MoveToInsertRow()
         ELSE
            oForm.Next()
         END IF
   END SELECT
   oDoc = thisComponent
   oController = oDoc.getCurrentController()
   oView = oController.getControl(oForm.getByname(oField.Tag))
   oView.setFocus
END SUB
```

The macro can be used just to jump back into the main form. It can also be used with the above settings to create a new record immediately or, using another button on the navigation bar for Next record, to jump to the next record. If the cursor in the main form is on the last record, the jump will lead to a new record. For more on macros, see Chapter 9.

Jumping to a button requires the button to be visible. However it can be quite small or even have a width and height of 0 cm. If the width and height are not defined, it can happen that the button appears as a line running across the entire screen.

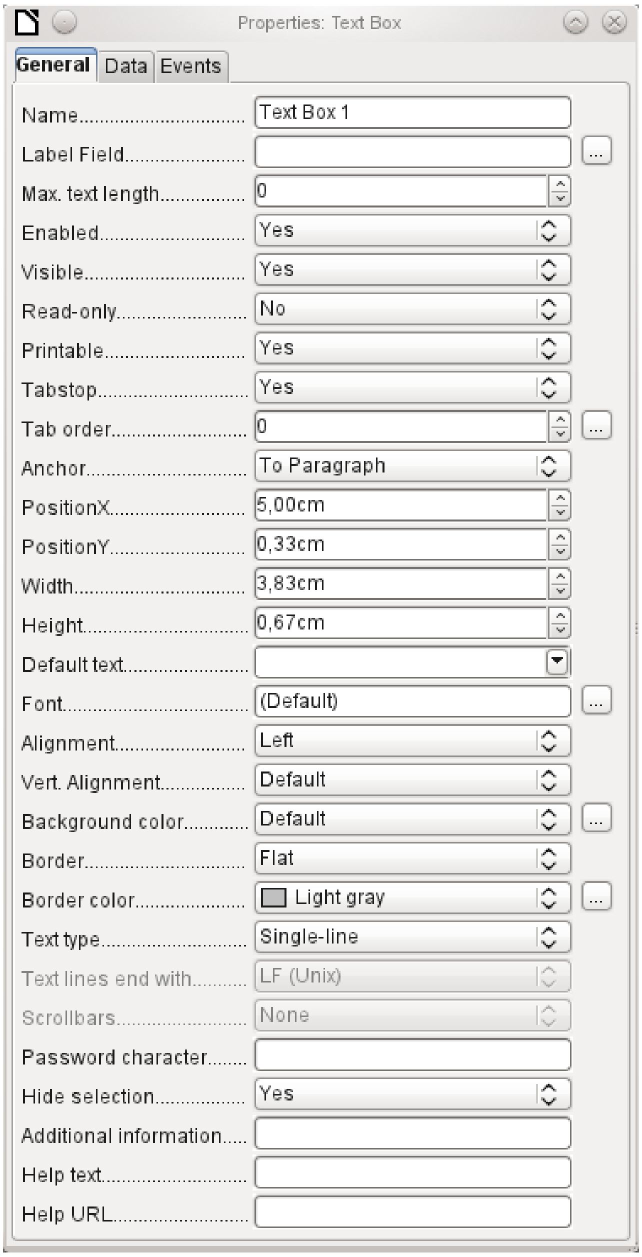
Printing from forms

Forms can be so constructed so that a direct printout of the form is possible. To get usable results, you must take care not to place elements outside the printable area. Choose **View > Normal**. Properties for the page can then be set. A colored background is not an advantage here.

Any individual element can be excluded from printing using **Properties > General > Printable**.

If the database is registered in LibreOffice (using **Tools > Options > LibreOffice Base > Database**, or directly as part of the creation process), the form can be used for mail merges. The form is opened for editing. The data sources are made accessible using **View > Data Sources**, or pressing *F4*. Database fields can now be dragged into the form by their table headers. Then the form is saved. If the same form is now opened for input, LibreOffice recognizes that these are fields for a mail merge and asks if you want to print the letters.

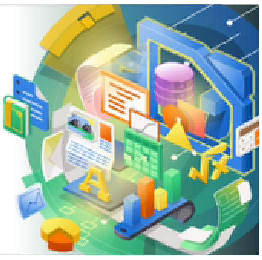
Details of how to do a mail merge are contained in Chapter 7, Linking to Databases.



Name	Text Box 1
Label Field	
Max. text length	0
Enabled	Yes
Visible	Yes
Read-only	No
Printable	Yes
Tabstop	Yes
Tab order	0
Anchor	To Paragraph
PositionX	5,00cm
PositionY	0,33cm
Width	3,83cm
Height	0,67cm
Default text	
Font	(Default)
Alignment	Left
Vert. Alignment	Default
Background color	Default
Border	Flat
Border color	▨ Light gray
Text type	Single-line
Text lines end with	LF (Unix)
Scrollbars	None
Password character	
Hide selection	Yes
Additional information	
Help text	
Help URL	

Figure 280: Properties dialog (General tab) for a text box, showing the Printable property set to Yes

LibreOffice Community



Base Guide 7.3

Chapter 5 Queries

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.

Base uses SQL (Structured Query Language), pronounced "sequel", which is similar in the supported databases. 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 explain the direct creation of queries in Design View.

In this chapter we continue to use the example database, Media_without_Macros.odb. 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.

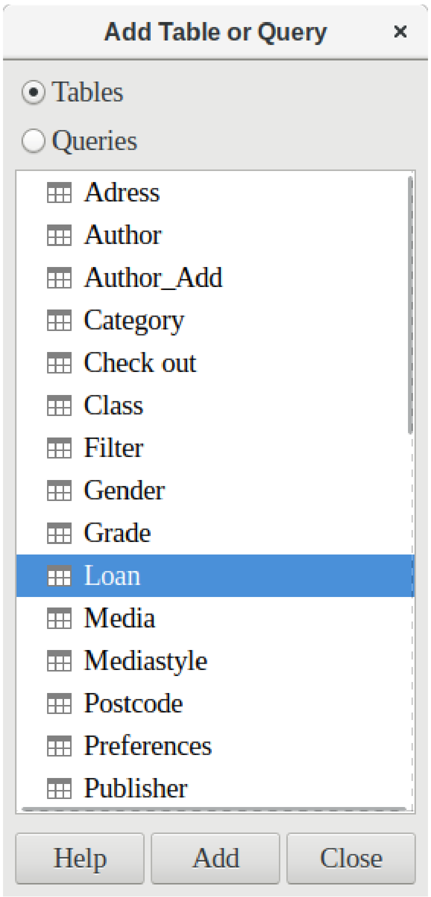
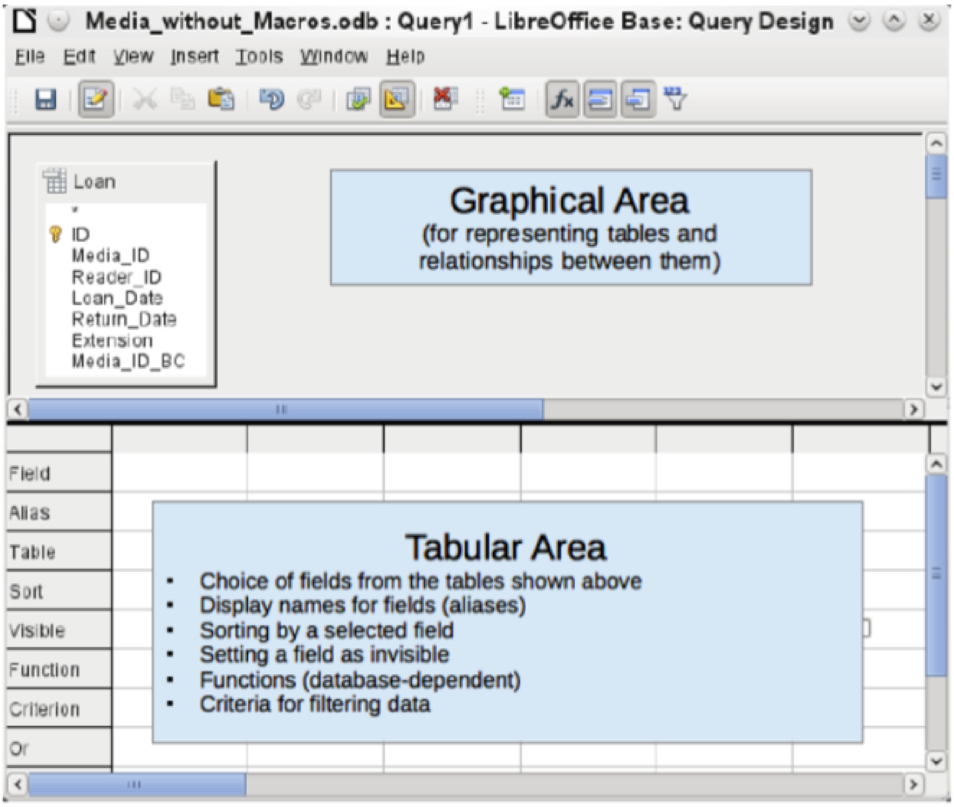


Figure 281: Add Table or Query dialog

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 (Figure 282).

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.



Graphical Area

(for representing tables and relationships between them)

Tabular Area

- Choice of fields from the tables shown above
- Display names for fields (aliases)
- Sorting by a selected field
- Setting a field as invisible
- Functions (database-dependent)
- Criteria for filtering data

Figure 282: Areas of the Query Design window

Figure 282 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 tabular 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 tabular area to reveal a down arrow as shown in Figure 283. 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.

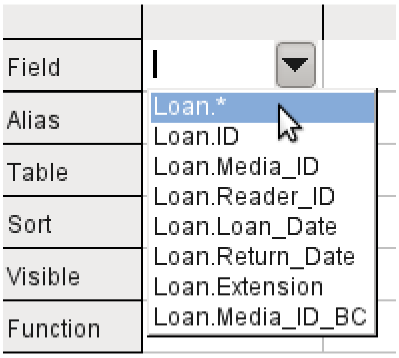


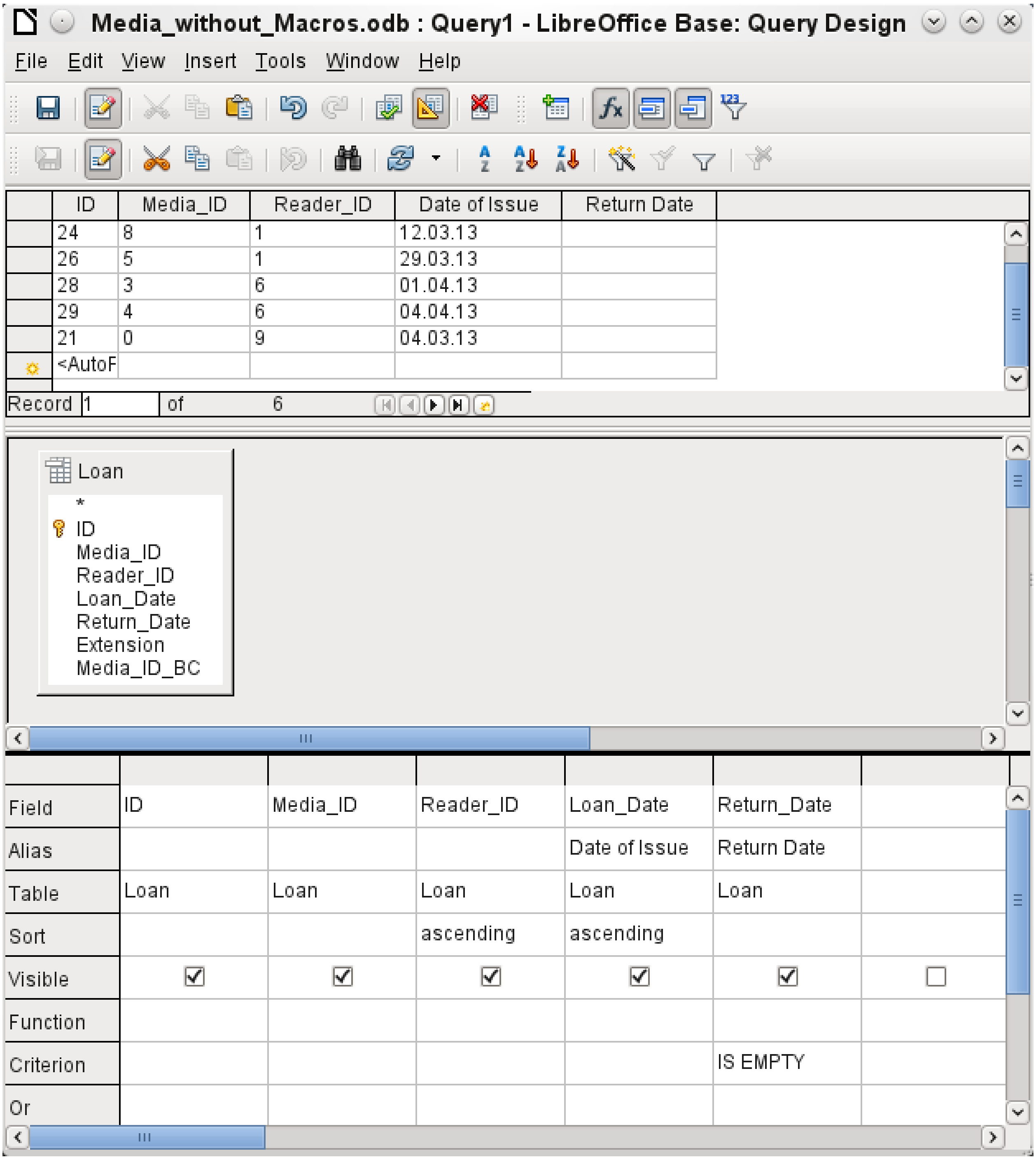
Figure 283: Drop-down list of available fields

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.

Tip

For a quick transfer of all the fields in a table into a query, just click the table view in the graphical interface (Loan.* above).

A double-click on a field inserts that field into the tabular area at the next free position.



Media_without_Macros.odb : Query1 - LibreOffice Base: Query Design
File Edit View Insert Tools Window Help

ID	Media_ID	Reader_ID	Date of Issue	Return Date
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	
<AutoF				

Record 1 of 6

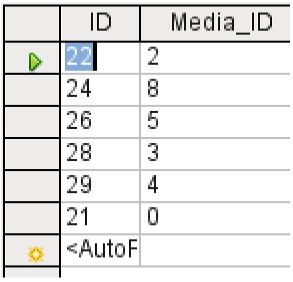
Loan
*
ID
Media_ID
Reader_ID
Loan_Date
Return_Date
Extension
Media_ID_BC

Field	ID	Media_ID	Reader_ID	Loan_Date	Return_Date	
Alias				Date of Issue	Return Date	
Table	Loan	Loan	Loan	Loan	Loan	
Sort			ascending	ascending		
Visible	☑	☑	☑	☑	☑	□
Function						
Criterion					IS EMPTY	
Or						

Figure 284: Query Design window showing preview area containing results of query

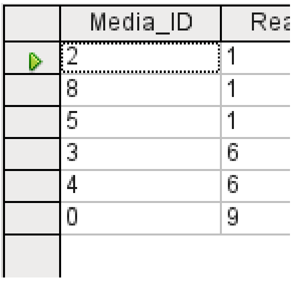
The first five fields of the Loan table are selected as shown in Figure 284. Queries in Design Mode can always be run as tests by clicking Run Query. This causes a tabular view of the data to appear above the graphical view of the Loan table with its list of fields. A test run of a query is always useful before saving it, to clarify 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 285: An editable query



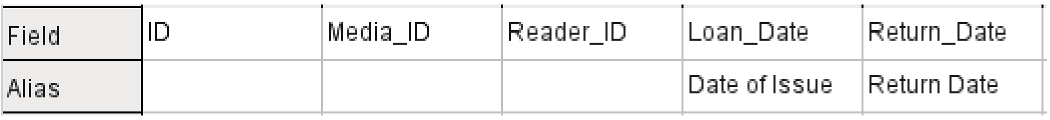
	Media_ID	Rea
▷	2	1
	8	1
	5	1
	3	6
	4	6
	0	9

Figure 286: A non-editable query

In the above test, pay special attention 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 285 is highlighted, the corresponding field in Figure 286 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 285 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

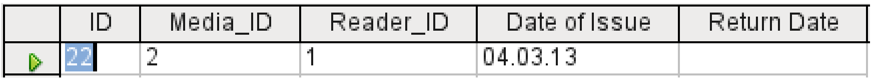
A basic rule is that no new entries are possible if the primary key in the queried table is not included in the query.



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

Figure 287: Defining aliases for the Loan_Date and Return_Date fields

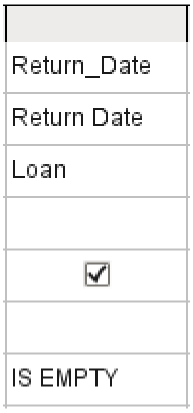
The Loan_Date and Return_Date fields are given aliases as shown in Figure 287. This does not cause them to be renamed but only to appear under these names for the user of the query.



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

Figure 288: Aliases used in column headings of query results

The table view in Figure 288 shows how the aliases replace the actual field names.



Return_Date
Return Date
Loan
☑
IS EMPTY

Figure 289: Limiting the records displayed to those with an empty Return_Date

The Return_Date field shown in Figure 289 is also given 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.

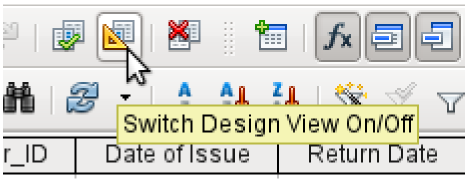
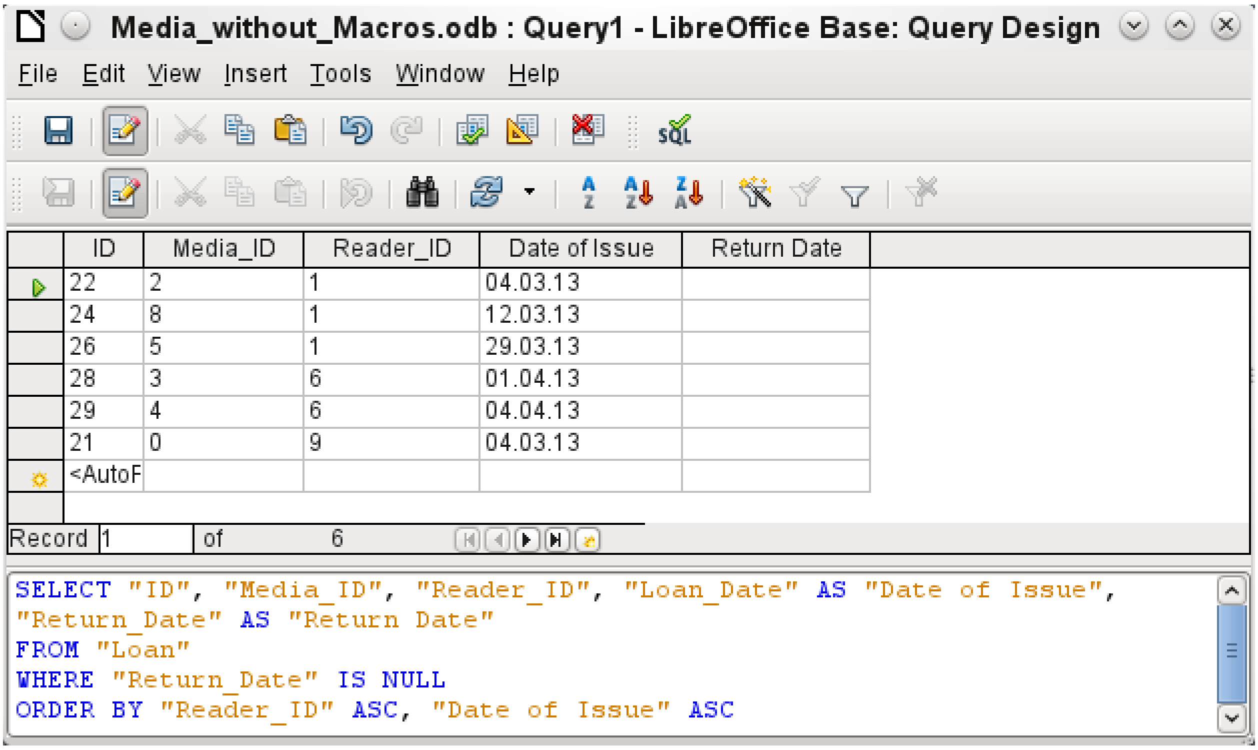


Figure 290: Switch Design View On/Off icon on Query Design toolbar

Switching between Design Mode and SQL Mode from time to time, using the tool shown in Figure 290, can help with learning the SQL language.


Media_without_Macros.odb : Query1 - LibreOffice Base: Query Design
File
Edit
View
Insert
Tools
Window
Help

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	
<AutoF				

Record 1 of 6

```
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
```

Figure 291: Query Design window with Design View off, showing the underlying SQL code

Here the SQL formula created by our previous choices is revealed in Figure 291. Some line breaks have been included to make it easier to read. 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 using the tool shown in Figure 292.

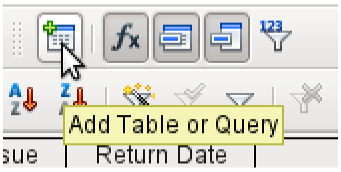


Figure 292: Add Table or Query icon on Design toolbar

Here further tables or queries can added subsequently 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 in Figure 293.

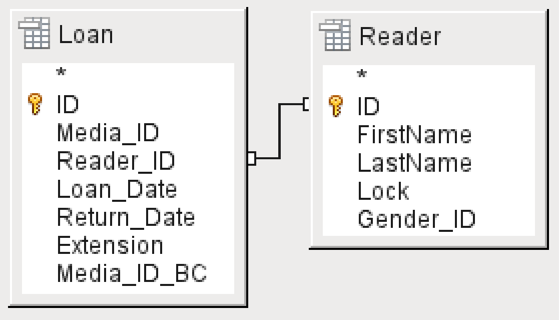


Figure 293: Reader table added to the graphical area of the Query Design window

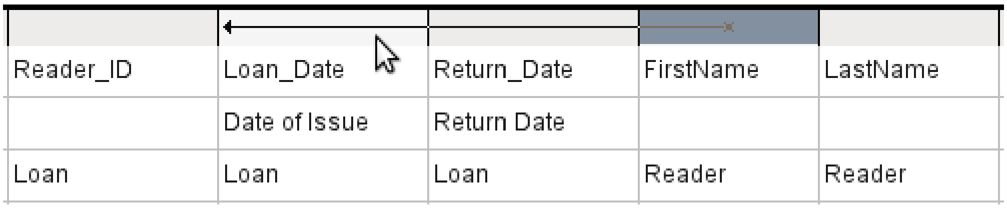
If a link is absent, it can be created at this point by dragging the mouse from "Loan"."Reader_ID" to "Reader"."ID".

Note

Linking tables only works in the internal database or in external relational databases. For example, tables from a spreadsheet cannot be linked together. They must first be imported into an internal database.

To create a link between the tables, a simple import is sufficient without additional creation of a primary key.

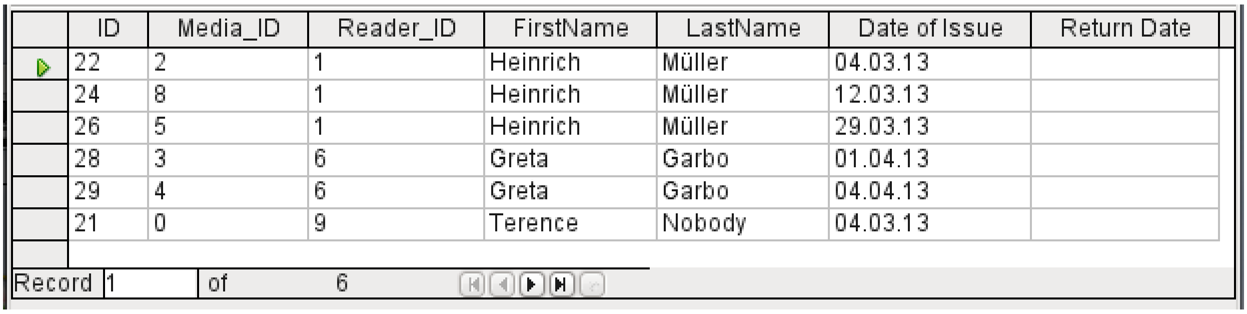
Now fields from the Reader table can be entered into the tabular area. The fields are initially added to the end of the query as shown in Figure 294.



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

Figure 294: Changing the position of a query field

The position of the fields can be corrected in the tabular area of the editor using the mouse. So for example, the FirstName field has been dragged into position directly before the Loan_Date field in Figure 295.



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

Figure 295: Updated tabular area of the Query Design window

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

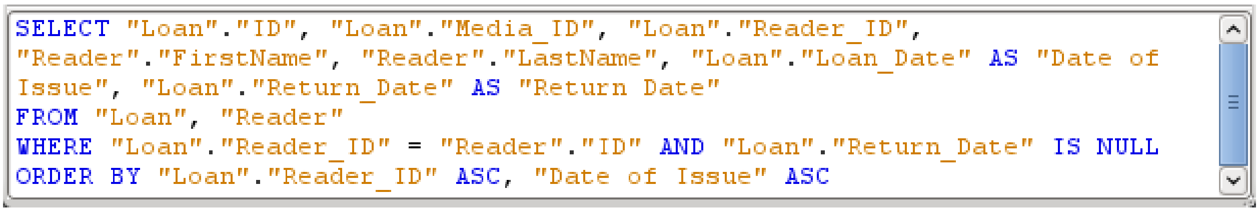
This query is no longer suitable for use as a query that allows new entries into the resulting table, since it lacks the primary key for the added Reader table. Only if this primary key is built in, does the query become editable again. In fact it then becomes completely editable so that the readers' 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.

Caution

Having a query that you can edit can create problems. Editing data in the query also 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
```

Figure 296: SQL code for query containing fields from multiple tables

If we now switch back to SQL View, we see in Figure 296 that all fields are now shown in double quotes: "Table _name"."Field_name" separated by a dot. 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:

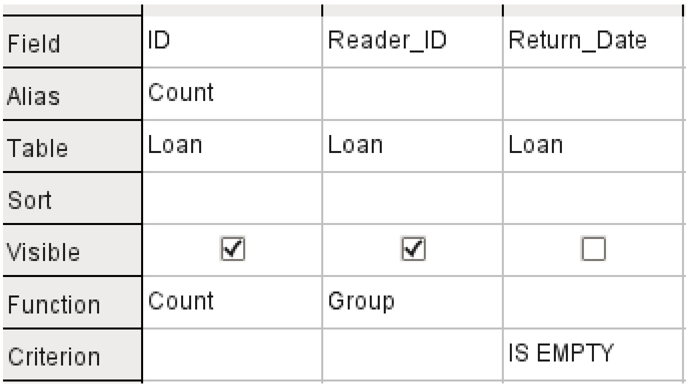
- The sequence of fields in the table area of the graphical user interface is changed (drag and drop "Loan_Date" to the left of "Loan"."Reader_ID", or
- An additional field is added, set to be invisible, just for sorting (however, the editor will register this only temporarily if no alias was defined 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").

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 query in Figure 297 calculates how many media have been loaned out, depending on the Reader_ID.



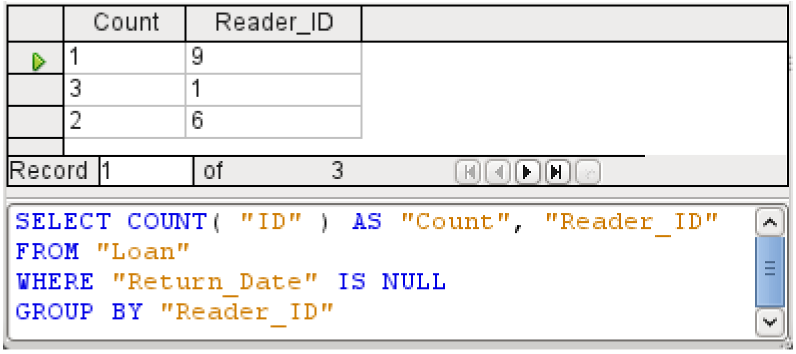
Field	ID	Reader_ID	Return_Date
Alias	Count		
Table	Loan	Loan	Loan
Sort			
Visible	☑	☑	□
Function	Count	Group	
Criterion			IS EMPTY

Figure 297: Using the Count function to calculate the number of on-loan items for each Reader_ID

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 in Figure 298 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"
```

Figure 298: Results and SQL code for query to determine the number of on-loan items for each Reader_ID

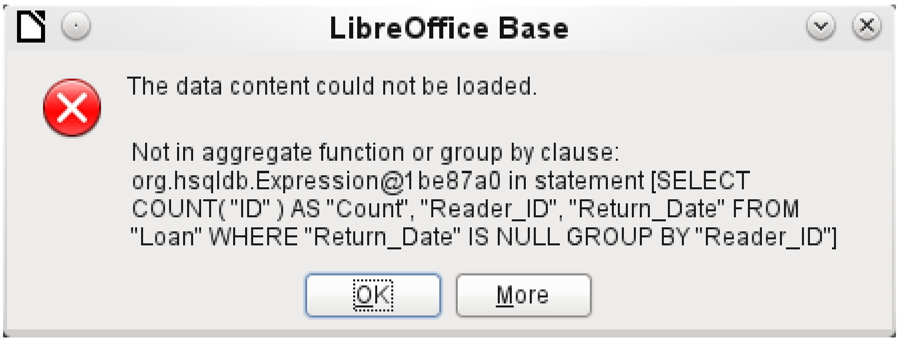
The result of the query shows that Reader_ID '0' has a total of 3 media on loan. If the Count function had been assigned to the Return_Date instead of the ID, every Reader_ID would have '0' media on loan, since Return_date is predefined as NULL.

The corresponding formula in SQL code is also shown in Figure 298.

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

For an explanation of the functions, see “Query enhancement using SQL Mode” on page 277.

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 error message shown in Figure 299:



The data content could not be loaded.

Not in aggregate function or group by clause: org.hsqldb.Expression@1be87a0 in statement [SELECT COUNT("ID") AS "Count", "Reader_ID", "Return_Date" FROM "Loan" WHERE "Return_Date" IS NULL GROUP BY "Reader_ID"]

Figure 299: Error dialog displayed when a field is not associated with a function

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

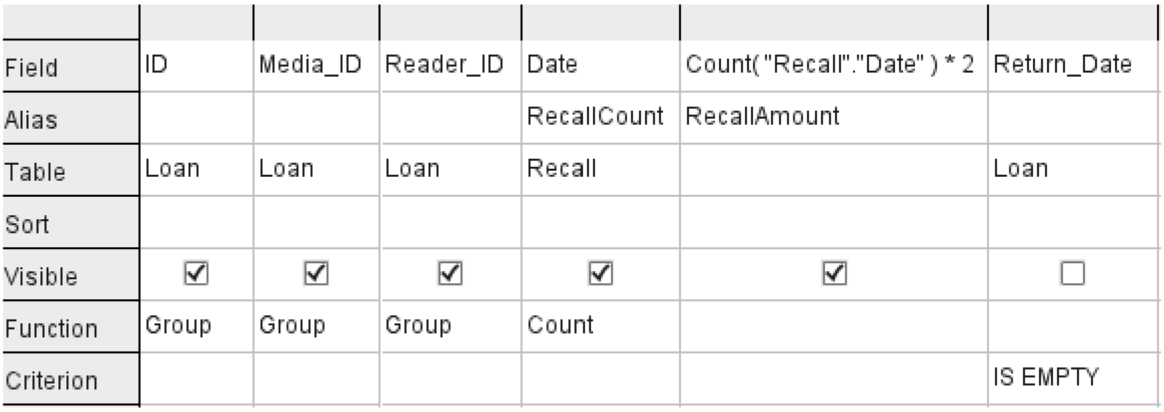
When a field is not associated with a function, the number of rows in the query output is determined by the search conditions. When a field is associated with a function, the number of rows in the query output is determined by whether there is any grouping or not. If there is no grouping, there is only 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 be associated with a grouping statement to prevent this conflict in the query output.

After this, the complete query is listed in the error message, but 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 comprises the SQL error code but this may not be very illuminating for the normal database user.

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

Figure 300: Query to calculate the fine per item for overdue media

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?

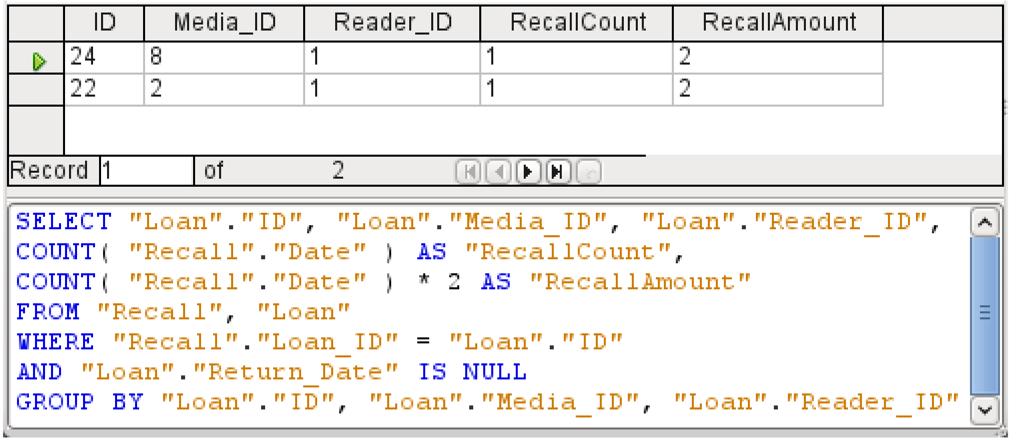
In the query shown in Figure 300, the Loan and Recalls tables are queried jointly. From the count of the data entries in the table Recalls, the total number of recall notices is determined. The fine for overdue media is set in the query to $2.00. Instead of a field name, the field designation is given 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 GUI, you must ensure that a decimal point rather than a comma is used as a decimal separator 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 as shown in Figure 301.



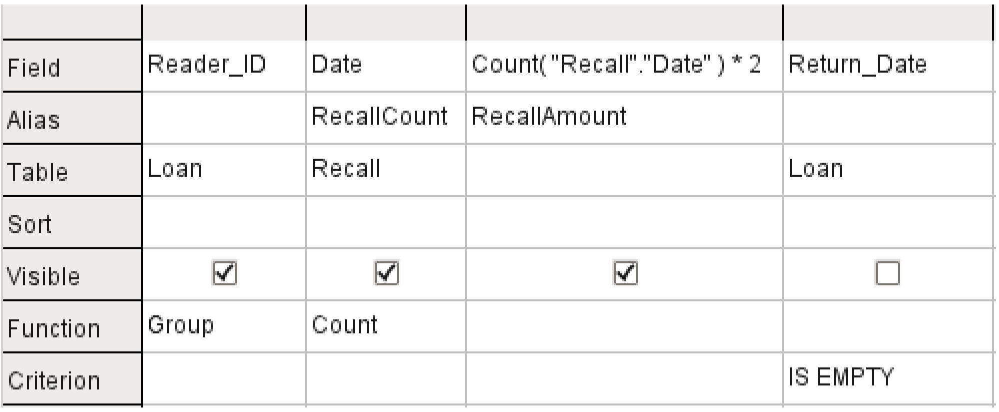
ID	Media_ID	Reader_ID	RecallCount	RecallAmount
24	8	1	1	2
22	2	1	1	2

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"
```

Figure 301: Results and SQL code for query to calculate the fine per item for overdue media

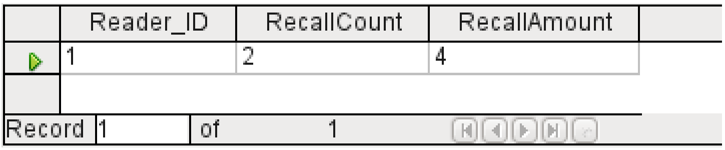
The query now yields for each medium still on loan the fines that have accrued, based on the recall notices issued and the additional multiplication field. The query structure in Figure 302 will also be useful for calculating the fines due from individual users.



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

Figure 302: Query to calculate the overdue media fines for each Reader_ID

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 is shown in Figure 303:



Reader_ID	RecallCount	RecallAmount
1	2	4

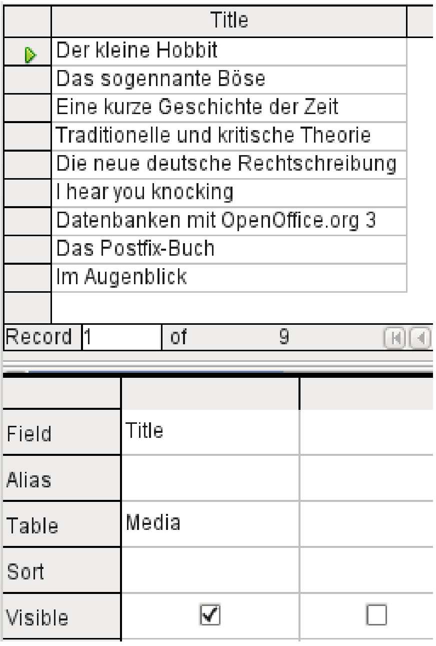
Record 1 of 1

Figure 303: Results of the query to calculate the overdue media fines for each Reader_ID

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. It is better to collect such data with a query.

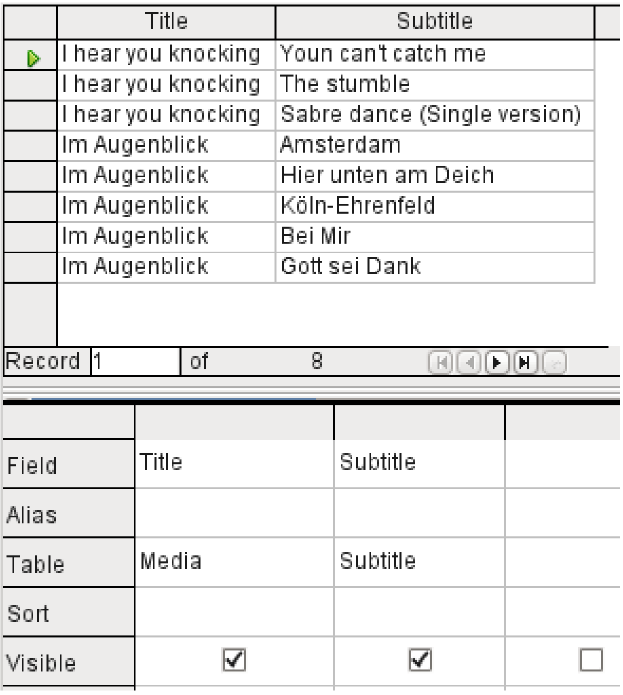


Title
Der kleine Hobbit
Das sogennante Böse
Eine kurze Geschichte der Zeit
Traditionelle und kritische Theorie
Die neue deutsche Rechtschreibung
I hear you knocking
Datenbanken mit OpenOffice.org 3
Das Postfix-Buch
Im Augenblick

Record 1 of 9

Field	Title	
Alias		
Table	Media	
Sort		
Visible	☑	☐

Figure 304: Query to show titles from the Media table



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	☑	☑	□

Figure 305: Query to show titles and subtitles from the Media table

The simple query for the Title field from the Media table in Figure 304 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 as shown in Figure 305. For all the other Titles, no subtitles exist. This corresponds to the join condition shown in Figure 306 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.

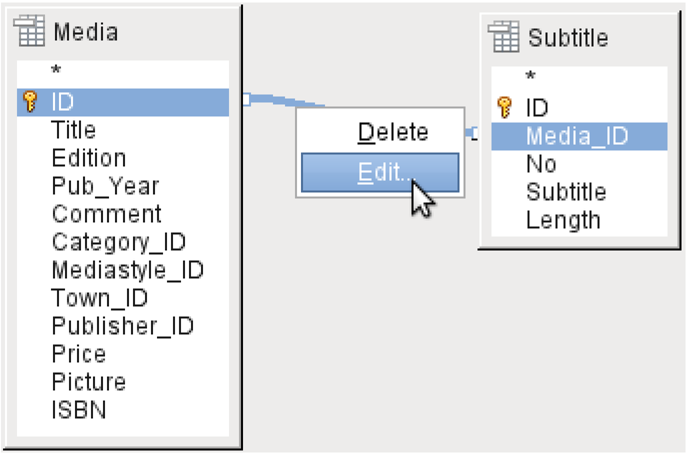
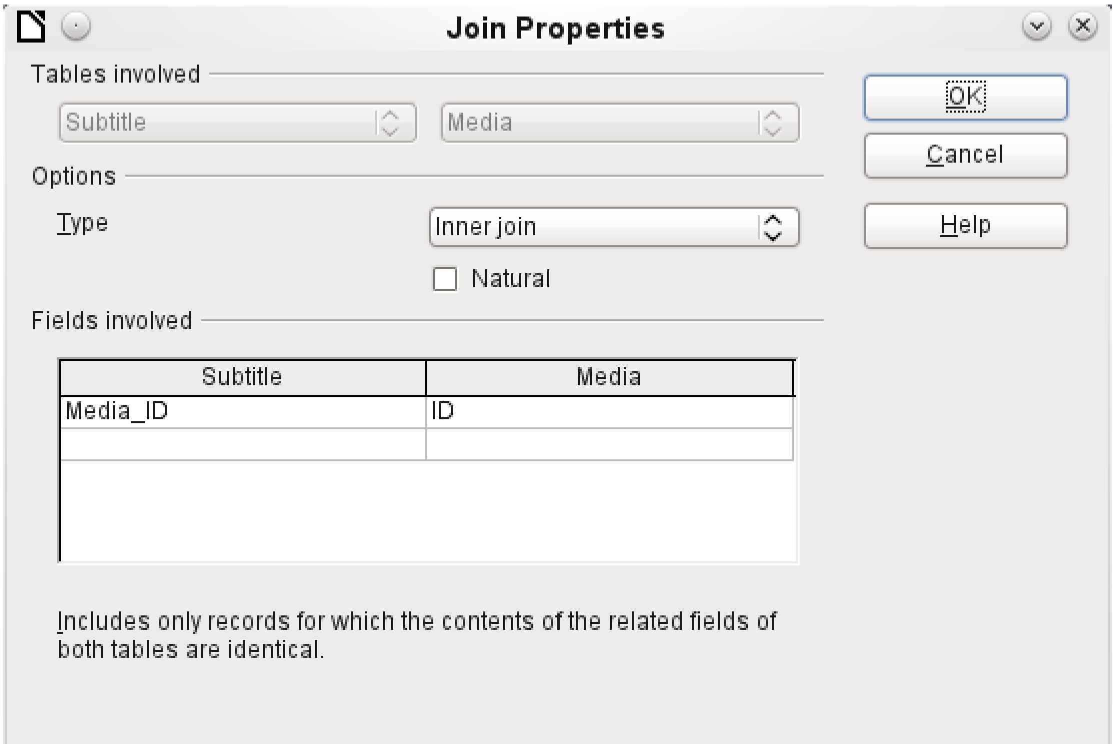


Figure 306: Editing a join within a query

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.



Join Properties

Tables involved

Subtitle | Media

Options

Type: Inner join

□ Natural

Fields involved

Subtitle	Media
Media_ID	ID

Includes only records for which the contents of the related fields of both tables are identical.

Figure 307: Join Properties dialog showing the default Type setting of Inner join

By default, relationships are set as Inner Joins, meaning the query returns only records for which the contents of the related fields of both tables are identical. Figure 307 shows the window providing information on the way this type of join works in practice.

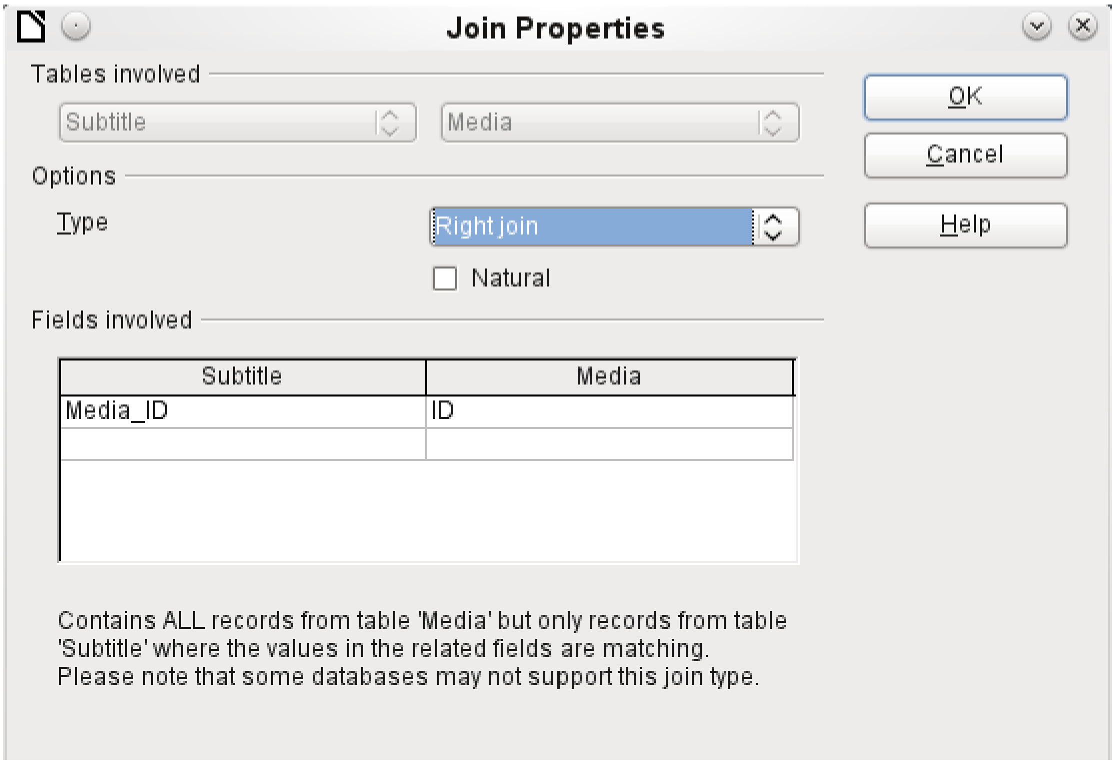
The two previously selected tables are listed as Tables Involved. They are not selectable here. The relevant fields from the two tables are read from the table definitions. If there is no relationship specified in the table definition, one can be created at this point for the query. However, if you have 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 are selected.

Or you can choose the opposite: that in any case all records from the Media table 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 shown in Figure 308.



Tables involved

Subtitle | Media

Options

Type: Right join

□ Natural

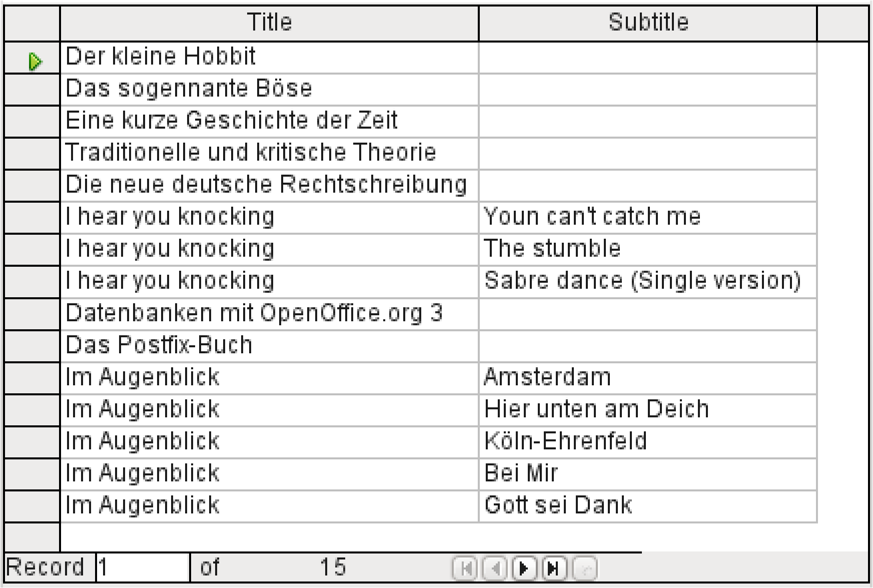
Fields involved

Subtitle	Media
Media_ID	ID

Contains ALL records from table 'Media' but only records from table 'Subtitle' where the values in the related fields are matching. Please note that some databases may not support this join type.

Figure 308: Join Properties dialog with Type set to Right join

After confirming the Right join, the query results look as we wanted them. Title and Subtitle are displayed together in one query shown in Figure 309. 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 8, Database Tasks, and in Chapter 9, Macros.



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

Figure 309: Result of query showing all titles and subtitles (where present)

Defining query properties

Starting with version 4.1 of LibreOffice, it is possible to define additional properties in the query editor.

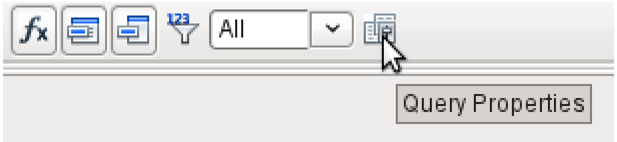
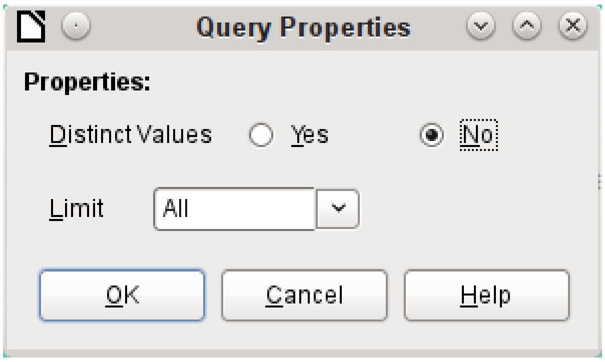


Figure 310: Query Properties icon on Design toolbar

Next to the button for opening the Query Properties is a combo box for regulating the number of displayed records as well as a button for Distinct Values (see Figure 310). These functions are repeated in the dialog shown in Figure 311:


Query Properties

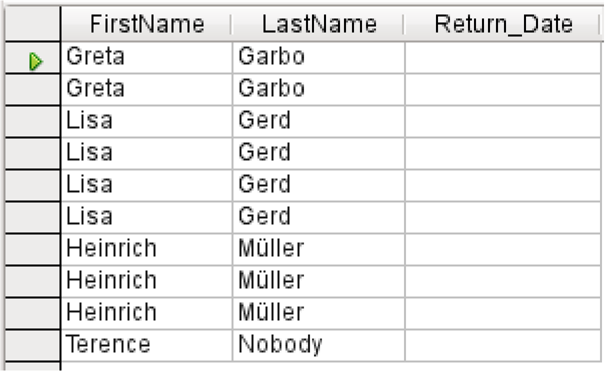
Properties:

Distinct Values ○ Yes ● No

Limit: All

Figure 311: Query Properties dialog

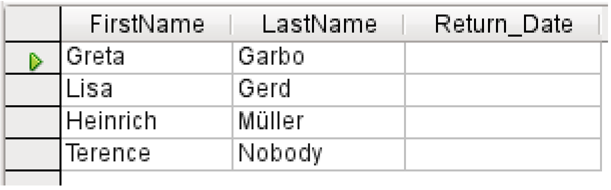
The Distinct Values setting determines whether the query should suppress duplicate records.



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

Figure 312: Query output with Distinct Values set to No

Suppose a query is made to determine which readers still have items out on loan. Their names are displayed if the return date field is empty. The names are displayed more than once for readers who have multiple items on loan as shown in Figure 312.



FirstName	LastName	Return_Date
Greta	Garbo	
Lisa	Gerd	
Heinrich	Müller	
Terence	Nobody	

Figure 313: Query output with Distinct Values set to Yes

If you choose Distinct Values, records with the same content disappear as shown in Figure 313.

The query then looks like this:

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

The original query:

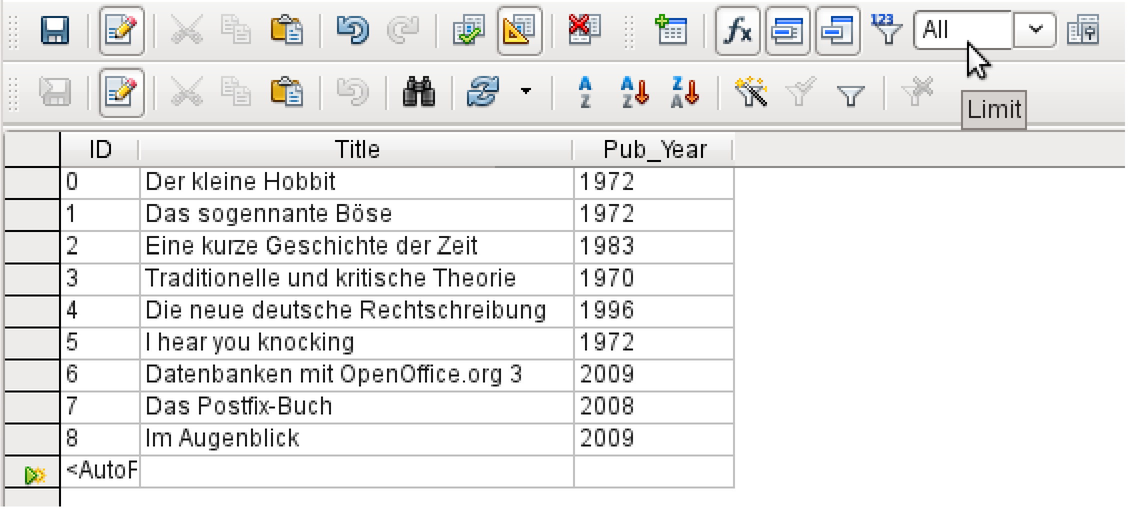
```
SELECT "Reader"."FirstName", "Reader"."LastName" …
```

Adding DISTINCT to the query suppresses the duplicate records.

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

In earlier versions to specify unique records it was necessary to switch from design view to SQL view to insert the DISTINCT qualifier. This property is backward-compatible with previous versions of LO without causing any problems.

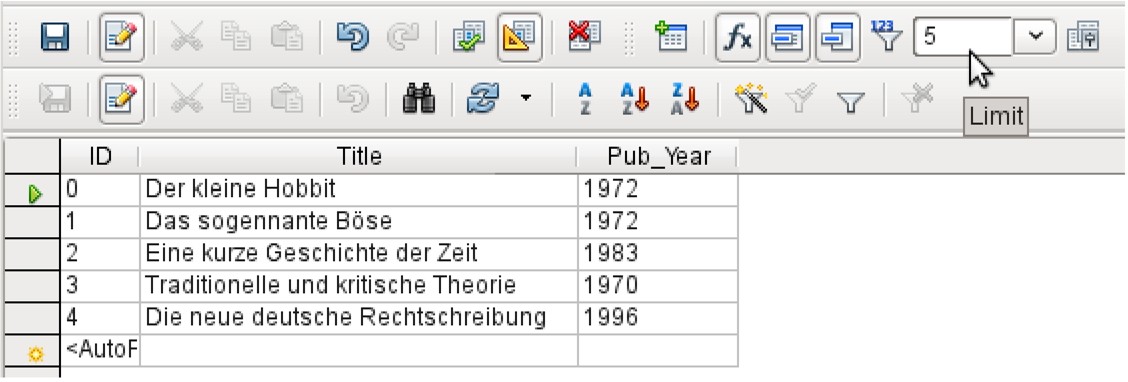
The Limit setting determines how many records are to be displayed in the query. Only a limited number of records are reproduced.



ID	Title	Pub_Year
0	Der kleine Hobbit	1972
1	Das sogennante 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
<AutoF		

Figure 314: Query output showing all records in the Media table

All records in the Media table are displayed in Figure 314. The query is editable as it includes the primary key.



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

Figure 315: Query output limited to show the first five records in the Media table

Only the first five records are displayed (ID 0-4) by setting the limit shown in Figure 315. A sort was not requested, so the default sort order by primary key is used. Despite the limitation in output, the query can be further edited. This distinguishes input in the graphical interface from what, in previous versions, was only accessible using SQL.

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

The original query has simply had “LIMIT 5” added. The size of the limit can be whatever you need.

Caution

Setting limits in the graphical interface is not backward-compatible. In all LO versions before 4.1, a limit could only be set in direct SQL mode. The limit then required a sort (`ORDER BY …`) or a condition (`WHERE …`).

Query enhancement using SQL Mode

If during graphical entry you use **View > Switch Design View On/Off** to switch Design View off, you see the SQL command for what previously appeared in Design View. This can help beginners learn the Standard Query Language for Databases. Sometimes it is also the only way to enter a query into the database when the GUI cannot translate your requirements into the necessary SQL commands.

```
SELECT * FROM "Table_name"
```

This will show everything that is in the Table_name table. The "*" represents all the fields of the table.

```
SELECT * FROM "Table_name" WHERE "Field_name" = 'Karl'
```

Here there is a significant restriction. Only those records are displayed for which the field Field_name contains the term ‘Karl’ – the exact term, not for example ‘Karl Egon’.

Sometimes queries in Base cannot be carried out using the GUI, as particular commands may not be recognized. In such cases it is necessary to switch Design View off and use **Edit > Run SQL command directly** for direct access to the database. This method has the disadvantage that you can only work with the query in SQL Mode.

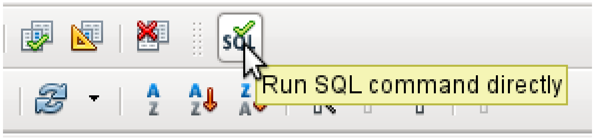


Figure 316: Run SQL command directly icon on SQL toolbar

Direct use of SQL commands is also accessible using the graphical user interface, as Figure 316 shows. Click the icon highlighted (**Run SQL command directly**) to turn the Design View Off/On icon off. Now when you click the Run icon, the query runs the SQL commands directly.

Creating Queries in SQL View

Here is an example of the extensive possibilities available for querying the database and specifying the type of result required:

```
SELECT [{LIMIT <offset> <limit> | TOP <limit>}][ALL | DISTINCT]
{ <Select-Formulation> | "Table_name".* | * } [, ...]
[INTO [CACHED | TEMP  | TEXT] "new_Table"]
FROM "Table_list"
[WHERE SQL-Expression]
[GROUP BY SQL-Expression [, ...]]
[HAVING SQL-Expression]
[{ UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT] }
|
INTERSECT [DISTINCT] } Query statement]
```

```
[ORDER BY Order-Expression [, ...]]
[LIMIT <limit> [OFFSET <offset>]];
```

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

This limits the number of records to be displayed. `LIMIT 10 20` starts at the 11th record and shows the following 20 records. `TOP 10` always shows the first 10 records. All entered values must be integers. This is the same as `LIMIT 0 10`. LIMIT 10 0 omits the first 10 records and displays all records starting from the 11th.

[ALL | DISTINCT]

`SELECT ALL` is the default. All records are displayed that fulfill the search conditions. Example: `SELECT ALL "Name" FROM "Table_name"` yields all names; if 'Peter' occurs three times and 'Egon' four times in the table, these names are displayed three and four times respectively. `SELECT DISTINCT "Name" FROM "Table_name"` suppresses query results that have the same content. In this case, 'Peter' and 'Egon' occur only once. `DISTINCT` refers to the whole of the record that is accessed by the query. So if, for example, the surname is asked for as well, records for 'Peter Müller' and 'Peter Maier' will count as distinct. Even if you specify the `DISTINCT` condition, both will be displayed.

<Select-Formulation>

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

Field names, calculations, record totals are all possible entries.

Note

Calculations within a database sometimes lead to unexpected results. Suppose a database contains the grades awarded for some classwork and you wish to calculate the average grade.

First you must add up all the grades. Suppose the sum is 80. Now this must be divided by the number of pupils (say 30). The query yields 2.

This occurs because you are working with fields of type INTEGER. The calculation must therefore yield an integer value. You need to have at least one value of the type DECIMAL in your calculation. You can achieve this either by using a HSQLDB conversion function or (more simply) you can divide by 30.0 rather than 30. This will give a result with one decimal place. Dividing by 30.00 gives two decimal places. Be careful to use English-style decimal points. The use of commas in queries is reserved for field separators.

In addition, different functions are available for the field shown. Except for COUNT(*) (which counts all the records) none of these functions access NULL fields.

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

`COUNT("Name")` counts all entries for the field Name.
`MIN("Name")` shows the first name alphabetically. The result of this function is always formatted just as it occurs in the field. Text is shown as text, integers as integers, decimals as decimals and so on.
`MAX("Name")` shows the last name alphabetically.
`SUM("Number")` can add only the values in numerical fields. The function fails for date fields.

Tip

This function also fails on time fields. Here the following may be useful:

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

The sum displayed is made up of separate hours, minutes and seconds. Then it is modified to yield the overall total in a single unit, in this case seconds. Here then only integers are added by the sum function. Now if you use:

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

you will get a time in hours with the minutes and seconds as decimal places. With suitable formatting, you can turn this back to a normal time value in a query or form.

`AVG("Number")` shows the average of the contents of a column. This function too is limited to numerical fields.
`SOME("Field_Name"), EVERY("Field_Name")`: Fields used with these functions must have the Yes/No[BOOLEAN] field type (contains only 0 or 1). Furthermore, they produce a summary of the field content to which they are applied.

`SOME` returns *TRUE* (or *1*) if at least one entry for the field is 1, and it returns *FALSE* (or *0*) only if all the entries are 0. `EVERY` returns 1 only if every entry for the field is 1, and returns FALSE if at least one entry is 0.

Tip

The Boolean field type is Yes/No[BOOLEAN]. However, this field contains only 0 or 1. In query search conditions, use either TRUE, 1, FALSE or 0. For the Yes condition, you can use either TRUE or 1. For the No condition, use either FALSE or 0. If you try to use either “Yes” or “No” instead, you get an error message. Then you will have to correct your error.

Example:

```
SELECT "Class", EVERY("Swimmer")
FROM "Table1"
GROUP BY "Class";
```

Class contains the names of the swimming class. Swimmer is a Boolean field describing whether a student can swim or not (1 or 0). Students contains the names of the students. Table1 contains these fields: its primary key, Class, Swimmer, and Students. Only Class and Swimmer are needed for this query.

Because the query is grouped by the entries of the field Class, EVERY will return a value for the field, Swimmer, for each class. When every person in a swimming class can swim, EVERY returns TRUE. Otherwise EVERY returns FALSE because at least one student of the class can not swim. Since the output for the Swimmer field is a checkbox, A check mark indicates TRUE, and no check mark indicates FALSE.

`VAR_POP | VAR_SAMP | STDDEV_POP | STDDEV_SAMP` are statistical functions and affect only integer and decimal fields.
All these functions return 0, if the values within the group are all equal.

The statistical functions do not allow the DISTINCT limitation to be used. They basically calculate over all values covered by the query, whereas DISTINCT excludes records with the same values from the display.

`[AS] "display_name"`: The fields can be given a different designation (alias) within the query.

"Table_name".* | * [, ...]

Use the symbol “*” to display the total content of a table, rather than enter all fields individually. It is unnecessary to use the table name, if the results will only apply to the one table. However, if the query includes all of the fields of one table and at least one field from a second table, use:
`"Table_name 1".*, “Table_name 2”.”Field_name”`.

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

The result of this query is to be written directly into a new table which is named here. The field properties for the new table are defined from the field definitions contained in the query. Writing into a new table does not work from SQL Mode as this handles only displayable results. Instead you must use **Tools > SQL**. The resultant table is initially not editable as it lacks a primary key.

FROM <Table_list>

```
"Table_name 1" [{CROSS | INNER | LEFT OUTER | RIGHT OUTER} JOIN
"Table_name 2" ON Expression] [, ...]
```

The tables which are to be jointly searched are usually in a list separated by commas. The relationship of the tables to one another is then additionally defined by the keyword `WHERE`.

If the tables are bound through a `JOIN` rather than a comma, their relationship is defined by the term beginning with `ON` which occurs directly after the second table.

A simple `JOIN` has the effect that only those records are displayed for which the conditions in both the tables apply.

Example:

```
SELECT "Table1"."Name", "Table2"."Class"
FROM "Table1", "Table2"
WHERE "Table1"."ClassID" = "Table2"."ID"
```

is equivalent to:

```
SELECT "Table1"."Name", "Table2"."Class"
FROM "Table1"
   JOIN "Table2"
ON "Table1"."ClassID" = "Table2"."ID"
```

Here the names and the corresponding classes are displayed. If a name has no class listed for it, that name is not included in the display. If a class has no names, it is also not displayed. The addition of `INNER` does not alter this.

```
SELECT "Table1"."Name", "Table2"."Class"
FROM "Table1"
   LEFT JOIN "Table2"
ON "Table1"."ClassID" = "Table2"."ID"
```

If `LEFT` is added, all Names from Table1 are displayed even if they have no Class. If, on the contrary, `RIGHT` is added, all Classes are displayed even if they have no names in them. Addition of `OUTER` need not be shown here. (Right Outer Join is the same thing as Right Join; Left Outer Join is the same thing as Left Join.)

```
SELECT "Table1"."Player1", "Table2"."Player2"
FROM "Table1" AS "Table1"
   CROSS JOIN "Table2" AS "Table1"
WHERE "Table1"."Player1" <> "Table2"."Player2"
```

A `CROSS JOIN` requires the table to be supplied with an alias, but the addition of the term `ON` is not always necessary. All records from the first table are paired with all records from the second table. Thus the above query yields all possible pairings of records from the first table with those of the second table except for pairings between records for the same player. In the case of a `CROSS JOIN`, the condition must not include a link between the tables specified in the `ON` term. Instead, `WHERE` conditions can be entered. If the conditions are formulated exactly as in the case of a simple JOIN, you get the same result:

```
SELECT "Table1"."Name", "Table2"."Class"
FROM "Table1"
   JOIN "Table2"
ON "Table1"."ClassID" = "Table2"."ID"
```

gives the same result as

```
SELECT "Table1"."Name", "Table2"."Class"
FROM "Table1" AS "Table1"
   CROSS JOIN "Table2" AS "Table2"
WHERE "Table1"."ClassID" = "Table2"."ID"
```

[WHERE SQL-Expression]

The standard introduction for conditions to request a more accurate filtering of the data. Here too the relationships between tables are usually defined if they are not linked together with JOIN.

[GROUP BY SQL-Expression [, …]]

Use this when you want to divide the query data into groups before applying the functions to each one of the groups separately. The division is based upon the values of the field or fields contained in the `GROUP BY` term.

Example:

```
SELECT "Name", SUM("Input"-"Output") AS "Balance"
FROM "Table1"
GROUP BY "Name";
```

Records with the same name `are sum`med. In the query result, the sum of Input – Output is given for each person. This field is to be called Balance. Each row of the query result contains a value from the Name table and the calculated balance for that specific value.

Tip

When fields are processed using a particular function (for example `COUNT`, `SUM` …), all fields that are not processed with a function but should be displayed are grouped together using `GROUP BY`.

[HAVING SQL-Expression]

The `HAVING` formula closely resembles the `WHERE` formula. The difference is that the WHERE formula applies to the values of selected fields in the query. The HAVING formula applies to selected calculated values. Specifically, the WHERE formula can not use an aggregate function as part of a search condition; the HAVING formula does.

The HAVING formula serves two purposes as shown in the two examples below. In the first one, the search condition requires that the minimum run-time be less than 40 minutes. In the second example, the search condition requires that an individual's balance must be positive.

The query results for the first one lists the names of people whose run-time has been less than 40 minutes at least one time and the minimum run-time. People whose run-times have all be greater than 40 minutes are not listed.

The query results for the second one lists the names of people who have a total greater output than input and their balance. People whose balance is 0 or less are not listed.

Examples:

```
SELECT "Name", "Runtime"
FROM "Table1"
GROUP BY "Name", "Runtime"

HAVING MIN("Runtime") < '00:40:00';
SELECT "Name", SUM("Input"-"Output") AS "Balance"
FROM "Table1"
GROUP BY "Name"
HAVING SUM("Input"-"Output") > 0;
```

[SQL Expression]

SQL expressions are combined according to the following scheme:

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

Example:

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

The records read from the table are those for which a Return_date has been entered and the ReaderID is 2. In practice this means that all media loaned to a specific person and returned can be retrieved. The conditions are only linked with AND. The NOT refers only to the first condition.

```
SELECT *
FROM "Table_name"
WHERE NOT ("Return_date" IS NULL AND "ReaderID" = 2);
```

Parentheses around the condition, with NOT outside them shows only those records that do not fulfill the condition in parentheses completely. This would cover all records, except for those for ReaderID number 2, which have not yet been returned.

[SQL Expression]: conditions

```
{ value [|| value]
```

A value can be single or several values joined by two vertical lines ||. Naturally this applies to field contents as well.

```
SELECT "Surname" || ', ' || "First_name" AS "Name"
FROM "Table_name"
```

The content of the Surname and First_name fields are displayed together in a field called Name. Note that a comma and a space are inserted between Surname and First_name.

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

These signs correspond to the well-known mathematical operators:

```
{ Equal to | Less than | Less than or equal to | Greater than |
Greater than or equal to | Not equal to | Not equal to }
```

```
| value IS [NOT] NULL
```

The corresponding field has no content, because nothing has been written to it. This cannot be determined unambiguously in the GUI, since a visually empty text field does not mean that the field is completely without content. However the default set-up in Base is that empty fields in the database are set to NULL.

```
| EXISTS(Query_result)
```

Example:

```
SELECT "Name"
FROM "Table1"
WHERE EXISTS
   (SELECT "First_name"
   FROM "Table2"
   WHERE "Table2"."First_name" = "Table1"."Name")
```

The names from Table1 are displayed for which first names are given in Table2.

```
| Value BETWEEN Value AND Value
```

`BETWEEN value1 AND value2` yields all values from value1 up to and including value2. If the values are letters, an alphabetic sort is used in which lower-case letters have the same value as the corresponding upper-case ones.

```
SELECT "Name"
FROM "Table_name"
WHERE "Name" BETWEEN 'A' AND 'E';
```

This query yields all names beginning with A, B, C or D (and also with the corresponding lower-case letters). As E is set as the upper limit, names beginning with E are not included. The letter E itself occurs just before the names that begin with E.

```
| value [NOT] IN ( {value [, ...] | Query result } )
```

This requires either a list of values or a query. The condition is fulfilled if the value is included in the value list or the query result.

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

The LIKE operator is one that is needed in many simple search functions. The value is entered using the following pattern:
'%' stands for any number of characters (including 0),
'_' replaces exactly one character.

To search for '%' or '_' itself, the characters must immediately follow another character defined as ESCAPE.

```
SELECT "Name"
FROM "Table_name"
WHERE "Name" LIKE '\_%' ESCAPE '\'
```

This query displays all names that begin with an underscore. '\' is defined here as the ESCAPE character.

[SQL Expression]: values

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

The values may have a preceding sign. Addition, subtraction, multiplication, division and concatenation of expressions are allowed. An example of concatenation:

```
SELECT "Surname"||', '||"First_name"
FROM "Table"
```

In this way records are displayed by the query with a field containing "Surname, First_name". The concatenation operator can be qualified by the following expressions.

```
| ( Condition )
```

See the previous section for this.

```
| Function ( [Parameter] [,...] )
```

See the section on Functions in the appendix.

The following queries are also referred to as sub-queries (subselects).

```
| Query result which yields exactly one answer
```

As a record can only have one value in each field, only a query which yields precisely one value can be displayed in full.

```
| {ANY|ALL} (Queryresult which yields exactly one answer from a
whole column)
```

Often there is a condition that compares an expression with a whole group of values.

Combined with `ANY` this signifies that the expression must occur at least once in the group. This can also be specified using the `IN` condition. `= ANY` yields the same result as `IN`.

Combined with `ALL` it signifies that all values of the group must correspond to the one expression.

[SQL Expression]: Expression

```
{ 'Text' | Integer | Floating-point number
| ["Table".]"Field" | TRUE | FALSE | NULL }
```

Basically values serve as arguments for various expressions, dependent on the source format. To search for the content of text fields, place the content in quotes. Integers are written without quotes, as are floating-point numbers.
Fields stand for the values that occur in those fields in the table. Usually fields are compared either with each other or with specific values. In SQL, field names should be placed in double quotes, as they may not be correctly recognized otherwise. Usually SQL assumes that text without double quotes is without special characters, that is a single word without spaces and in upper case. If several tables are contained in the query, the table name must be given in addition to the field name, separated from the latter by a period.
TRUE and FALSE usually derive from Yes/No fields.
NULL means no content. It is not the same thing as 0 but rather corresponds to “empty”.

UNION [ALL | DISTINCT] Query_result

This links queries so that the content of the second query is written under the first. For this to work, all fields in both queries must match in type. This linkage of several queries’ functions only in direct SQL command mode.

```
SELECT "First_name"
FROM "Table1"
UNION DISTINCT
   SELECT "First_name"
   FROM "Table2";
```

This query yields all first names from Table1 and Table2; the additional term DISTINCT means that no duplicate first names will be displayed. DISTINCT is the default in this context. By default the first names are sorted alphabetically in ascending order. ALL causes all first names in Table1 to be displayed, followed by the first name in Table2. In this case the default is to sort by primary key.

Using this query technique makes it possible to list values from a record directly under one another in a column. Suppose you have a table called Stock in which there are fields for

Sales_price, Rebate_price_1, and Rebate_price_2. From this you wish to calculate a combination field which will list these prices directly under each other.

```
SELECT
   "Sales_price"
FROM "Stock" WHERE "Stock_ID" = 1
   UNION
   SELECT
      "Rebate_price_1"
   FROM "Stock" WHERE "Stock_ID" = 1
   UNION
   SELECT
      "Rebate_price_2"
   FROM "Stock" WHERE "Stock_ID" = 1;
```

The primary key for the Stock table must naturally be set using a macro, as the combination field will have a matching entry.

MINUS [DISTINCT] | EXCEPT [DISTINCT] Query_result

```
SELECT "First_name"
FROM "Table1"
EXCEPT
   SELECT "First_name"
   FROM "Table2";
```

Shows all first names from Table1 except for the first names contained in Table 2. `MINUS` and `EXCEPT` lead to the same result. Sorting is alphabetic.

INTERSECT [DISTINCT] Query_result

```
SELECT "First_name"
FROM "Table1"
INTERSECT
   SELECT "First_name"
   FROM "Table2";
```

This displays the first names that occur in both tables. Sorting is again alphabetic. At present this only works in direct SQL command mode.

[ORDER BY Ordering-Expression [, …]]

The expression can be a field name, a column number (beginning with 1 from the left), an alias (formulated with AS for example) or a composite value expression (see [SQL Expression]: values). The sort order is usually ascending (ASC). If you want a descending sort you must specify DESC explicitly.

```
SELECT "First_name", "Surname" AS "Name"
FROM "Table1"
ORDER BY "Surname";
```

is identical to

```
SELECT "First_name", "Surname" AS "Name"
FROM "Table1"
ORDER BY 2;
```

is identical to

```
SELECT "First_name", "Surname" AS "Name"
FROM "Table1"
ORDER BY "Name";
```

[LIMIT <limit> [OFFSET <offset>]]:

`LIMIT 10` shows only 10 records. Adding `OFFSET 20` causes the display to begin at the 21st record. This final form of display limitation requires either a sort instruction (`ORDER BY…`) or a condition (`WHERE…`).

All entered values for the limit must be integers. It is not possible to replace an entry by a subquery so that, for example, the five last records of a series can be displayed each time.

Using an alias in a query

Queries can reproduce fields with changed names.

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

The *Surname* field is called *Name* in the display.

If a query involves two tables, each field name must be preceded by the name of the table:

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

The table name can also be given an alias, but this change is only in the query and will not be reflected in table view. If such an alias is set, all the table names in the query must be altered accordingly:

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

The assignment of an alias for a table can be carried out more briefly without using the term AS:

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

This however makes the code less readable. Because of this, the abbreviated form should be used only in exceptional circumstances.

✓ **Note**

The code for the query editor in the graphical user interface has recently been changed so that alias designations are created without using the prefix AS. This is because, when external databases are used, whose method of specifying aliases cannot be predicted, the inclusion of AS has led to error messages.

If the query code is opened for editing, not directly in SQL but using the GUI, existing aliases lose their AS prefix. If this is important to you, you must always edit queries in SQL view.

An alias name also makes it possible to use a table with corresponding filtering more than once within a query:

```
SELECT "KasseAccount"."Balance", "Account"."Date",
   "a"."Balance" AS "Actual",
   "b"."Balance" AS "Desired"
FROM "Account"
   LEFT JOIN "Account" AS "a"
   ON "Account"."ID" = "a"."ID" AND "a"."Balance" >= 0
   LEFT JOIN "Account" AS "b"
   ON "Account"."ID" = "b"."ID" AND "b"."Balance" < 0
```

Queries for the creation of list box fields

List box fields show a value that does not correspond to the content of the underlying table. They are used to display the value assigned by a user to a foreign key rather than the key itself. The value that is finally saved in the form must not occur in the first position of the list box field.

```
SELECT "FirstName", "ID"
FROM "Table1";
```

This query would show all first names and the primary key "ID" values that the form’s underlying table provides. Of course it is not yet optimal. The first names appear unsorted and, in the case of identical first names, it is impossible to determine which person is intended.

```
SELECT "FirstName"||' '||"LastName", "ID"
FROM "Table1"
ORDER BY "FirstName"||' '||"LastName";
```

Now the first name and the surname both appear, separated by a space. The names become distinguishable and are also sorted. But the sort follows the usual logic of starting with the first letter of the string, so it sorts by first name and only then by surname. A different sort order to that in which the fields are displayed would only be confusing.

```
SELECT "LastName"||', '||"FirstName", "ID"
FROM "Table1"
ORDER BY "LastName"||', '||"FirstName";
```

This now leads to a sorting that corresponds better to normal custom. Family members appear together, one under another; however different families with the same surname would be interleaved. To distinguish them, we would need to group them differently within the table.

There is one final problem: if two people have the same surname and first name, they will still not be distinguishable. One solution might be to use a name suffix. But imagine how it would look if a salutation read Mr "Müller II"!

```
SELECT "LastName"||', '||"FirstName"||' - ID:'||"ID", "ID"
FROM "Table1"
ORDER BY "LastName"||', '||"FirstName"||"ID";
```

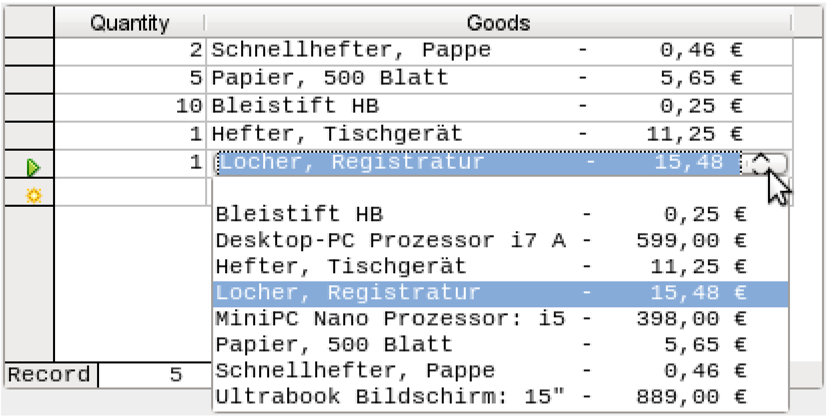
Here all records are distinguishable. What is actually displayed is "LastName, FirstName – ID:ID value". So for instance, someone named Heinrich Müller with an ID of 1 would display as “Müller, Heinrich – ID: 1”.

In the loan form, there is a list box which only shows the media that have not yet been loaned out. It is created using the following SQL formula:

```
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
```

It is important that this list field is always be updated if a medium within it goes out on loan.

The following list field in Figure 317 shows the content of several fields in tabular form so that elements that belong together are directly under one another.



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 €
Ultrabook Bildschirm: 15" -  889,00 €
```

Figure 317: Example list field

To make such a representation work, you must first choose a suitable fixed-width font. Courier or any mono-font such as Liberation Mono could be used here. You create the tabular form using SQL code:

```
SELECT
   LEFT("Stock"||SPACE(25),25) || ' - ' ||
      RIGHT(SPACE(8)||"Price",8) || ' €',
   "ID"
FROM "Stock"
ORDER BY ("Stock" || ' - ' || "Price" || ' $') ASC
```

The content of the Stock field has been padded with spaces so that the whole string has a minimum length of 25 characters. Afterwards the first 25 letters are placed there and the surplus cut.

It gets more complicated when the contents of the list field contains non-printing characters like newlines. Then the code must be customized to fit:

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

This replaces a newline in Linux with a space. In Windows you must additionally remove the carriage return (CHAR(13)).

The number of spaces that are necessary can also be determined on a per-query basis. This prevents a value of “Stock” being accidentally truncated.

```
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
```

As the price is to be shown right-justified, it is left-padded with spaces and placed a maximum of eight characters from the right. The representation selected will work for all prices up to $ 99999,99.

If you want to replace the decimal point with a comma within SQL, you will need some extra code:

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

Queries as a basis for additional information in forms

If you wish a form to display additional Information that would otherwise not be visible, there are various query possibilities. The simplest is to retrieve this information with independent queries and insert the results into the form. The disadvantage of this method is that changes in the records may affect the query result, but these changes are not automatically displayed.

Table 9 is an example from the sphere of stock control for a simple checkout.

The checkout table contains totals and foreign keys for stock items, and a receipt number. The shopper has very little information if there is no additional query result printed on the receipt. After all, the items are identified only by reading in a barcode. Without a query, the form shows only:

Table 9: Number of items for each barcode

Total	***Barcode***
3	17
2	24

What is hidden behind the numbers cannot be made visible by using a list box, as the foreign key is input directly using the barcode. In the same way, it is impossible to use a list box next to the item to show at least the unit price.

Here a query can help.

```
SELECT "Checkout"."Receipt_ID", "Checkout"."Total", "Stock"."Item",
"Stock"."Unit_Price", "Checkout"."Total"*"Stock"."Unit_price" AS
"Total_Price"
FROM "Checkout", "Stock"
WHERE "Stock"."ID" = "Checkout"."Item_ID";
```

Now at least after the information has been entered, we know how much needs to be paid for 3 * Item'17'. In addition only the information relevant to the corresponding Receipt_ID needs to be filtered through the form. What is still lacking is what the customer needs to pay overall.

```
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";
```

Design the form to show one record of the query at a time. Since the query is grouped by Receipt_ID, the form shows information about one customer at a time.

Tip

If a form needs to show date values that depend on another date (for example a loan period for a medium might be 21 days so what is the return date?), you can’t use HSQLDB’s built-in functions. There is no “DATEADD“ function.

The query

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

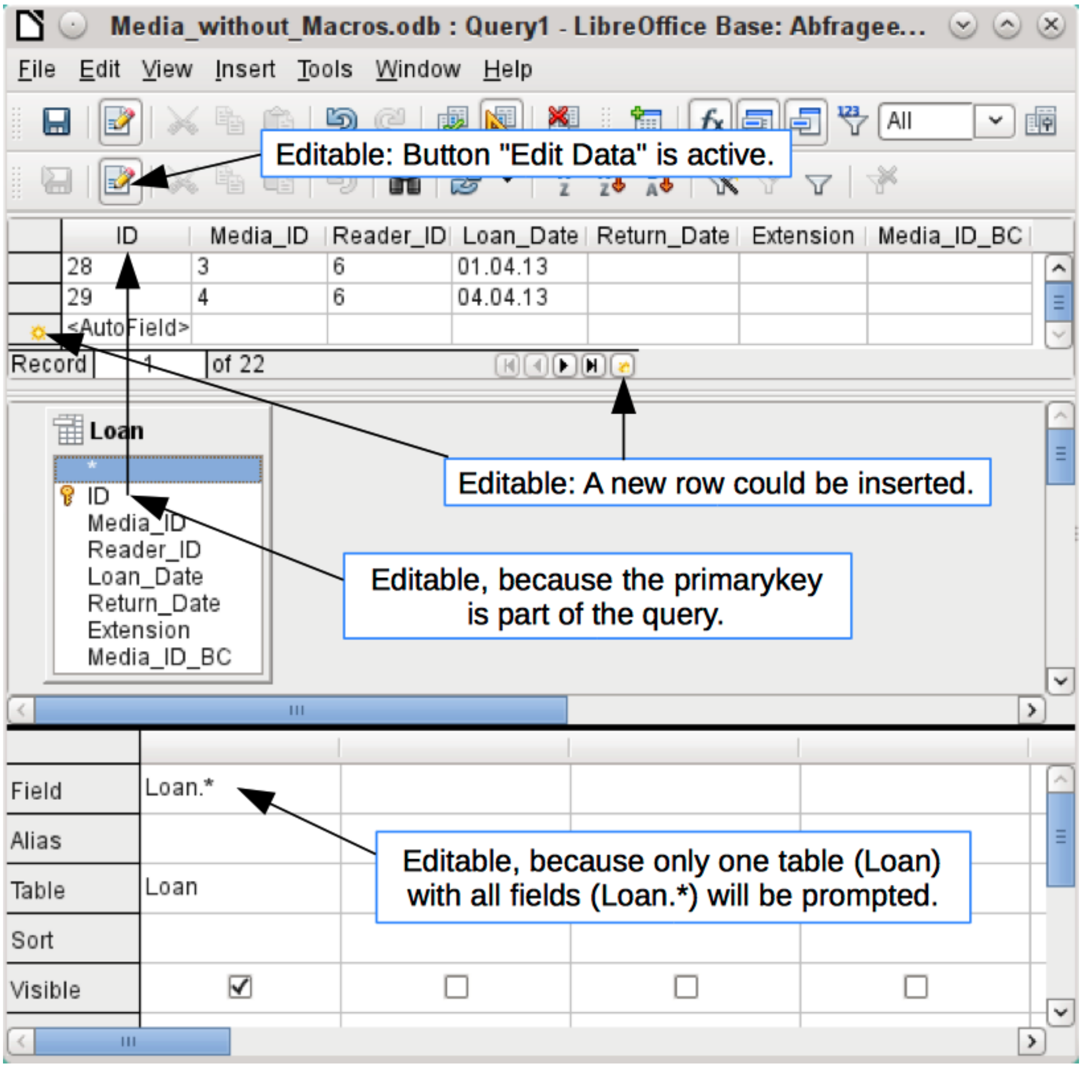
will yield the correct target date for return in a form. This query counts days from 30.12.1899. This is the default date, which Calc also uses as a zero value.

However the returned value is just a number and not a date that can be used in a further query.

The returned number is unsuitable for use in queries because the formatting of queries is not saved. You need to create a view instead.

Data entry possibilities within queries

To make entries into a query that prompts only one table, the primary key for the table underlying the query must be present as shown in Figure 318.



Media_without_Macros.odb : Query1 - LibreOffice Base: Abfragee...
File Edit View Insert Tools Window Help
All
Record 1 of 22
Loan
*
ID
Media_ID
Reader_ID
Loan_Date
Return_Date
Extension
Media_ID_BC

ID	Media_ID	Reader_ID	Loan_Date	Return_Date	Extension	Media_ID_BC
28	3	6	01.04.13			
29	4	6	04.04.13			
<AutoField>						

Field	Loan.*			
Alias				
Table	Loan			
Sort				
Visible	☑	□	□	□

Editable: Button "Edit Data" is active.

Editable: A new row could be inserted.

Editable, because the primarykey is part of the query.

Editable, because only one table (Loan) with all fields (Loan.*) will be prompted.

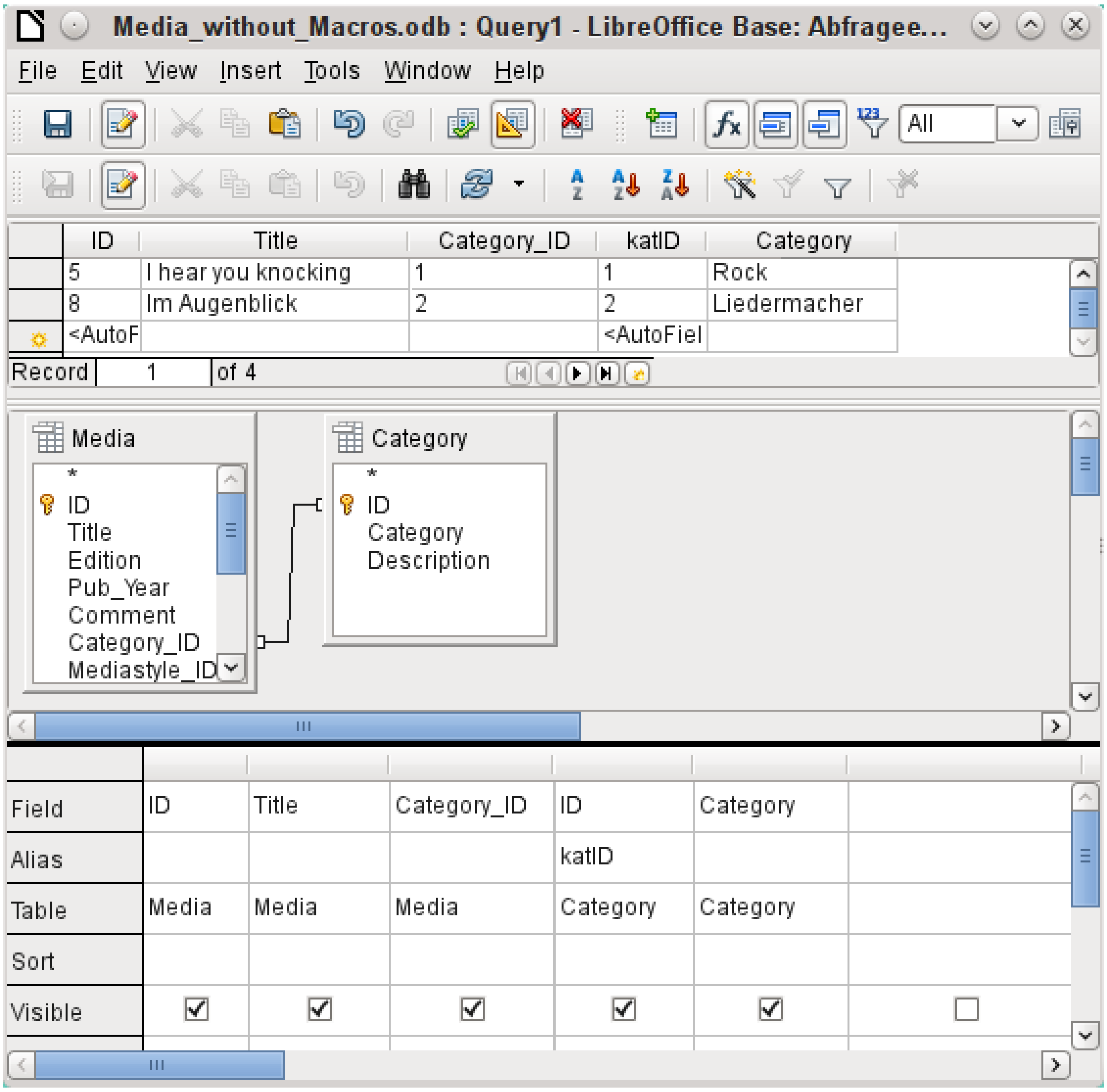
Figure 318: Conditions for a query to be editable

In the loaning of media, it makes no sense to display items that have already been returned some time ago.

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

In this way a form can show within a table control field everything that a particular reader has borrowed over time. Here too the query must filter using the appropriate form structure (reader in the main form, query in the sub-form), so that only media that are actually on loan are displayed. The query is suitable for data entry since the primary key is included in the query.

When a query consists of more than one table, it is editable if the primary keys of all tables are present and the tables are not accessed via an alias.



	ID	Title	Category_ID	katID	Category
	5	I hear you knocking	1	1	Rock
	8	Im Augenblick	2	2	Liedermacher
☼	<AutoF			<AutoFiel	

Record 1 of 4

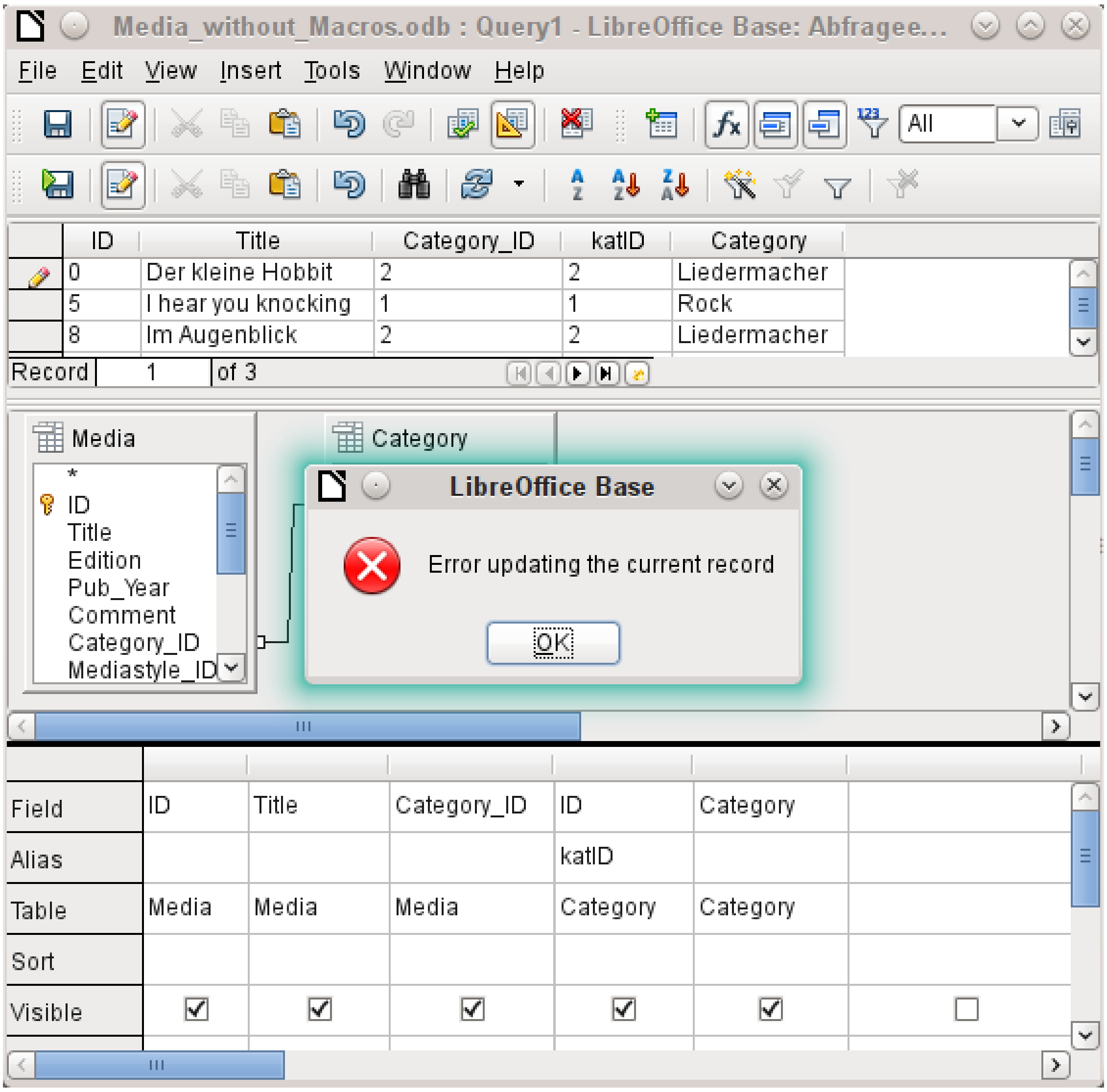
Field	ID	Title	Category_ID	ID	Category	
Alias				katID		
Table	Media	Media	Media	Category	Category	
Sort						
Visible	☑	☑	☑	☑	☑	□

Figure 319: Example of an editable query involving two tables

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

The query in Figure 319 is editable as both primary keys are included and can be accessed in the tables without using an alias. A query that links several tables together also requires all primary keys to be present.

In a query that involves several tables, it is not possible to alter a foreign key field in one table that refers to a record in another table. In the record shown in Figure 320, an attempt was made to change the category for the title “The Little Hobbit”. The Category_ID field was changed from 0 to 2. The change seemed to go through and the new category appeared in the record. However it proved impossible to save it.



ID	Title	Category_ID	katID	Category
0	Der kleine Hobbit	2	2	Liedermacher
5	I hear you knocking	1	1	Rock
8	Im Augenblick	2	2	Liedermacher

Record 1 of 3

Error updating the current record

Field	ID	Title	Category_ID	ID	Category	
Alias				katID		
Table	Media	Media	Media	Category	Category	
Sort						
Visible	☑	☑	☑	☑	☑	□

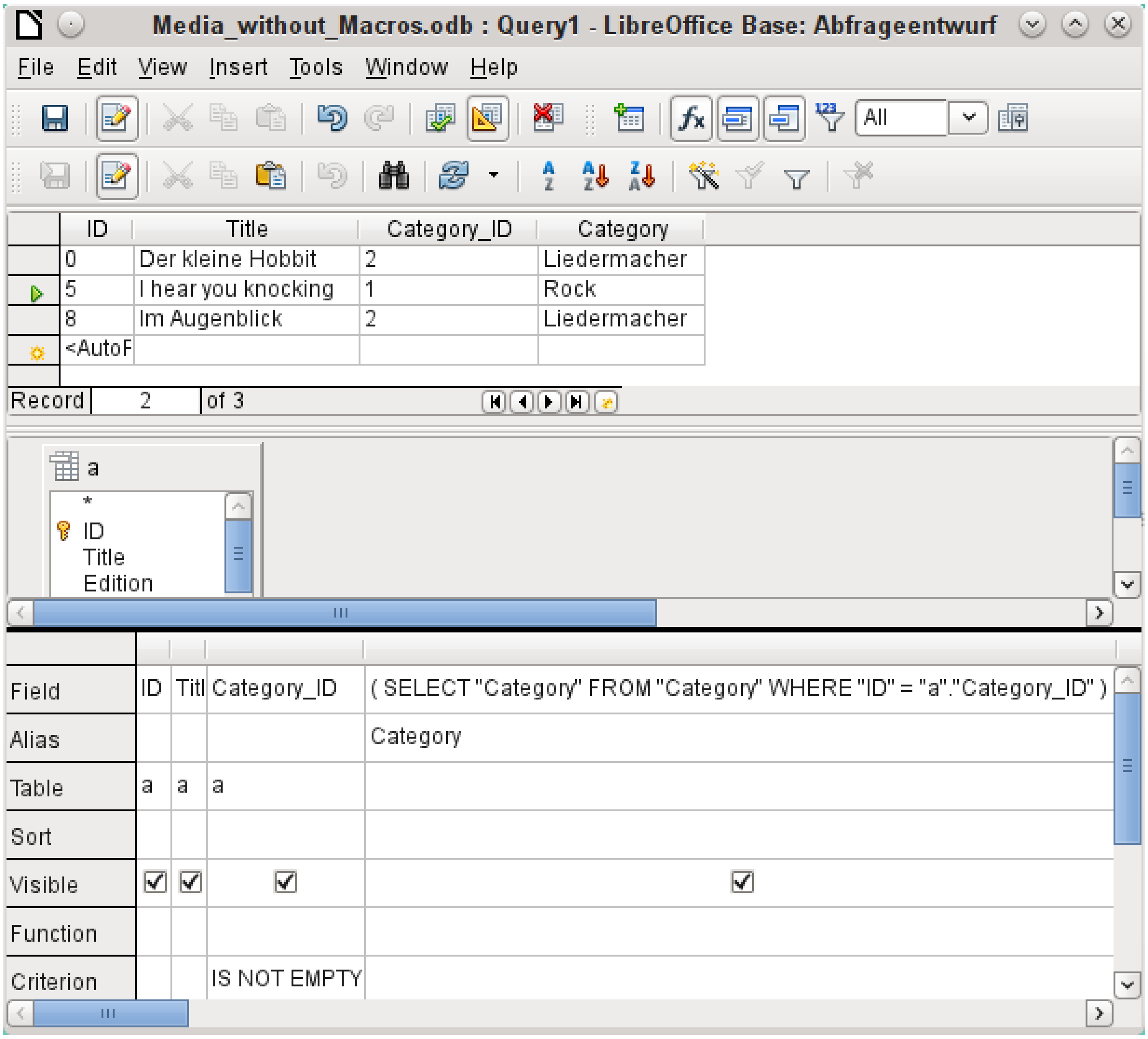
Figure 320: Error when edited query cannot be saved

However it is possible to edit the content of the corresponding category record, for example to replace “Fantasie” by “Fantasy”. The name of the category will then be altered for all records that are linked to this category.

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

In this query the Media table is accessed using an alias. The query cannot be edited. It makes no difference in this case whether or not the primary key is present in the query.

In the above example, this problem is easily avoided. If, however, a correlated subquery (see page 296) is used, you need to use a table alias. A query is only editable in that case if it contains only one table in the main query.


Media_without_Macros.odb : Query1 - LibreOffice Base: Abfrageentwurf
File Edit View Insert Tools Window Help
All

	ID	Title	Category_ID	Category
	0	Der kleine Hobbit	2	Liedermacher
▷	5	I hear you knocking	1	Rock
	8	Im Augenblick	2	Liedermacher
☼	<AutoF			

Record 2 of 3

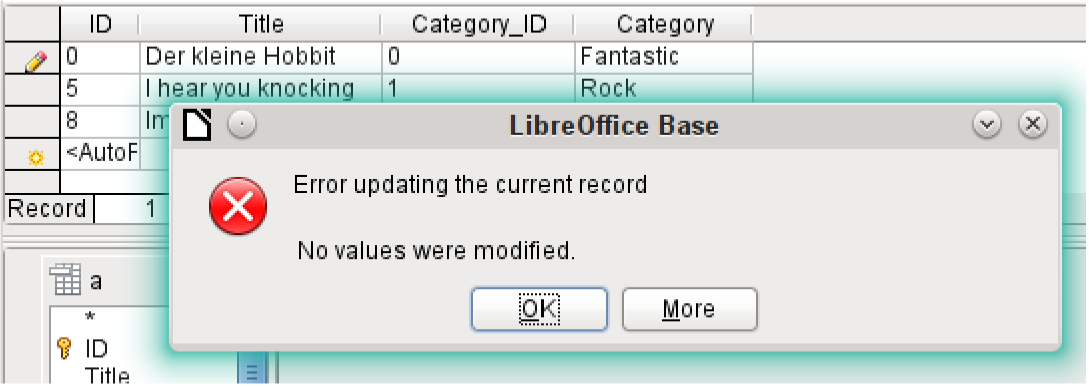
a
*
ID
Title
Edition

Field	ID	Titl	Category_ID	(SELECT "Category" FROM "Category" WHERE "ID" = "a"."Category_ID")
Alias				Category
Table	a	a	a	
Sort				
Visible	☑	☑	☑	☑
Function				
Criterion			IS NOT EMPTY	

Figure 321: Editable query using a correlated subquery with an alias

In design view shown in Figure 321 only one table appears. The Media table is given an alias so that the content of the Category_ID field can be accessed using a correlated subquery.

In a query like this, it is now possible to change the foreign key field Category_ID to another category. In the above example the Category_ID field is changed from 0 to 2. The title “The Little Hobbit” is thus assigned to the category “Songwriter”.



	ID	Title	Category_ID	Category
✎	0	Der kleine Hobbit	0	Fantastic
	5	I hear you knocking	1	Rock
	8	Im		
☼	<AutoF			

Record 1

a
*
ID
Title

LibreOffice Base

Error updating the current record

No values were modified.

Figure 322: Error when attempting to change a value in a correlated subquery field

However it is no longer possible to change a value in the field which has received its content via a correlated subquery. An attempted change in Category from ‘Fantasy’ to ‘Fantastic’ is shown in Figure 322. This change is not registered and cannot be saved either. In the table that the design view displays, the Category field is not present.

Use of parameters in queries

If you often use the same basic query but with different values each time, queries with parameters can be used. In principle queries with parameters function just like queries for a subform:

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

This query shows only the media on loan to the reader with the number 2.

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

Now when you run the query, an entry field appears. It prompts you to enter a reader number. Whatever value you enter here, the media currently on loan to that reader will be displayed.

```
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;
```

This query is clearly more user-friendly than the previous one. It is no longer necessary to know the reader’s number. All you need to enter is part of the surname and all media on loan to matching readers are displayed.

If you replace

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

by

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

It no longer matters whether the name is entered in upper or lower case.

If the parameter in the above query is left empty, all versions of LibreOffice up to 4.4 would show **all** readers, since only in Version 4.4 does an empty parameter field read as NULL rather than as an empty string. If you don’t want this behavior, you must use a trick to prevent it:

```
LOWER ("Reader"."LastName") LIKE '%' || IFNULL(NULLIF (LOWER
(:Readername), ''), '§§' ) || '%'
```

The empty parameter field returns an empty string and not a NULL to the query. Therefore the empty parameter field must be assigned the NULL property using NULLIF. Then, since the parameter entry does now yield NULL, it can be reset to a value that normally does not occur in any record. In the above example this is ‘$$’. This value of course will not be found in the search.

From Version 4.4, adaptations to this query technique are necessary:

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

must, in the absence of an entry, inevitably give for the combination:

```
 '%' || LOWER (:Readername) || '%' a the NULL value.
```

To prevent this, add a further condition, that for an empty field, all values are actually shown:

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

The whole thing needs to be put in brackets. Then either a name is searched for or, if the field is empty (NULL from LibreOffice 4.4), the second condition will apply.

When using forms, the parameter can be passed from the main form to a subform. However it sometimes happens that queries using parameters in subforms are not updated, if data is changed or newly entered.

Often it would be nice to alter the contents of list boxes using settings in the main form. So for example, we could prevent library media from being loaned to individuals who are currently banned from borrowing media. Controlling list box settings in this personalized way by using parameters is not possible.

Subqueries

Subqueries built into fields can always only return one record. The field can also return only one value.

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

This query allows data entry (primary key included). The subquery yields precisely one value, namely the total balance. This allows the balance at the till to be read after each entry. This is still not comparable with the supermarket checkout form described in “Queries as a basis for additional information in forms” on page 289. Naturally it lacks the individual calculations of Total * Unit_price, but also the presence of the receipt number. Only the total sum is given. At least the receipt number can be included by using a query parameter:

```
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;
```

In a query with parameters, the parameter must be the same in both query statements if it is to be recognized as a parameter.

For subforms such parameters can be included. The subform then receives, instead of a field name, the corresponding parameter name. This link can only be entered in the properties of the subform, and not when using the Wizard.

Note

Subforms based on queries are not automatically updated on the basis of their parameters. It is more appropriate to pass on the parameter directly from the main form.

Correlated subqueries

Using a still more refined query, an editable query allows you to even carry the running balance for the till:

```
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
```

The Checkout table is the same as Table "a". "a" however yields only the relationship to the current values in this record. In this way the current value of ID from the outer query can be evaluated within the subquery. Thus, depending on the ID, the previous balance at the corresponding time is determined, if you start from the fact that the ID, which is an autovalue, increments by itself.

Note

If the subquery is to be filtered for content using the query editor's filter function, this currently only works if you use double brackets instead of single ones at the beginning and end of the subquery: `((SELECT ….)) AS "Saldo"`

Queries as source tables for queries

A query is required to set a lock against all readers who have received a third overdue notice for a medium.

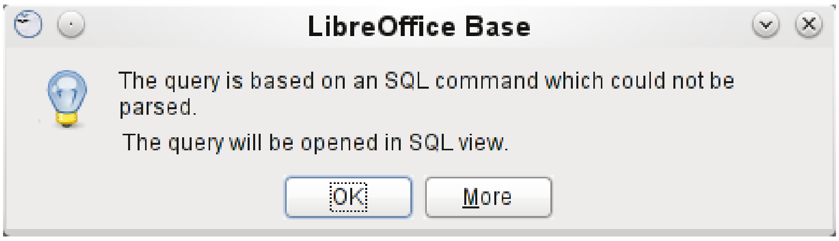
```
SELECT "Loan"."Reader_ID", '3rd Overdue – the reader is blacklisted' AS
"Lock"
FROM
   (SELECT COUNT( "Date" ) AS "Total_Count", "Loan_ID"
      FROM "Recalls" GROUP BY "Loan_ID") AS "a",
   "Loan"
WHERE "a"."Loan_ID" = "Loan"."ID" AND "a"."Total_Count" > 2
```

First let us examine the inner query, to which the outer query relates. In this query the number of date entries grouped by the foreign key Loan_ID is determined. This must not be made dependent on the Reader_ID, as that would cause not only three overdue notices for a single medium but also three media with one overdue notice each to be counted. The inner query is given an alias so that it can be linked to the Reader_ID from the outer query.

The outer query relates in this case only to the conditional formula from the inner query. It shows only a Reader_ID and the text for the Lock field when the "Loan"."ID" and "a"."Loan_ID" are equal and "a"."Total_Count" > 2.

In principle all fields in the inner query are available to the outer one. So for example the sum "a"."Total_Count" can be merged into the outer query to give the actual fines total.

However it can happen, in the Query Design dialog, that the Design View Mode no longer works after such a construction. If you try to open the query for editing again you get the warning shown in Figure 323:



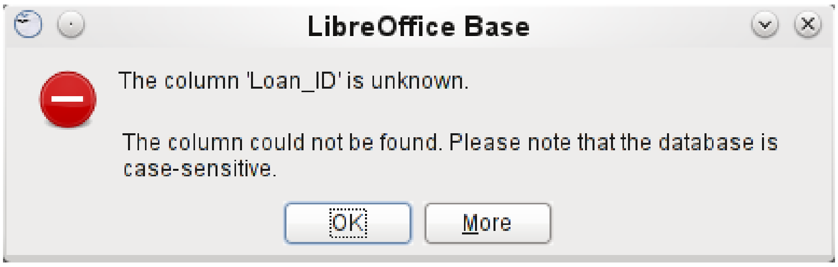
LibreOffice Base

The query is based on an SQL command which could not be parsed.

The query will be opened in SQL view.

Figure 323: Error when trying to open a query that cannot be opened in Design View

If you then open the query for editing in SQL view and try to switch from there into the Design View, you get the error message shown in Figure 324:



LibreOffice Base

The column 'Loan_ID' is unknown.

The column could not be found. Please note that the database is case-sensitive.

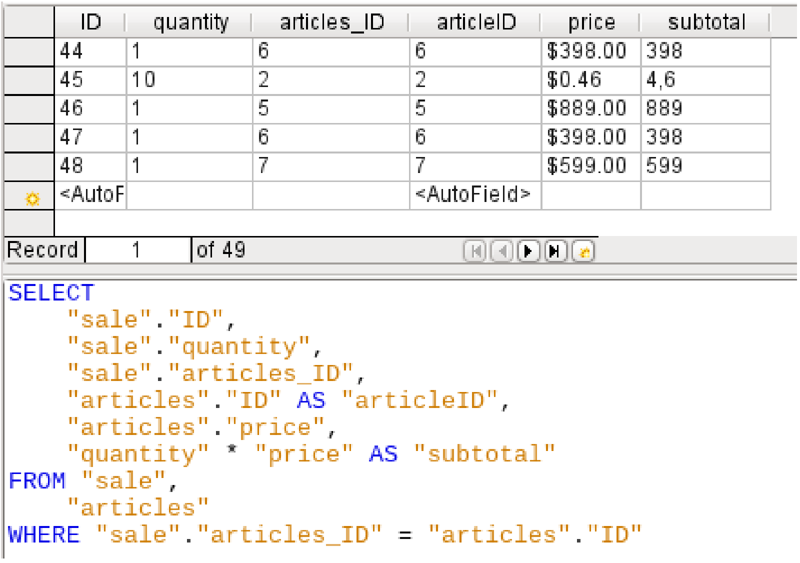
Figure 324: Error when unable to switch from SQL mode to Design View

The Design View Mode cannot find the field contained in the inner query "Loan_ID", which governs the relationship between the inner and outer queries.

When the query is run in SQL Mode, the corresponding content from the subquery is reproduced without error. Therefore you do not have to use direct SQL mode in this case.

The outer query used the results of the inner query to produce the final results. These are a list of the "Loan_ID" values that should be locked and why. If you want to further limit the final results, use the sort and filter functions of the graphical user interface.

The following screenshots show how the different ways to a query result with subqueries can go. Here a query to a stock database is trying to determine what the customer needs to pay at the till. The individual prices are multiplied by the number of items bought giving a subtotal. Then the sum of these subtotals needs to be determined. All this needs to be editable so that the query can be used as the basis for a form.



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
<AutoF			<AutoField>		

Record 1 of 49

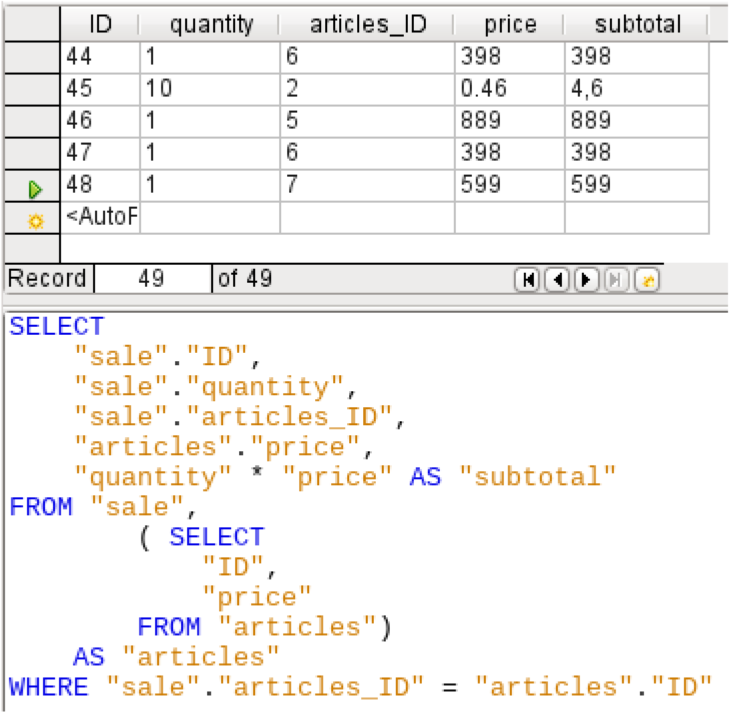
```
SELECT
    "sale"."ID",
    "sale"."quantity",
    "sale"."articles_ID",
    "articles"."ID" AS "articleID",
    "articles"."price",
    "quantity" * "price" AS "subtotal"
FROM "sale",
    "articles"
WHERE "sale"."articles_ID" = "articles"."ID"
```

Figure 325: Simple query to create subtotals

The query shown in Figure 325 uses two tables (*sale* and *articles*). To make the query editable, both primary keys must be included

Note

Because of Bug 61871, Base does not update the partial result automatically.



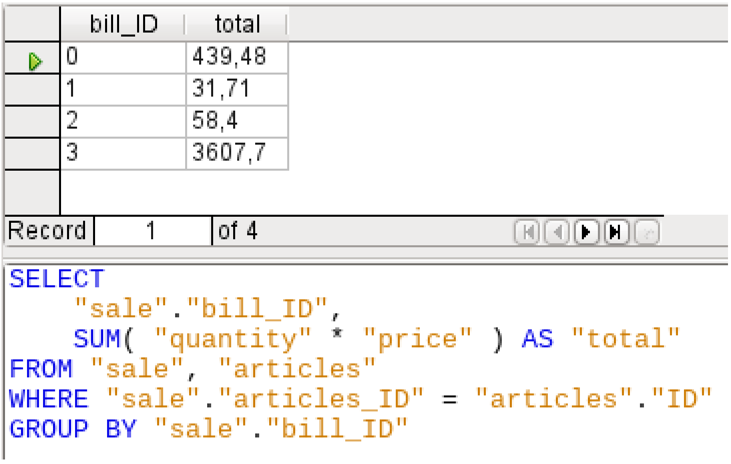
ID	quantity	articles_ID	price	subtotal
44	1	6	398	398
45	10	2	0.46	4,6
46	1	5	889	889
47	1	6	398	398
48	1	7	599	599
<AutoF				

Record 49 of 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"
```

Figure 326: The articles table is moved into a subquery

In Figure 326, the *articles* table is moved into a subquery, which is created in the table area (after the “FROM” term) and given an alias. Now the primary key of the *articles* table is no longer strictly necessary to make the query editable.



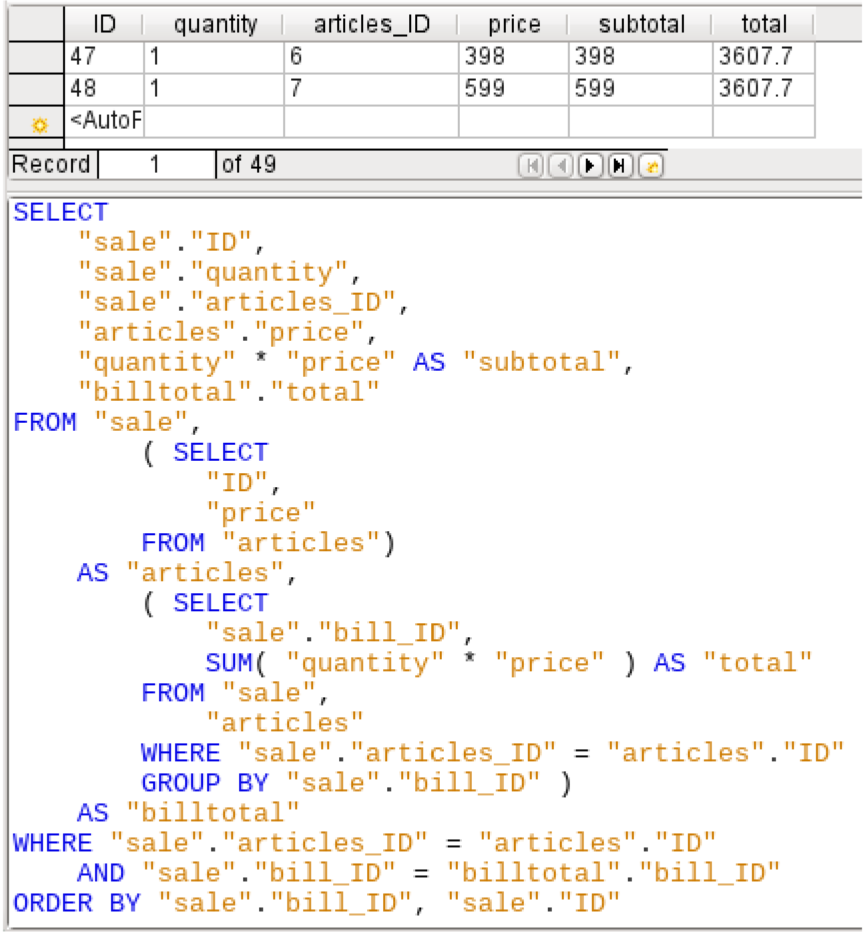
bill_ID	total
0	439,48
1	31,71
2	58,4
3	3607,7

Record 1 of 4

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

Figure 327: Calculating the sales value by bill_id

In Figure 327, the calculated sum must appear in the query. Already the simple query for the calculation sum is not editable so it is grouped and summed here.



	ID	quantity	articles_ID	price	subtotal	total
	47	1	6	398	398	3607.7
	48	1	7	599	599	3607.7
☼	<AutoF					

```
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"
```

Figure 328: Using two subqueries

In Figure 328, with the second subquery the seemingly impossible becomes possible. The previous query is inserted as a subquery into the table definition of the main query (after “FROM”). As a result, the whole query remains editable. In this case entries are only possible in the “Sum” and “WarID” columns. This is subsequently made clear in the query form.

Summarizing data with queries

When data is searched for over a whole database, the use of simple form functions often leads to problems. A form refers after all to only one table, and the search function moves only through the underlying records for this form.

Getting at all the data is simpler when you use queries, which can provide a picture of all the records. The section on “Relationship definition in the query” suggests such a query construction. This is constructed for the example database as follows:

```
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"
```

Here all Titles, Subtitles, and Authors are shown together.

The Media table contains a total of 9 Titles. For two of these titles, there are a total of 8 Subtitles. Without a LEFT JOIN, both tables displayed together yield only 8 records. For each Subtitle, the corresponding Title is searched for, and that is the end of the query. Titles without Subtitle are not shown.

Now to show all Media including those without a Subtitle: Media is on the left side of the assignment, Subtitle on the right side. A LEFT JOIN will show every Title from Media, but only a Subtitle for those that have a Title. Media becomes the decisive table for determining which records are to be displayed. This was already planned when the table was constructed (see Chapter 3, Tables). As Subtitles exist for two of the nine Titles, the query now displays 9 + 8 – 2 = 15 records.

Note

The normal linking of tables, after all tables have been listed, follows the keyword WHERE.

If there is a LEFT JOIN or a RIGHT JOIN, the assignment is defined directly after the two table names using ON. The sequence is therefore always

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

Two Titles of the Media table do not yet have an Author entry or a Subtitle. At the same time one Title has a total of three Authors. If the Author table is linked without a LEFT JOIN, the two Media without an Author will not be shown. But as one medium has three authors instead of one, the total number of records displayed will still be 15.

Only by using LEFT JOIN will the query be instructed to use the Media table to determine which records to show. Now the records without Subtitle or Author appear again, giving a total of 17 records.

Using appropriate Joins usually increases the amount of data displayed. But this enlarged data set can easily be scanned, since authors and subtitles are displayed in addition to the titles. In the example database, all of the media-dependent tables can be accessed.

More rapid access to queries using table views

Views in SQL are quicker than queries, especially for external databases, as they are anchored directly into the database and the server returns only the results. By contrast queries are first sent to the server and processed there.

If a new query relates to another query, the SQL view in Base makes the other query look like a table. If you create a View from it, you can see that you are actually working with a subquery (Select used within another Select). Because of this, a Query 2 that relates to another Query 1 cannot be run by using **Edit > Run SQL** command directly, since only the graphical user interface and not the database itself knows about Query 1.

The database gives you no direct access to queries. This also applies to access using macros. Views, on the other hand, can be accessed from both macros and tables. However, no records can be edited in a view. (They must be edited in a table or form.)

Tip

A query created using Create Query in SQL View has the disadvantage that it cannot be sorted or filtered using the GUI. There are therefore limits to its use.

A View on the other hand can be managed in Base just like a normal table – with the exception that no change in the data is possible. Here therefore even in direct SQL-commands all possibilities for sorting and filtering are available.

In addition, the formatting of columns in a view is retained when the database is closed, unlike columns in a query.

Views are a solution for many queries, if you want to get any results at all. If for example a Subselect is to be used on the results of a query, create a View that gives you these results. Then use the subselect on the View. Corresponding examples are to be found in Chapter 8, Database Tasks.

Creating a View from a query is rather easy and straightforward.

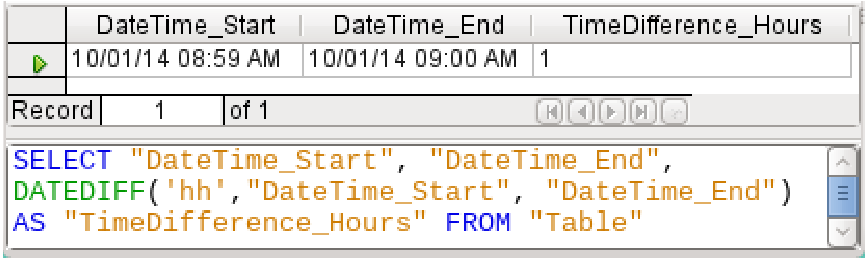
1) Click the **Table** object in the Database section.
2) Click **Create View**.
3) Close the Add Table dialog.
4) Click the **Design View On/Off** icon. (This is the SQL Mode for a View.)
5) Getting the SQL for the View:
    c) Edit the query in SQL View.
    d) Use *Control+A* to highlight the query’s SQL.
    e) Use *Control+C* to copy the SQL.
6) In the SQL Mode of the View, use *Control+V* to paste the SQL.
7) Close, save, and name the View.

Calculation errors in queries

Queries are also used to calculate values. Sometimes the internal HSQLDB database produces apparent errors, which on closer examination turn out to be logically correct interpretations of the data. There are also rounding problems, which can easily cause confusion.

Times within HSQLDB are formatted correctly only up to a difference of 23:59:59 hours. If several times are to be added, for example to calculate hours worked, another way must be found. Here there are several complicated approaches:

- Time is directly expressed only as a total of minutes or even seconds. Advantage: the values allow subsequent problem-free calculation.
- The time is split into hour, minute and second parts and reassembled as text using ‘:’ as a separator. Advantage: the text appears in queries as properly formatted time from the beginning.
- The time is created as a decimal number. A day is 1, an hour is 1/24 and so on. Advantage: the values can subsequently be reformatted as time in the query and presented as a formatable form field.



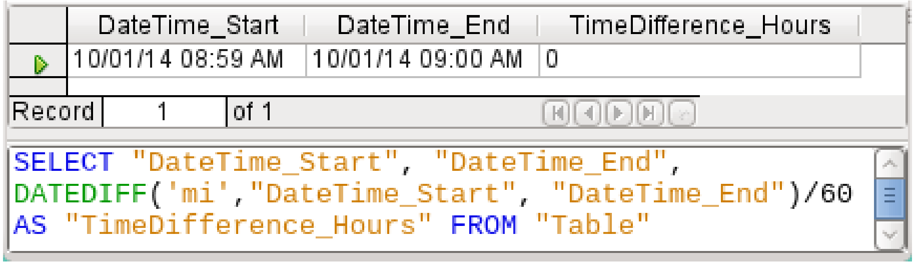
DateTime_Start	DateTime_End	TimeDifference_Hours
10/01/14 08:59 AM	10/01/14 09:00 AM	1

Record 1 of 1

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

Figure 329: Using the DATEDIFF function to calculate a time difference in hours

DATEDIFF allows time intervals to be determined. It explicitly asks for the difference that is to be determined. In the example in Figure 329, minutes have not been requested, but all elements that are greater than a minute are considered. That gives one hour (!) as the difference between 8:59 and 9:00. By contrast, a date difference is calculated as a time difference in hours. If for example the Date_time_end field is set to 2.10.14 09:00, that is calculated as 25 hours.



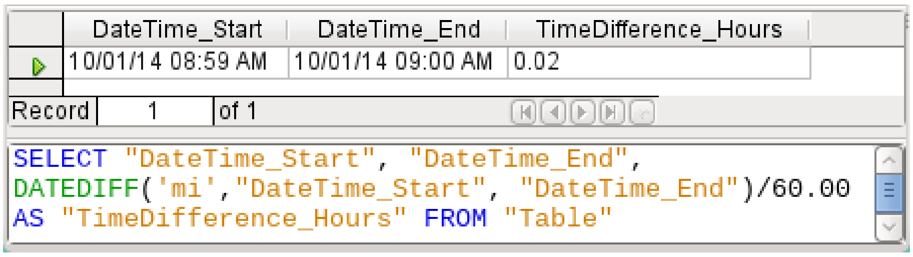
DateTime_Start	DateTime_End	TimeDifference_Hours
10/01/14 08:59 AM	10/01/14 09:00 AM	0

Record 1 of 1

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

Figure 330: Calculate the time difference in minutes and divide by 60

If instead the time interval is calculated in minutes and then divided by 60, the time difference in hours becomes zero as shown in Figure 330. This looks more like the correct value, except: where did that one minute go?



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"
```

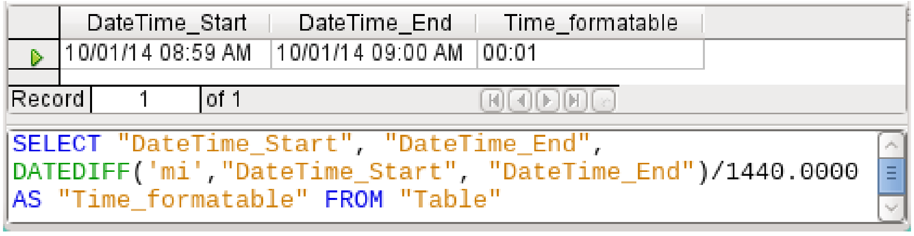
Figure 331: Calculate the time difference in minutes and divide by 60.00

In Figure 331 the time interval has been given as an integer. An integer has been divided by an integer. The result of the query in such cases must also be an integer, not a decimal number. This can easily be corrected. The time difference in minutes is divided by a decimal number with two decimal places (60.00). This gives a result that also has two decimal places. 0.02 hours is still not exactly one minute but is much closer to it than before. The number of decimal places could be increased by using more zeros. A period of 0.016 is a closer approximation still, but later calculation errors cannot always be excluded.

Instead of having to work with lots of added zeros, the data type of DATEDIFF can be influenced directly. Using `(CONVERT(DATEDIFF('mi', "Date_time_start", "Date_time_end"),DECIMAL(50,49)))/60` you can achieve an accuracy of 49 decimal places.

When using calculation functions you must always understand that the data types in HSQLDB have only a limited precision. However many decimal places you use, the fact remains that intermediate results involving a time count can only be used to a limited extent for further calculations.

If a time value is subsequently to be used in a form or report as a formatted time, you must ensure that the day is valid as a basis for the time format.



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

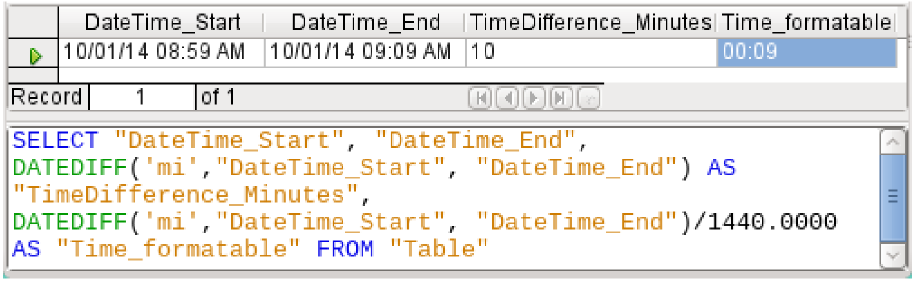
Record 1 of 1

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

Figure 332: Calculate the time difference in minutes and divide by 1440.0000

In Figure 332 the difference is calculated in minutes. The result is given as a fraction of a day. One day has 60 * 24 minutes. If you simply divided by 1440, the result would be zero, so once again you need to give the decimal places explicitly. It then appears as a formatted time of 0 hours and 1 minute.

The format code for a time longer than one day is [HH]:MM. If you use the wrong format, a time difference of 1 day and 1 minute could be shown as only 1 minute.



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

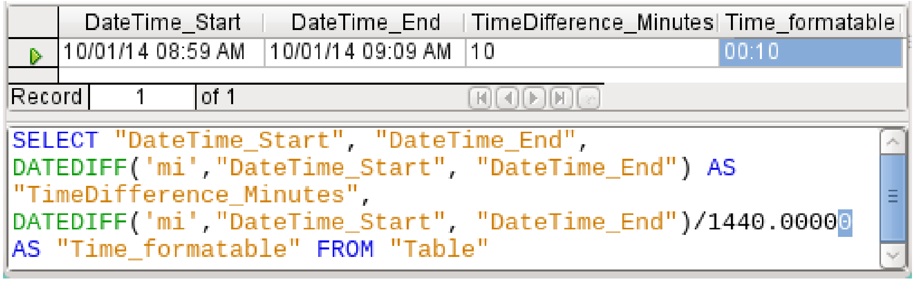
```
SELECT "DateTime_Start", "DateTime_End",
DATEDIFF('mi',"DateTime_Start", "DateTime_End") AS
"TimeDifference_Minutes",
DATEDIFF('mi',"DateTime_Start", "DateTime_End")/1440.0000
AS "Time_formatable" FROM "Table"
```

Figure 333: Calculate the time difference in minutes in addition to the time difference in minutes divided by 1440.0000

The error fiend strikes again! A time difference of 10 minutes should not show up as 9 minutes when correctly formatted as shown in Figure 333. To find out where the problem lies, we need to consider exactly how the calculation is done:

1) 10/1440 = 0.00694. The result is rounded down to 0.0069 because only four decimal places were specified.

2) 0.0069 * 1440 = 9.936 minutes, which is 9 minutes 56.16 seconds. And seconds are not displayed in the chosen format!



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"
```

Figure 334: Calculate the time difference in minutes in addition to the time difference in minutes divided by 1440.00000

Finally, in Figure 334 lengthening the divisor by just one decimal place (from 1440.0000 to 1440.00000) cures this error. Now the rounding is to 0.00694. 0.00694*1440 gives 9.9936 which is 59.616 seconds. The number of seconds is rounded up to 60 seconds, so the 9 minutes have 1 minute added, making 10 minutes in all.

Here too there might be further problems. Might there be further decimal places which, when formatted, do not yield 1 minute? To settle this, a short calculation using Calc with similarly rounded figures can help. Column A contains a sequence of numbers from 1 (for the minutes). Column B contains the formula =ROUND(A1/1440;4) and is formatted to show hours and minutes. If this continues downwards, we can see, next to 10 minutes in column A, 00:09 in column B. Similarly for 28 minutes, etc. If you round to 5 places, these errors disappear.

However nice it might be to have a suitably formatted display in a form, you need to be aware that you are dealing with rounded values which are not suitable for further use in blindly mechanical calculations. If a value needs to be used for further calculation, it is better to use only a representation of time difference in the smallest available units, in this case minutes.

Base Guide 7.3

Chapter 6
Reports

Creating reports using the Report Builder

Reports are used to present data in a way that makes it readily understood by people without knowledge of the database. Reports can:

- Present data in easy-to-read tables
- Create charts for displaying data
- Make it possible to use data for printing labels
- Produce form letters such as bills, recall notices, or notifications to people joining or leaving an association

To create a report requires careful preparatory work on the underlying database. Unlike a form, a report cannot include subreports and thus incorporate additional data sources. Neither can a report present different data elements than those that are available in the underlying data source, as a form can do using list boxes.

Reports are best prepared using queries. In this way all variables can be determined. In particular, if sorting within the report is required, always use a query that makes provision for sorting. This means that queries in direct SQL mode should be avoided under these conditions. If you must use a query of this type in your database, you can carry out the sort by first creating a view from the query. Such a view can always be sorted and filtered using the graphical user interface (GUI) of Base.

Caution

When using the Report Builder, you should frequently save your work during editing. In addition to saving within the Report Builder itself after each significant step, you should also save the whole database.

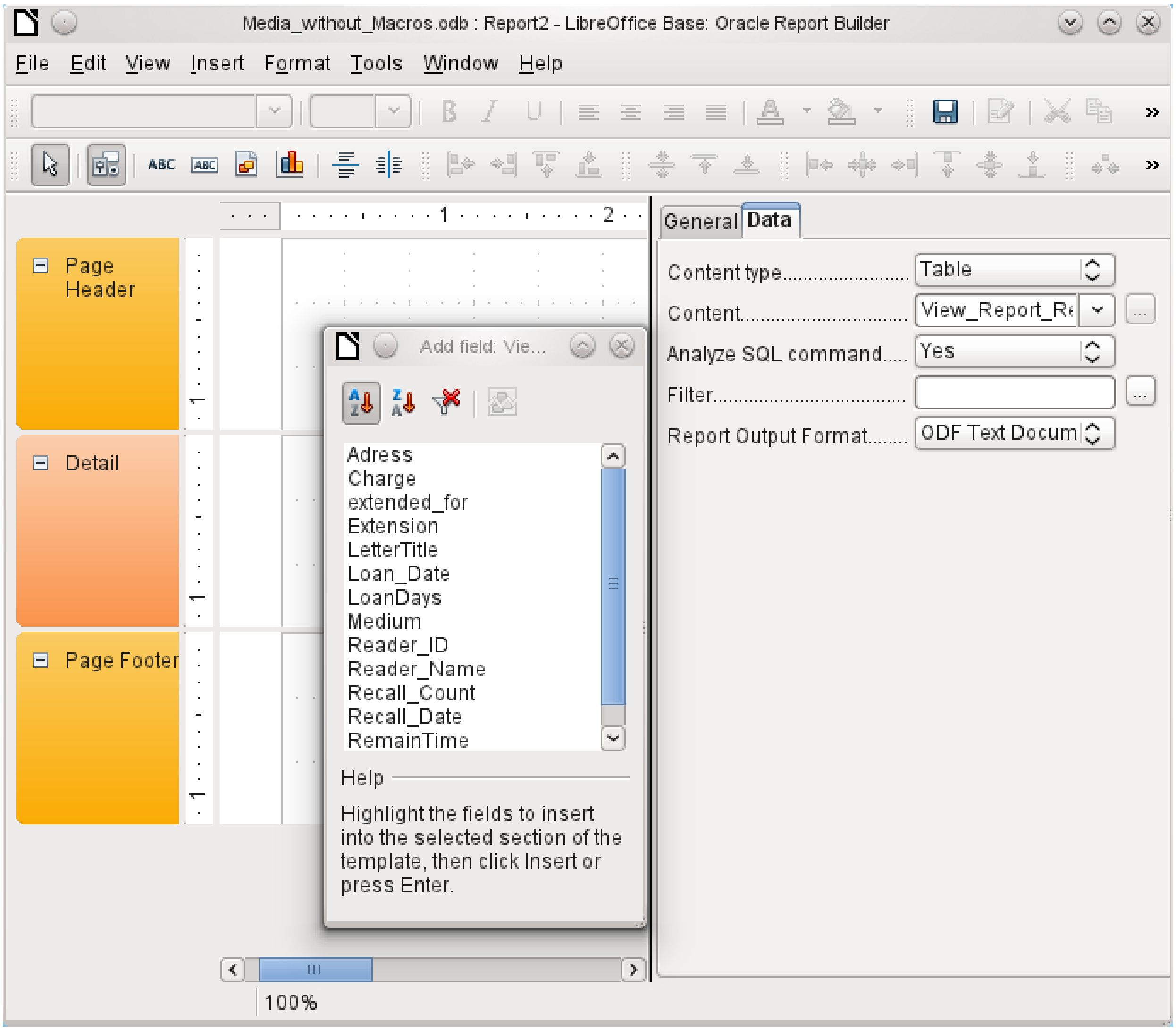
The user interface of the Report Builder

To start the Report Builder from within Base, use **Reports > Create Report in Design View**.

The initial window of the Report Builder (Figure 335) shows three parts. On the left is the current division of the report into Page header, Detail, and Page footer; in the middle are the corresponding areas where the content will be entered; and, to the right, the properties of these regions are shown.

At the same time the Add fields dialog is displayed. This dialog corresponds to the one in form creation. It creates fields with their corresponding field labels.

Without content from the database, a report has no proper function. For this reason, the dialog opens at the Data tab where report content can be set. As an example the View_Report_Recall table is used. As long as Analyze SQL command is set to Yes, the report can be subjected to sorting, grouping, and filtering. A view has been chosen for the basis of this report, so no filter will be applied; it has already been included in the query underlying the view.



Highlight the fields to insert into the selected section of the template, then click Insert or press Enter.

Content type: Table
Content: View_Report_Re
Analyze SQL command: Yes
Filter:
Report Output Format: ODF Text Docum

Figure 335: Initial layout of Report Builder window

Note

The content type provided can be a table, a view, a query or direct SQL coding. The Report Builder works best when it is given data that have been prepared in advance as far as possible. So, for example, calculations in queries can be carried out in advance and the scope of records that are to appear in the report limited where necessary.

Two output formats for reports are available for selection: ODF Text document (a Writer document) or ODF Spreadsheet (a Calc document). If you just want a tabular view of your data, the Calc document should definitely be chosen for your report. It is significantly faster to create and is also easier to subsequently format, as there are fewer options to consider and columns can easily be dragged to the required width afterward.

By default, the Report Builder looks for its data source in the first table in the database. This ensures that at least a test of the functions is possible. A data source has to be chosen before the report can be provided with fields.

The Report Builder provides many additional buttons and these are shown, together with their descriptions, in Figures 336 and 337. The buttons for aligning elements are not further described in this chapter. They are useful for quick adjustment of fields in a single area of the Report Builder, but in principle everything can be done by direct editing of field properties.

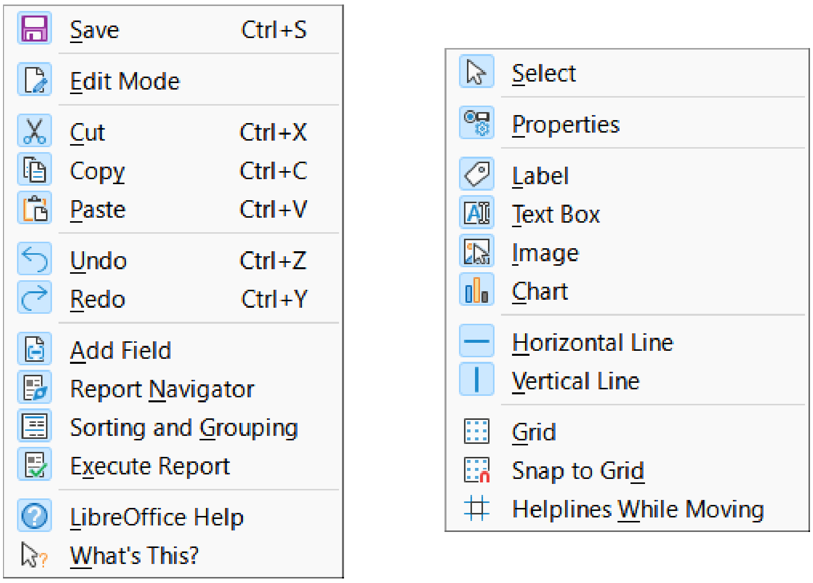


Figure 336: Buttons for editing content - Standard (left) and Report Controls (right) toolbars

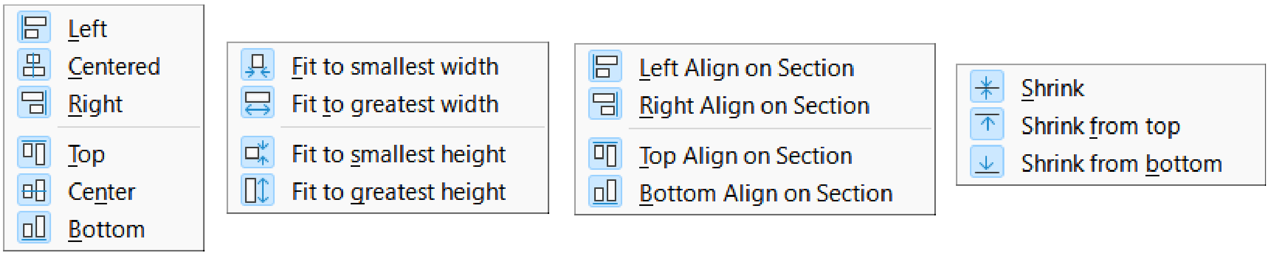


Figure 337: Buttons for aligning elements - Align, Object Resizing, Align at Section, and Shrink at Section toolbars (left to right)

Just as with forms, it is helpful to use the appropriate navigator. So, for example, a careless click at the start of the Report Builder can make it difficult to find the properties of the data for the report. Such data may only be reachable through the report navigator. Left-click on Report and the properties of the report are once more accessible.

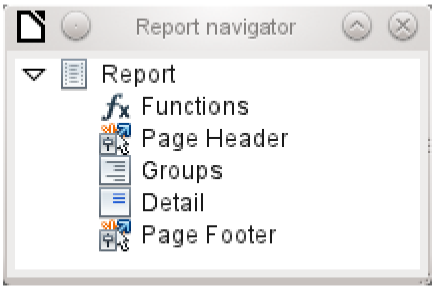
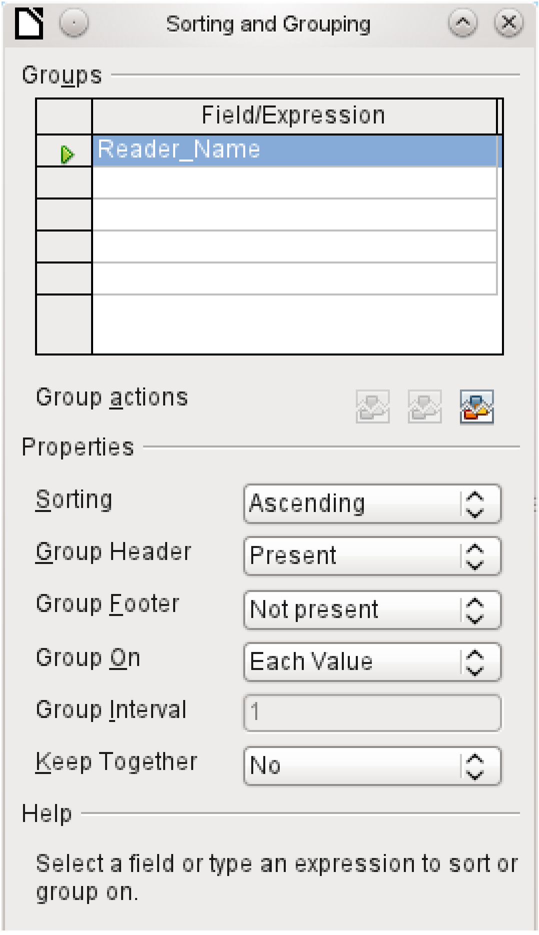


Figure 338: Report navigator

Initially the navigator (see Figure 338) shows the sections of the report, both , in addition to the visible (Page Header, Groups, Detail, and Page Footer) and invisible (Functions). Groups can be used, for example, to assign all media being recalled to the person who has borrowed them, to avoid multiple recall notices. Detail areas show the records belonging to a group. Functions are used for calculations such as sums.

To obtain useful output in the example, the content of the view must be reproduced with suitable grouping. Each reader should be linked to the recall notices for all of their loaned and overdue media.

View > Sorting and Grouping or the corresponding button starts the grouping function.



Sorting and Grouping

Groups

	Field/Expression
▷	Reader_Name

Group actions

Properties

Sorting	Ascending
Group Header	Present
Group Footer	Not present
Group On	Each Value
Group Interval	1
Keep Together	No

Help

Select a field or type an expression to sort or group on.

Figure 339: Sorting and Grouping dialog

Here grouping and sorting are by the Reader_Name field. Additional fields could also be included in the table above. For example, if you also want to group and sort by the Loan_Date field, choose this as the second line.

Directly under the table, several grouping actions are available for selection. You can move a group up or down the list or completely remove it. As only one group is necessary for the planned report, Figure 339 shows only the Delete symbol at the extreme right of the group actions as available.

The Sorting property is self-explanatory.

When the entry was created, the left side of the Report Builder immediately showed a new division. Next to the field description Reader_Name you can now see Header. This section is for the group header in the report. The header might contain the name of the person who will receive the recall notice. In this case there is no group footer. Such a footer could contain the fine due, or the place and current date and a space for the signature of the person sending the notice.

By default there is a new group for each value. So if the Reader_Name changes, a new group is started. Alternatively you can group by initial letter. In the case of a recall notice, however, this would put all readers with the same initial together in one group. Schmidt, Schulze, and Schulte would receive a common recall notice, which would be quite pointless in this example.

When grouping by initial letter, you can additionally specify how many letters later the next group should begin. One can imagine for example a grouping for a small telephone directory. According to the size of the contact list, one might imagine a grouping on every second initial letter. So A and B would form the first group, then C and D, and so on.

A group can be set either to be kept together with the first details section, or, as far as possible, as a complete group. By default, this option is set to No. For recall notices, you would probably want the group to be arranged so that a separate page is printed for each person who is to receive a recall letter. In another menu, you can choose that each group (in this case, each reader name) be followed by a page break before dealing with the next value.

If you have chosen to have a group header and perhaps a group footer, these elements will appear as sections in the report navigator under the corresponding fieldname Reader_Name. Here too you have the possibility of using functions, which will then be limited to this group.

To add fields, use the Add field function, as with forms. However in this case, the label and the field contents are not tied together. Both can be independently moved, changed in size and dragged to different sections.

Figure 340 shows the report design for the recall notice. In the page header is the heading Libre Office Library, inserted as a label field. Here you could also have a letterhead with a logo, since graphics can be included. This level is called Page Header, but that does not imply there is no space above it. That depends on the page settings; if an upper margin has been set, it lies above the page header.

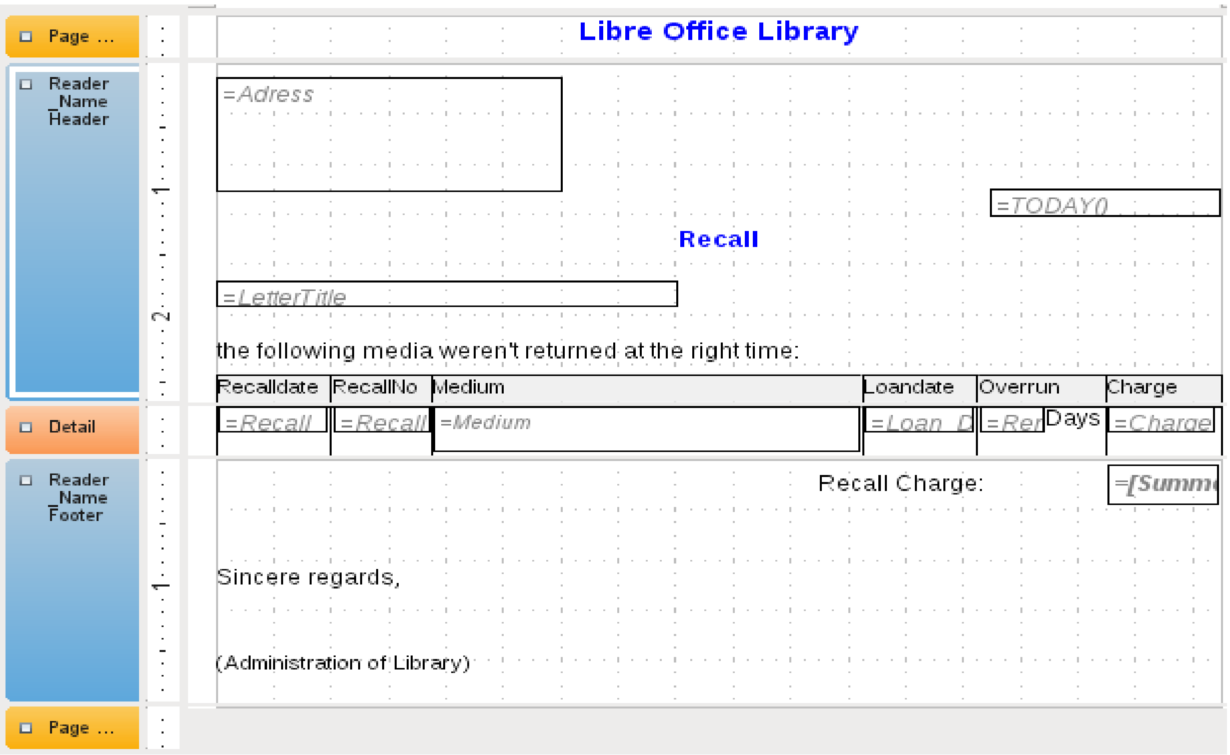
Reader_Name Header is the header for the grouped and sorted data. In the fields that are to contain data, the names of the corresponding data fields are shown in light gray. So, for example, the view underlying the report has a field named Address, containing the complete address of the recipient with street and town. To put this into a single field requires line breaks in the query. You can use `CHAR(13)||CHAR(10)` to create them.

Example:

```
SELECT "Salutation"||CHAR(13)||CHAR(10)||"FirstName"||' '||"LastName"||
CHAR(13)||CHAR(10)||"Street"||' '||"No"||CHAR13||CHAR(10)||"Postcode"||'
'||"Town" AS "Adress" FROM "Reader"
```

The =TODAY() field represents a built-in function, which inserts the current date into this position.

In Reader_Name Header, in addition to the salutation, we see the column headings for the following table view. These elements should appear only once, even if several media are listed.



the following media weren't returned at the right time:

Recalldate	RecallNo	Medium	Loandate	Overrun	Charge
=Recall	=Recall	=Medium	=Loan_D	=Rer Days	=Charge

Sincere regards,

(Administration of Library)

Figure 340: Report design for example recall notice

In the background of these column headers is a gray rectangle, which also serves as a frame for the data.

The Details area is repeated as often as there are separate records with the same Reader_Name data. Here are listed all media that have not been returned on time. There is another rectangle in the background to frame the contents. This rectangle is filled with white rather than gray.

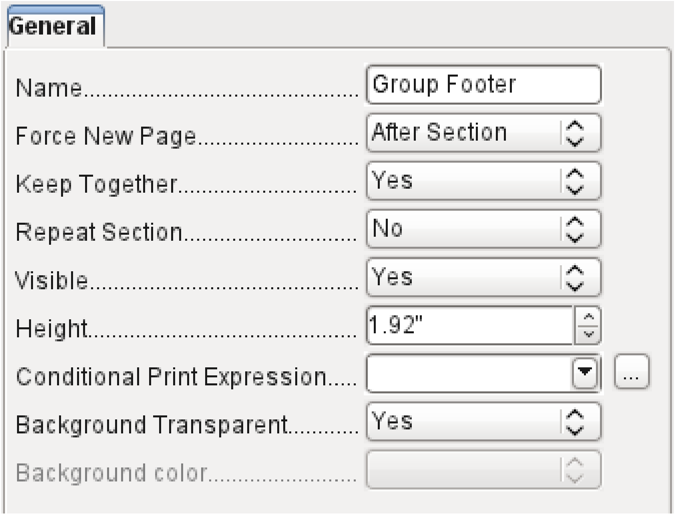
Note

In principle LibreOffice provides for the possibility of adding horizontal and vertical lines. These lines have the disadvantage that they are interpreted only as hairlines. They can be reproduced better if rectangles are used. Set the background of the rectangle to black and the size to, for example, 17cm wide and 0.03cm high. This will create a horizontal line with a thickness of 0.03cm and a length of 17cm.

However, this variant also has a disadvantage: graphical elements cannot be positioned correctly if the area stretches over more than one page.

The Reader_Name Footer closes the letter with a greeting formula and an area for the signature. The footer is so defined that an additional page break will occur after this area. Also, contrary to the default setup, it is specified that this area should be kept together in all cases. After all, it would look rather odd if many recall notices had the signature on a separate page.

Keep together refers here to the page break. If you want the content of a record to be kept together independently of the break, this is only possible at present if the record is not read in as Details but is used as the basis for a grouping. You can choose Keep together = Yes, but it does not work; the Details area becomes separated. You have to put the content of Details in a separate group to keep it together.

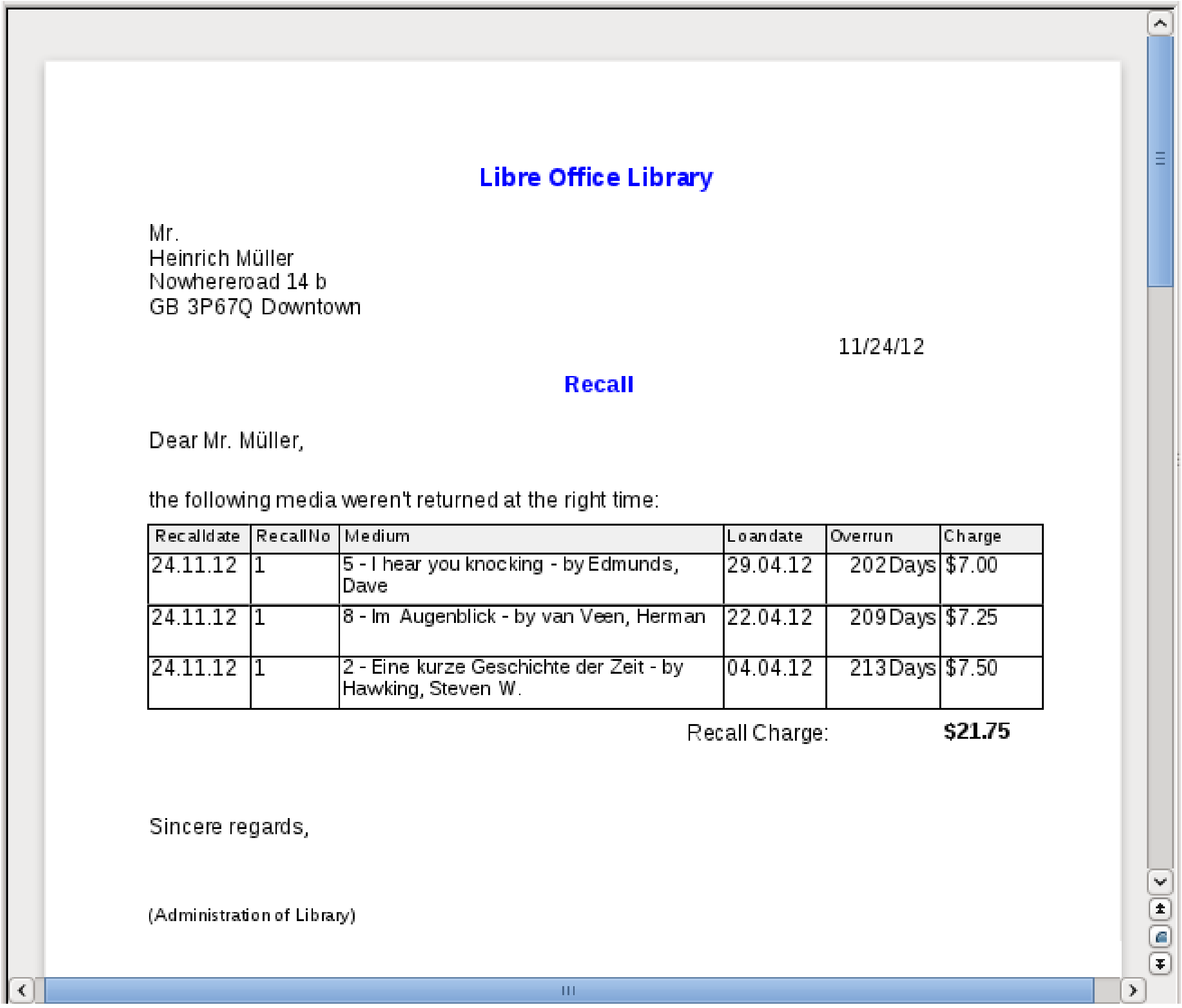


Name	Group Footer
Force New Page	After Section
Keep Together	Yes
Repeat Section	No
Visible	Yes
Height	1.92"
Conditional Print Expression	
Background Transparent	Yes
Background color	

Figure 341: Properties for the Reader_Name Footer

A built-in function is used for calculating the total fines.

A recall notice is shown in Figure 342. The details area contains media items that the reader has taken out on loan. The group footer contains the total fine due.



Libre Office Library

Mr.
Heinrich Müller
Nowhereroad 14 b
GB 3P67Q Downtown

11/24/12

Recall

Dear Mr. Müller,

the following media weren't returned at the right time:

Recalldate	RecallNo	Medium	Loandate	Overrun	Charge
24.11.12	1	5 - I hear you knocking - by Edmunds, Dave	29.04.12	202 Days	$7.00
24.11.12	1	8 - Im Augenblick - by van Veen, Herman	22.04.12	209 Days	$7.25
24.11.12	1	2 - Eine kurze Geschichte der Zeit - by Hawking, Steven W.	04.04.12	213 Days	$7.50

Recall Charge: **$21.75**

Sincere regards,

(Administration of Library)

Figure 342: Recall notice report

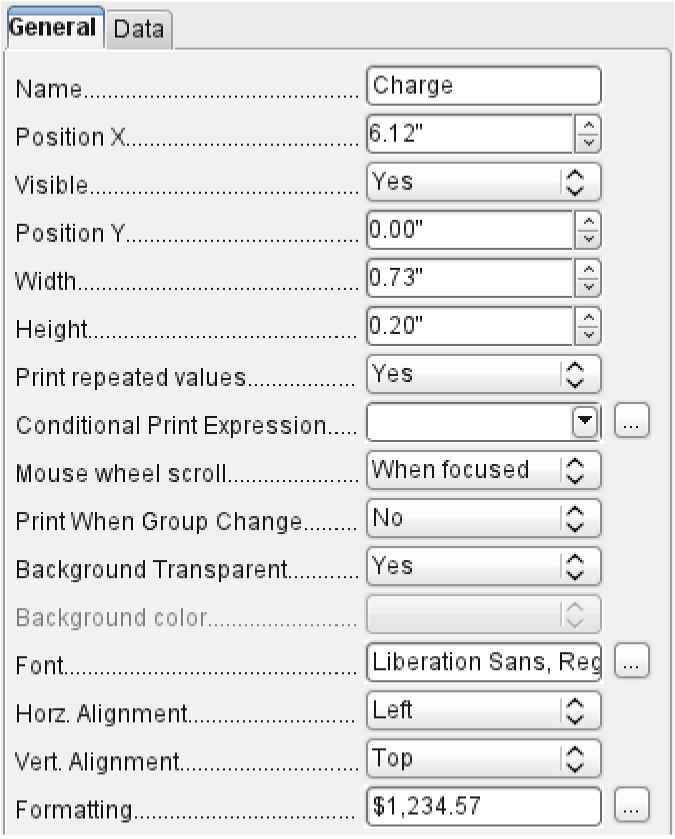
Note

Reports for single records can also extend over more than one page. The size of the report is quite separate from the page size. However, stretching the details area over more than one page can lead to faulty breaks. Here the Report Builder still has problems in calculating the spacing correctly. If both grouping areas and graphical elements are included, this may result in unpredictable sizes for certain areas.

So far individual elements can be moved to positions outside the size of a single page only with the mouse and cursor keys. The properties of the elements always provide the same maximum distance from the upper corner of any area that lies on the first page.

General properties of fields

There are only three types of field for the presentation of data. In addition to text fields (which, contrary to their name, can also contain numbers and formatting), there is also a field type that can contain images from the database. The Charge field displays a summary of data in Figure 343.



General | Data

Name	Charge
Position X	6.12"
Visible	Yes
Position Y	0.00"
Width	0.73"
Height	0.20"
Print repeated values	Yes
Conditional Print Expression	
Mouse wheel scroll	When focused
Print When Group Change	No
Background Transparent	Yes
Background color	
Font	Liberation Sans, Reg
Horz. Alignment	Left
Vert. Alignment	Top
Formatting	$1,234.57

Figure 343: Properties (General tab) for the Charge field

As with forms, fields are given names. By default, the name is that of the underlying database field.

A field can be set to be invisible. This may seem a bit pointless in the case of fields but is useful for group headers and footers, which may be required to carry out other functions of the grouping without containing anything that needs to be displayed.

If Print repeated values is deactivated, display of the field is inhibited when a field with the same content is loaded directly before. This functions correctly only for data fields that contain text. Numeric fields or date fields ignore the deactivation instruction, Label fields are completely faded out when deactivated, even if they occur only once.

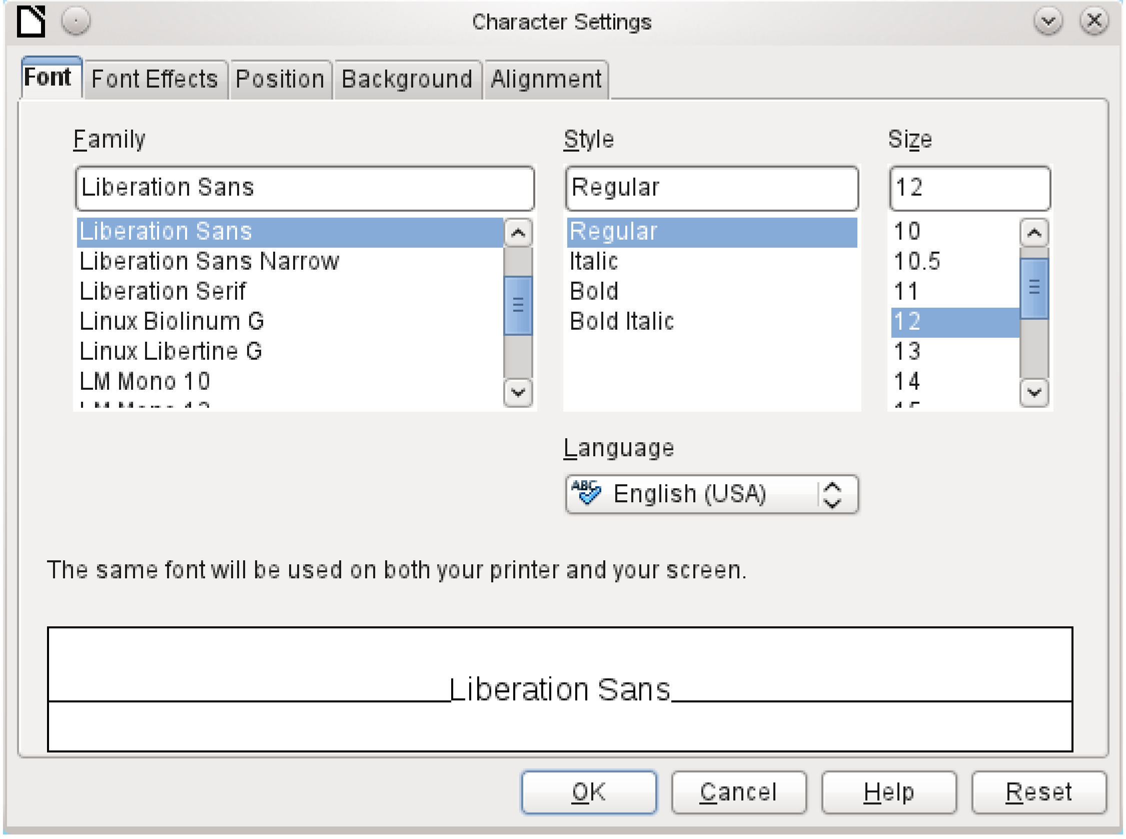
In the Report Builder the display of certain content can be inhibited by using Conditional Print Expression or the value of the field can be used as a base for formatting text and background. More on conditional expressions is given in “Conditional print” on page 330.

The setting for the mouse wheel has no effect because report fields are not editable. It seems to be a leftover from the form definition dialog.

The Print When Group Change function could not be reproduced in reports either.

If the background is not defined as transparent, a background color can be defined for each field.

The other entries deal with the internal content of the field in question. This covers the font (for font color, font weight, and so on, see Figure 344), the alignment of the text within the field, and formatting with the corresponding Character dialog (see Figure 345).



Family: Liberation Sans

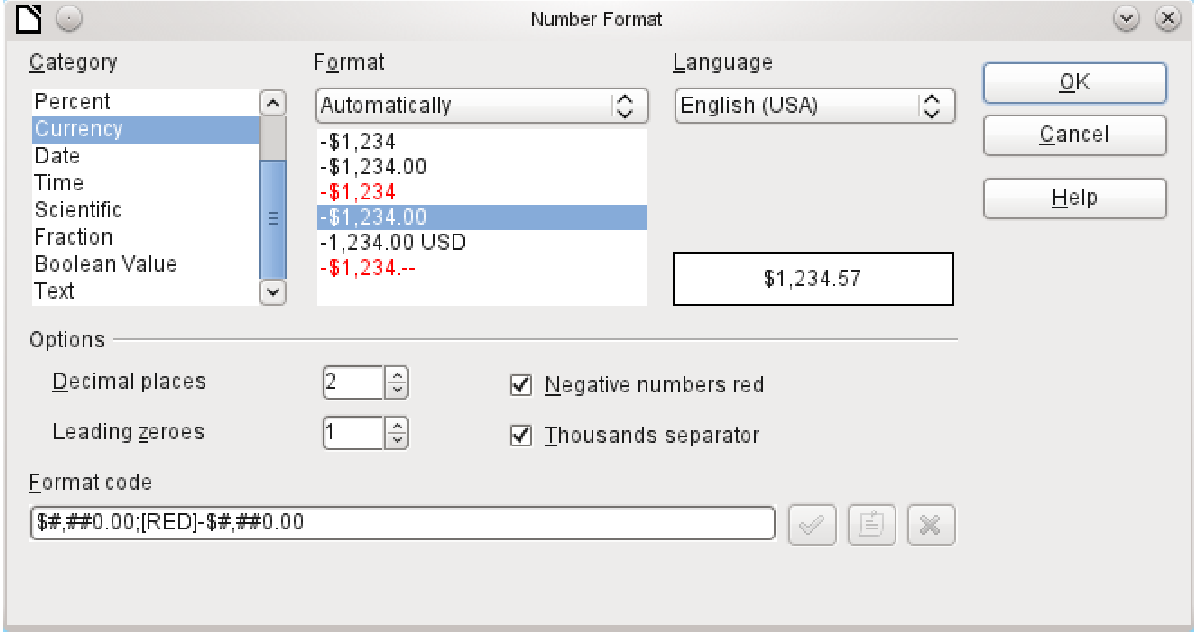
Style: Regular

Size: 12

Language: English (USA)

The same font will be used on both your printer and your screen.

Figure 344: Character Settings dialog



Category: Currency

Format: Automatically

Language: English (USA)

$1,234.57

Options

Decimal places: 2

Leading zeroes: 1

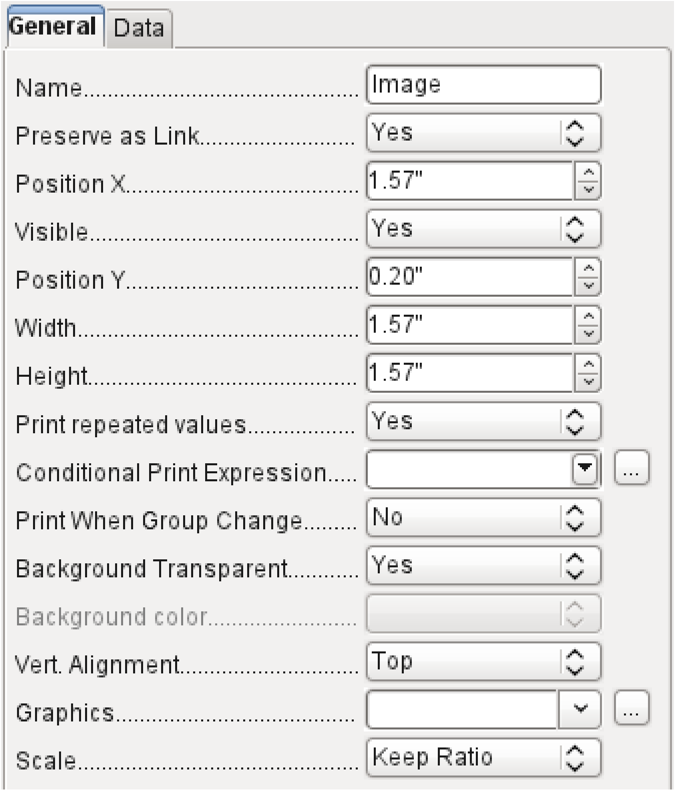
☑ Negative numbers red

☑ Thousands separator

Format code: $#,##0.00;[RED]-$#,##0.00

Figure 345: Number Format dialog

Special properties of graphical controls



Name	Image
Preserve as Link	Yes
Position X	1.57"
Visible	Yes
Position Y	0.20"
Width	1.57"
Height	1.57"
Print repeated values	Yes
Conditional Print Expression	
Print When Group Change	No
Background Transparent	Yes
Background color	
Vert. Alignment	Top
Graphics	
Scale	Keep Ratio

Figure 346: Properties (General tab) for an image control

A graphical control (see Figure 346) can contain graphics from both inside and outside the database. However, it is not possible at present to store a graphic such as a logo permanently in Base. Therefore it is essential that the graphic is available in the search path, even when you are presented with the choice of embedding rather than linking images and the first field Set up as link can be set (literally closed) to a corresponding planned functionality. This is one of several functions that are planned for Base and are in the GUI but have not actually been implemented yet—so the buttons and checkboxes have no effect.

Alternatively, of course, a graphic can be stored in the database itself and so becomes available internally. But in that case, it must be accessible through one of the fields in the query underlying the report.

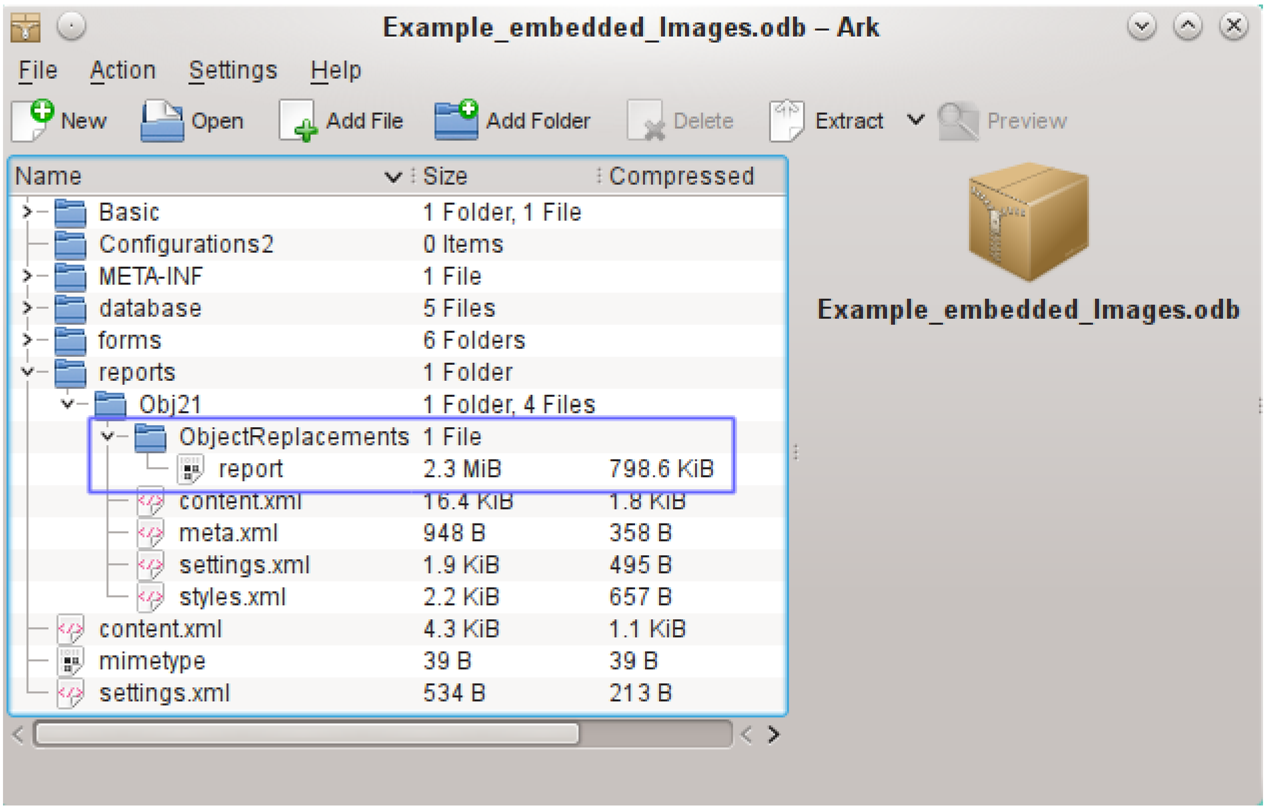
To take up an external graphic, use the selection button beside the Graphic field to load it. To load a graphical database field, specify the field under the Data tab.

The vertical alignment setting does not seem to have any effect during the design stage. When you call up the report, however, the graphic appears in the correct position.

When scaling, you can select *No, Keep aspect ratio*, or *Autom. Size*. This corresponds to the settings for a form:

- **No**: The image is not fitted to the control. If it is too large, a cropped version is shown. The original image is not affected by this.
- **Keep aspect ratio**: The image is fitted to the control but not distorted.
- **Automatic size**: The image is fitted to the control and in some cases may be distorted.

When reports containing images are edited, it can happen that the database becomes significantly larger. Inside the *.odb file shown in Figure 347, Base places in the report directory an ObjectReplacements folder, for reasons that are not fully understood. This folder contains a “report” file that is responsible for the enlargement.



Name	Size	Compressed
Basic	1 Folder, 1 File	
Configurations2	0 Items	
META-INF	1 File	
database	5 Files	
forms	6 Folders	
reports	1 Folder	
Obj21	1 Folder, 4 Files	
ObjectReplacements	1 File	
report	2.3 MiB	798.6 KiB
content.xml	16.4 KiB	1.8 KiB
meta.xml	948 B	358 B
settings.xml	1.9 KiB	495 B
styles.xml	2.2 KiB	657 B
content.xml	4.3 KiB	1.1 KiB
mimetype	39 B	39 B
settings.xml	534 B	213 B

Figure 347: Contents of an ODB file viewed with the Ark file archive manager

If the database is opened in an archiving program, this folder with its content is visible in the reports subdirectory. You can safely use the archiving program to find and delete the folder.

Note

If reports are not going to be repeatedly edited, deleting the ObjectReplacements folder once is good enough. The size of this folder can grow very rapidly. This depends on the number and size of the included files. A test showed that a single 2.8MB jpg file enlarged the *.odb file by 11MB! See Bug 80320.

Incorporating charts into the report

You can insert charts into a report by using the corresponding control or with **Insert > Report controls > Chart**. A chart is the only way to reproduce data that is not found in the data source specified for the report. A chart can therefore be seen as a kind of subreport, but also as a free-standing component of the report.

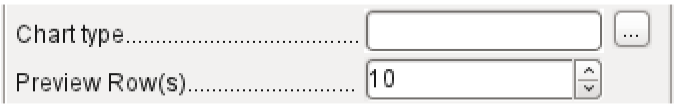


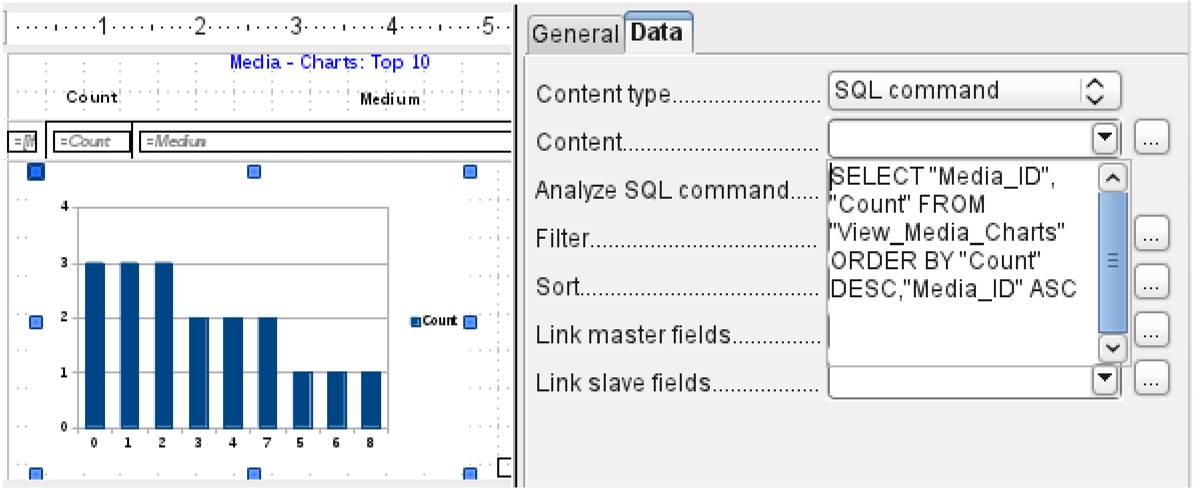
Chart type..

Preview Row(s)........................... 10

Figure 348: Select chart type on Properties (General tab)

You must draw the place for the chart using the mouse. In the general properties (see Figure 348), in addition to the familiar fields, you can choose a Chart type (see the corresponding types in Calc). In addition, you can set a maximum number of records for the preview, which will give an impression of how the chart will finally look.

Charts can be formatted in the same way as in Calc (double-click on the chart). For further information, see the description in the *LibreOffice Calc Guide*.



Content type: SQL command

Content:

SELECT "Media_ID", "Count" FROM "View_Media_Charts" ORDER BY "Count" DESC,"Media_ID" ASC

Analyze SQL command

Filter

Sort

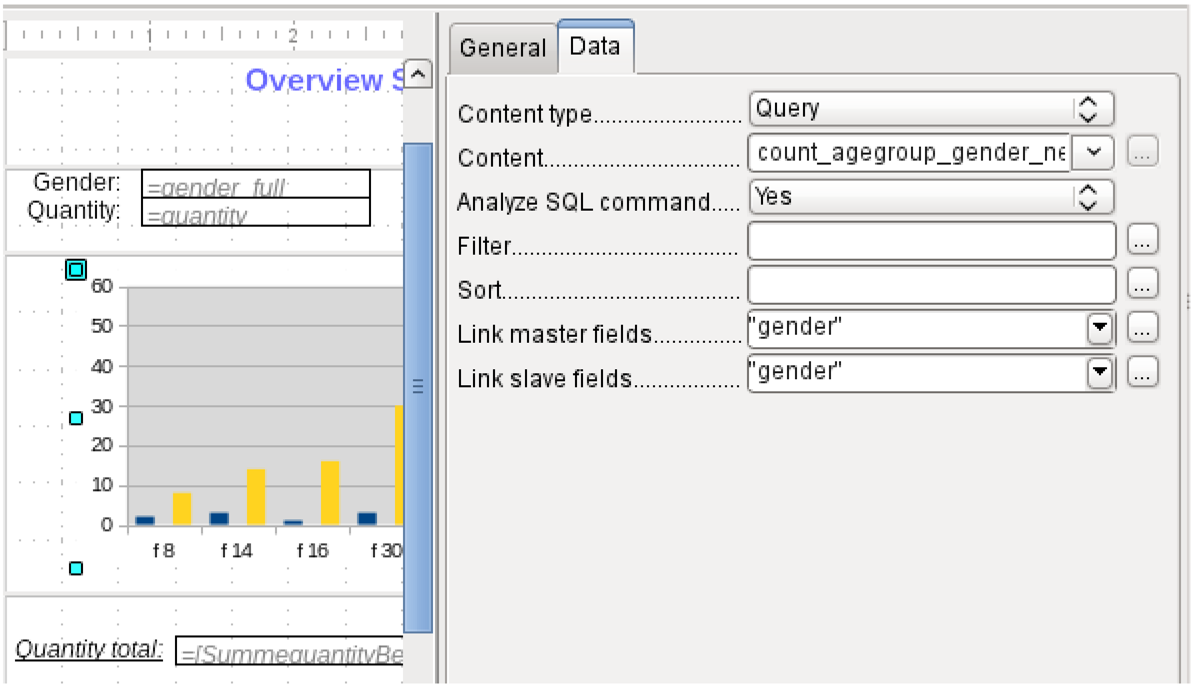
Link master fields

Link slave fields

Figure 349: Using a SQL command to populate a chart showing the frequency with which media items are borrowed

The chart is linked in the Data section with the necessary data fields. The Media Top 10 list example in Figure 349 chart shows the frequency with which particular media are borrowed. The Query Editor is used to create a suitable SQL command, as you would do for a listbox in a form. The first column in the query will be used to provide the labels for the vertical bars in the chart, while the second column yields the total number of loan transactions, shown in the height of the bars.

In the example above, the chart shows very little at first, since only limited test loans were carried out before the SQL command was issued.



Content type: Query

Content: count_agegroup_gender_ne

Analyze SQL command: Yes

Filter:

Sort:

Link master fields: "gender"

Link slave fields: "gender"

Figure 350: Populating a chart using a query

A chart can be populated using a query in the Data properties as shown in Figure 350. This chart from the database Example_sport.odb shows something special in addition to the basics of creating charts in reports: the preview of the chart shows more columns than was anticipated.

This results from the content of the query, which creates additional columns that will not all appear in the chart itself.

Note

The database Example_sport.odb is included in the example databases package for this handbook.

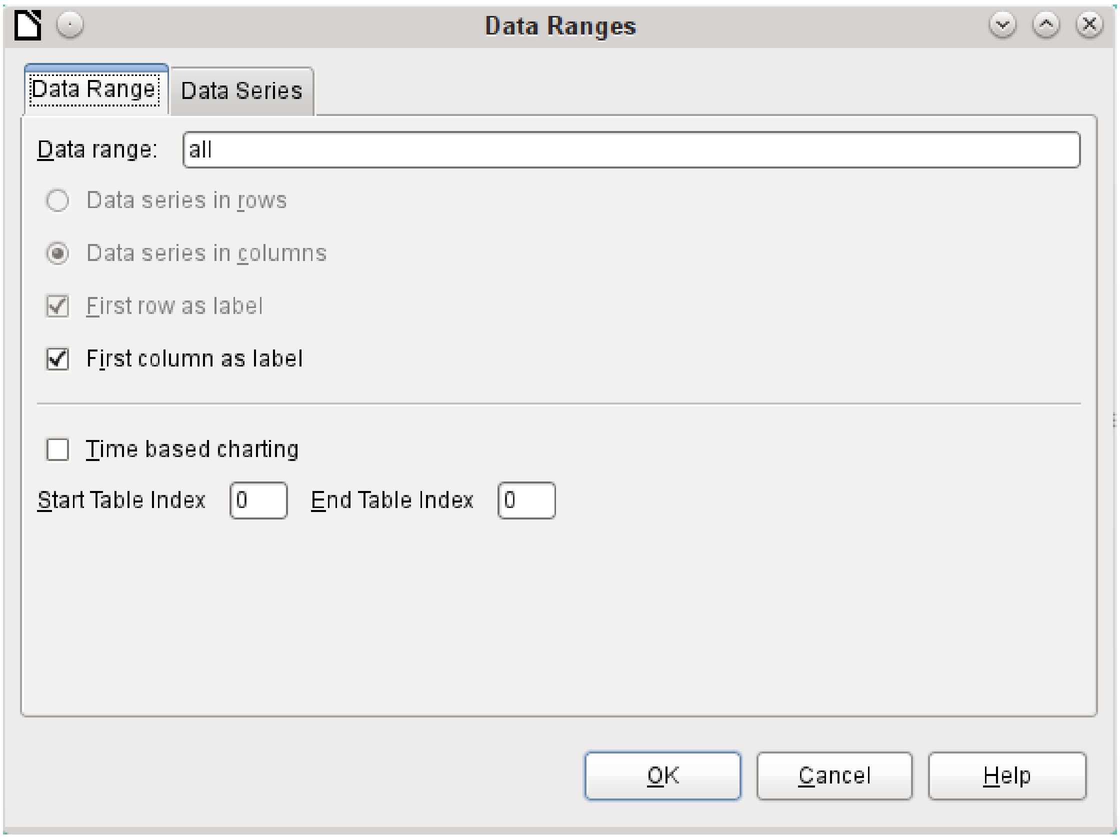
A filter and sort with the internal tools of the Report Builder is not necessary, as this has already been done as far as possible within the query.

Tip

Basically you want to remove as many tasks as possible from the creation of your reports. Whatever can be managed early in the process by using queries does not need to be done again during the relatively slow process of creating the report itself.

As with main forms and subforms, some fields are now linked together. In the actual report, the age groups for male and female sport camp participants are listed as a table. They are grouped by gender. In each group, there is now a separate chart. To ensure that the chart only uses data for the correct gender, the two fields called “Gender” – in the report and in the chart – are linked together.

The X-axis of the chart is automatically linked to the first column in the data source table. If there are more than two columns in the table, additional columns are automatically put into the chart. Further settings for the chart can be accessed by selecting the whole chart with a double-click. Clicking the right mouse button opens a context menu above the diagram, its content depending on which element has been selected. It contains possible settings for the data ranges shown in Figure 351:



Data Ranges

Data Range | Data Series

Data range: all

○ Data series in rows

● Data series in columns

☑ First row as label

☑ First column as label

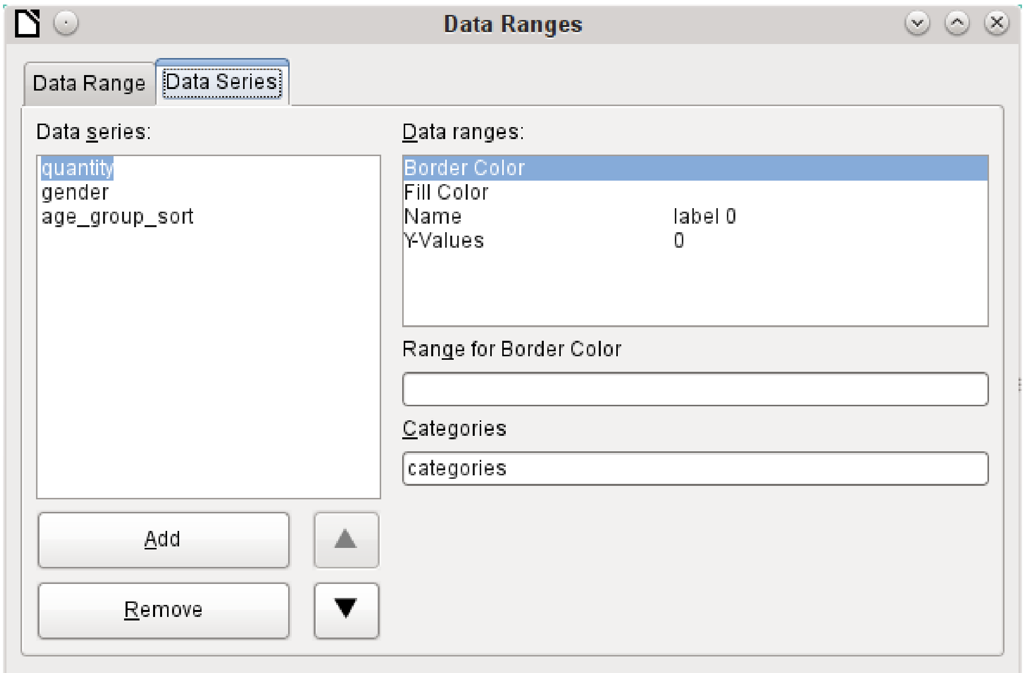
□ Time based charting

Start Table Index 0 End Table Index 0

Figure 351: Data Ranges dialog (Data Range tab)

Data series in columns is grayed out and therefore cannot be changed. Nor can you change the checkbox *First row as label*. The remaining settings on the Data Range tab should not be altered, as there are more possibilities here than the Report Builder can actually handle.

The Data Series tab, on the other hand, hides a couple of settings that can significantly change the default appearance of your chart. It shows all data series that are available for the first column of the query. Any that you do not want to display can be removed at this point.



Data series:

- quantity
- gender
- age_group_sort

Data ranges:

Border Color	
Fill Color	
Name	label 0
Y-Values	0

Range for Border Color

Categories: categories

Figure 352: Data Ranges dialog (Data Series tab)

In the example shown in Figure 352, there were too many columns visible in the chart. This calls for improvement! Neither gender nor age_group_sort, whose names come from the underlying query, are of any use here. The gender series is used to bind the chart with the data source of the report and cannot in any case be represented numerically. And age_group_sort serves only to ensure a correct sort of the values in the query, since otherwise a code like “m8” would come immediately before “m80” instead of at the beginning (sorting in text fields often leads to similar undesirable results).

When all rows have been removed before the Total row, the preview of the chart looks as shown in Figure 353:

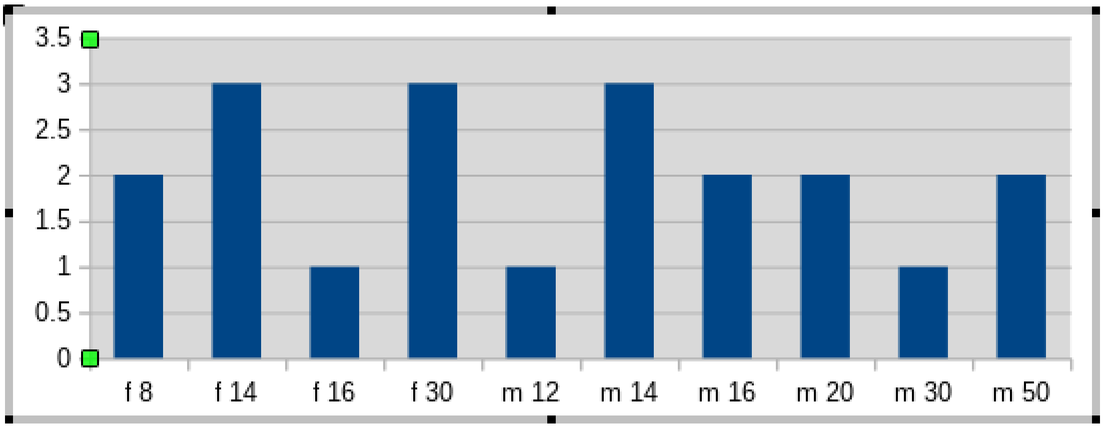


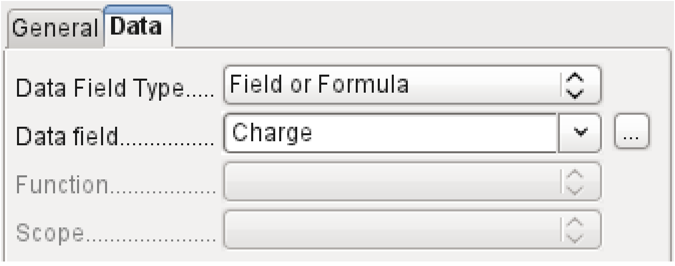
Figure 353: Preview of chart showing the number of competition entrants for each gender / age group combination

This preview shows 10 columns – the first ten columns of the query. In the actual run, only those columns will be displayed that belong to the correct gender: the "m" columns for males and the "f" ones for females.

The Y-axis still shows an unfortunate feature. After all, there is no such thing as half a person! Again, this could be improved. However in an automated run with this setting, these numbers will be changed to integers if the range of values does not, as in the above example, stop short at '3'. If this automatic processing were to be switched off, then some manual improvement would indeed be necessary.

All further settings are similar to those that Calc uses for chart creation.

Data properties of fields



Data Field Type..... Field or Formula
Data field............... Charge
Function.................
Scope.....................

Figure 354: Properties (Data tab) for the Charge field

In the properties dialog, the Data tab in Figure 354 shows by default only the database field from which the data for this report field will be read. However, in addition to the field types Field and Formula, the types Function, Counter, and User-defined function are available.

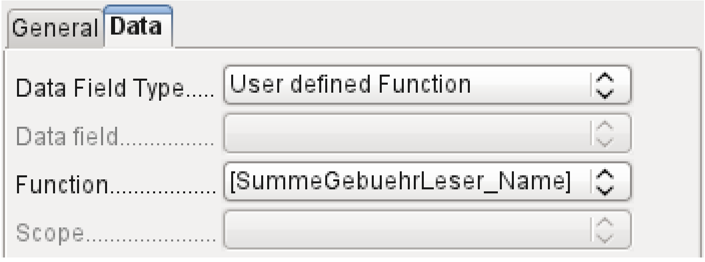
You can select in advance the Sum, Minimum, and Maximum functions. They will apply either to the current group or to the whole report. These functions can lead to problems if a field is empty (NULL). In such fields, if they have been formatted as numbers, NaN appears; that is, there is no numerical value present. For empty fields, no calculation is carried out and the result is always 0.

Such fields can be reformatted to display a value of 0 by using the following formula in the Data area of the view.

```
IF([numericfield];[numericfield];0)
```

This function calculates with the actual value of a field that has no value. It would seem simpler to formulate the underlying query for the report so that 0 is given instead of NULL for numeric fields.

The Counter counts only the records that will occur either in the group or in the report as a whole. If the counter is inserted into the Details area, each record will be provided with a running number. The numbering will apply only to records in the group or in the whole report.



Data Field Type..... User defined Function
Data field...............
Function................. [SummeGebuehrLeser_Name]
Scope.....................

Figure 355: Properties (Data tab) with the Data Field Type set to User defined Function

Finally the detailed User-defined Function is available as shown in Figure 355. It may happen that the Report Builder itself chooses this variant if a calculation has been requested, but for some reason it cannot correctly interpret the data source.

Functions in the Report Builder

The Report Builder provides a variety of functions, both for displaying data and for setting conditions. If these are not sufficient, user-defined functions can be created using simple calculation steps, which are particularly useful in group footers and summaries.

Entering formulas

The Report Builder is based on the Pentaho Report Builder. A small part of its documentation is at http://wiki.pentaho.com/display/Reporting/9.+Report+Designer+Formula+Expressions.

A further source is the Specifications for the OpenFormula Standard: http://www.oasis-open.org/committees/download.php/16826/openformula-spec-20060221.html with the basic principles for entering formulas summarized in Table 10.

Table 10: Basic principles for entering formulas

Principle	***Detail***
Formulas start with an equals sign.	`=`
References to data fields are placed in square brackets.	`[Field name]`
If the data fields contain special characters (including spaces), the field name must also be enclosed in quotes.	`["This fieldname should be in quotes"]`
Text entry must always be in double quotes.	`"Text entry"`
The following operators are allowed.	+, -, * (Multiplication), / (Division), % (divide the preceding number by 100), ^ (Raise to the power of the following number), & (concatenate text),
The following relationships are possible.	`= , <> , < , <= , > , >=`
Round brackets are allowed.	`( )`
Default error message.	`NA` (Not available)
Error message for an empty field that was defined as a number.	`NaN` (Perhaps "not a number"?)

All formula input applies only to the current record. Relationships with previous or following records are therefore not possible.

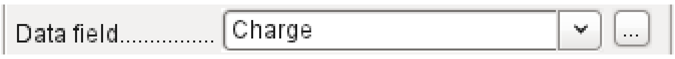
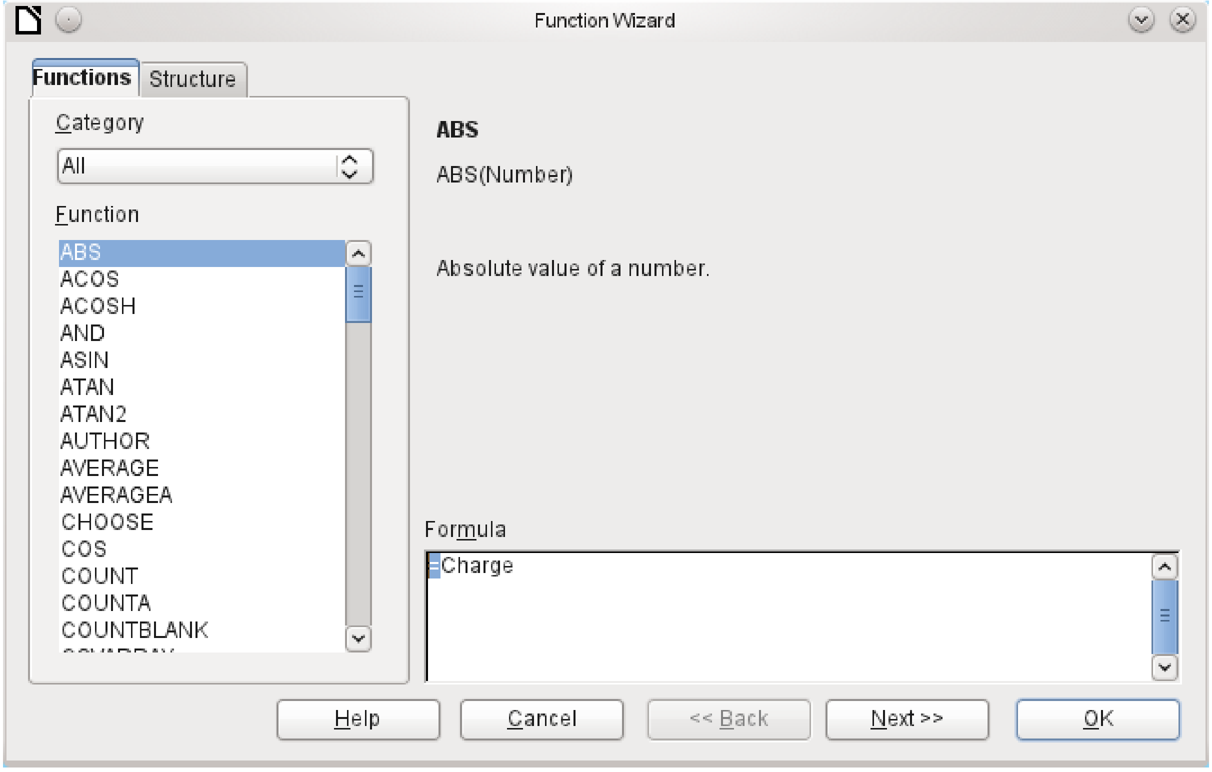

Data field............... Charge

Figure 356: Click the ellipsis button adjacent to Data field to access the Function Wizard

Next to the data field is a button with three dots whenever a formula can be entered as shown in Figure 356. This button starts the Function Wizard shown in Figure 357.



Category: All

ABS

ABS(Number)

Absolute value of a number.

Formula: =Charge

Figure 357: Function Wizard dialog

There are far fewer functions than in Calc although many functions have Calc equivalents. There the Wizard calculates the result of the function directly.

The Function Wizard does not always work perfectly. For instance, text entries are not taken up with double quotes. However, only entries with double quotes are processed when starting the function.

The functions listed in Table 11 are available.

Table 11: Functions available in the Function Wizard

Function	***Description***
Date and Time Functions	
DATE	Produces a valid date from numeric values for the year, the month and the day.
DATEDIF (DAY | MONTH | YEAR)	Returns the total years, months, or days between two date values.
DATEVALUE	Converts an American date entry in text form (quoted) into a date value. The American variant that is produced can then be reformatted.
DAY	Returns the day of the month for a given date. DAY([Date field])
DAYS	Returns the number of days between two dates.
HOUR	Returns the hours of a given time in 24-hour format. HOUR([DateTimeField]) calculates the hours in the field.
MINUTE	Returns the minutes of a date in the internal numeric format MINUTE([Timefield]) calculates the minutes part of the time.
MONTH	Returns the month for an entered date as a number. MONTH([Datefield])

Function	*Description*
NOW	Returns the current date and time.
SECOND	Returns the seconds of a date in the internal numeric format SECOND(NOW()) shows the seconds part of the time the command is executed.
TIME	Shows the current time.
TIMEVALUE	Converts a text entry for a time into a time value for calculations.
TODAY	Shows the current date.
WEEKDAY	Returns the day of the week as a number. Day number 1 is Sunday.
YEAR	Returns the year part of a date entry.
Logical functions	
AND	Yields TRUE when all its arguments are TRUE.
FALSE	Defines the logical value as FALSE.
IF	If a condition is TRUE, then this value, else another value.
IFNA	
NOT	Reverses the logical value of an argument.
OR	Yields TRUE when one of its conditions is TRUE.
TRUE	Defines the logical value as TRUE.
XOR	Yields TRUE when only one of the linked values is TRUE.
Rounding functions	
INT	Rounds down to the previous integer.
Mathematical functions	
ABS	Returns the absolute (non-negative) value of a number.
ACOS	Calculates the arccosine of a number. - arguments between -1 and 1.
ACOSH	Calculates the areacosine (inverse hyperbolic cosine) – argument >= 1.
ASIN	Calculates the arcsine of a number – argument between -1 and 1.
ATAN	Calculates the arctangent of a number.
ATAN2	Calculates the arctangent of an x-coordinate and a y-coordinate.
AVERAGE	Gives the average of the entered values.
AVERAGEA	Gives the average of the entered values. Text is treated as zero.
COS	Argument is the angle in radians whose cosine is to be calculated.
EVEN	Rounds a positive number up or a negative number down to the next even integer.
EXP	Calculates the exponential function (Base 'e').
LN	Calculates the natural logarithm of a number.

Function	***Description***
LibreOfficeG10	Calculates the logarithm of a number (Base '10').
MAX	Returns the maximum of a series of numbers.
MAXA	Returns the maximum value in a row. Any text is set to zero.
MIN	Returns the smallest of a series of values.
MINA	Returns the minimum value in a row. Any text is set to zero.
MOD	Returns the remainder for a division when you enter the dividend and divisor.
ODD	Rounds a positive number up or a negative number down to the next odd integer.
PI	Gives the value of the number 'π'.
POWER	Raises the base to the power of the exponent.
SIN	Calculates the sine of a number.
SQRT	Calculates the square root of a number.
SUM	Sums a list of numeric values.
SUMA	Sums a list of numeric values. Text and Yes/No fields are allowed. Unfortunately this function ends with an error message.
VAR	Calculates the variance, starting from a sample.
Text functions	
EXACT	Shows if two text strings are exactly equal.
FIND	Gives the offset of a text string within another string.
LEFT	The specified number of characters of a text string are reproduced starting from the left.
LEN	Gives the number of characters in a text string.
LOWER	Converts text to lower case.
MESSAGE	Formats the value into the given output format.
MID	The specified number of characters of a text string are reproduced starting from a specified character position.
REPLACE	Replaces a substring by a different substring. The starting position and the length of the substring to be replaced must be given.
REPT	Repeats text a specified number of times.
RIGHT	The specified number of characters of a text string are reproduced starting from the right.
SUBSTITUTE	Replaces specific parts of a given text string by new text. Additionally you can specify which of several occurrences of the target string are to be replaced.
T	Returns the text, or an empty text string if the value is not text (for example a number).
TEXT	Conversion of numbers or times into text.
TRIM	Removes leading spaces and terminal spaces, and reduces multiple spaces to a single space.

Function	***Description***
UNICHAR	Converts a Unicode decimal number into a Unicode character. For example, 196 becomes 'Ä' ('Ä' has the hexadecimal value 00C4, which is 196 in decimals without leading zeros).
UNICODE	Converts a Unicode character into a Unicode decimal number. 'Ä' becomes 196.
UPPER	Returns a text string in upper case.
URLENCODE	Converts a given text into one that conforms to a valid URL. If no particular standard is specified, ISO-8859-1 is followed..
Information functions	
CHOOSE	The first argument is an index, followed by a list of values. The value represented by the index is returned. `CHOOSE(2;"Apple";"Pear";"Banana") returns Pear.` `CHOOSE([age_level_field];"Milk";"Cola";"Beer")` returns a possible drink for the given 'age_level_field' .
COUNT	Only fields containing a number or a date are counted. `COUNT([time];[number])` returns 2, if both fields contain a value (non-NULL) or else 1 or 0.
COUNTA	Includes also fields containing text. Even NULL is counted, along with boolean fields.
COUNTBLANK	Counts the empty fields in a region.
HASCHANGED	Checks if the named column has changed. However no information about the column is provided.
INDEX	Works with regions
ISBLANK	Tests if the field is NULL (empty).
ISERR	Returns TRUE if the entry has an error other than NA. `ISERR(1/0)` gives TRUE
ISERROR	Like ISERR, except that NA also returns TRUE.
ISEVEN	Tests if a number is even.
ISLOGICAL (ISTLOG)	Tests if this is a Yes/No value. `ISLOGICAL(TRUE())` or `ISLOGICAL(FALSE())` yield TRUE, Text values such as `ISLOGICAL("TRUE")` yield FALSE.
ISNA	Tests if the expression is an error of type NA.
ISNONTEXT	Tests if the value is not text.
ISNUMBER	Tests if something is numeric. `ISNUMBER(1)` yields TRUE, `ISNUMBER("1")` yields FALSE
ISODD	Tests if a number is an odd number.
ISREF	Tests if something is a field reference. `ISREF([Fieldname])` yields TRUE, `ISREF(1)` yields FALSE.
ISTEXT	Tests if the content of the field is text.

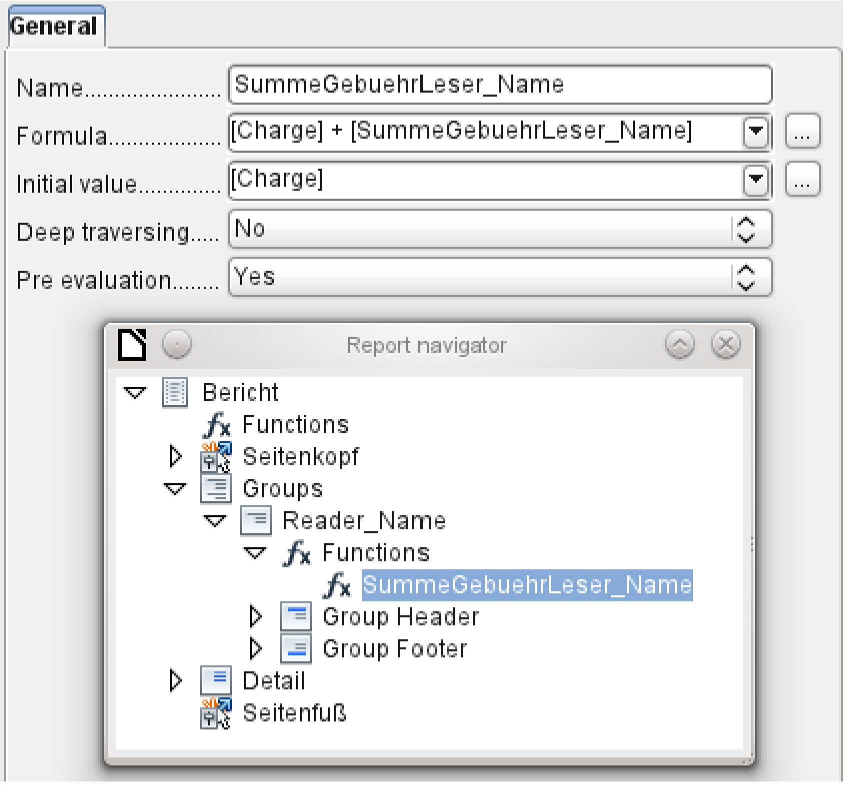
Function	***Description***
NA (NV)	Returns the error code NA.
VALUE	
User defined	
CSVARRAY	Converts CSV text into an array.
CSVTEXT	Converts an array into CSV text.
NORMALIZEARRAY	
NULL	Returns NULL.
PARSEDATE	Converts text into a date. Uses the SimpleDateFormat. Requires a date in text as described in this date format. Example: `PARSEDATE("9.10.2012";"dd.MM.yyyy")` yields the internally usable number for the date.
Document information	
AUTHOR	Author, as read from the **Tools > Options > LibreOffice > User data**. This is not therefore the actual author but the current user of the database.
TITLE	Returns the title of the report.

User-defined functions

You can use user-defined functions to return specific intermediate results for a group of records. In our example, a function of this sort was used to calculate the fines in the Reader_Name_Footer area.

In the Report Navigator the function is displayed under Reader_Name group. By right-clicking on this function, you can define additional functions by name (see Figure 358).

The properties of the function SummeGebuehrLeser_Name are shown above. The formula adds the field Charge to the value already stored in the function itself. The initial value is the value of the Charge field on the first traverse of the group. This value is stored in the function under the function name and is reused in the formula, until the loop is ended and the group footer is written.



General

Name: SummeGebuehrLeser_Name

Formula: [Charge] + [SummeGebuehrLeser_Name]

Initial value: [Charge]

Deep traversing: No

Pre evaluation: Yes

Report navigator

- Bericht
  - Functions
  - Seitenkopf
  - Groups
    - Reader_Name
      - Functions
        - SummeGebuehrLeser_Name
      - Group Header
      - Group Footer
  - Detail
  - Seitenfuß

Figure 358: Select a user-defined function in the Report navigator to access its properties

Deep traversing seems to have no function for now, unless charts are being treated here as subreports.

If Pre evaluation is activated for the function, the result can also be placed in the group header. Otherwise the group header contains only the corresponding value of the first field of the group.

User-defined functions can also reference other user-defined functions. In that case you must ensure that the functions used have already been created. Pre-calculation in functions that refer to other functions must be excluded.

```
[SumMarksClass] / ([ClassNumber]+1)
```

Refers to the Class group. The content of the Marks field is summed and the sum for all the records is returned. The sum of the marks is divided by the sum of the records. To get the correct number, 1 must be added as shown with [ClassNumber]. This will then yield the average marks.

Formula entry for a field

Using **Data > Data field** you can enter formulas that affect only one field in the Details area.

```
IF([boolean_field];"Yes";"No")
```

sets the allowable values to "Yes" or "No" instead of TRUE and FALSE.

It can happen that in a field with a formula input, just a single number appears. In text fields this is a zero. To fix this, you must change the text field from the default “Number” to “Text”.

Conditional print

Conditional Print Expression:

Figure 359: Conditional Print Expression field on the General tab of Properties

The general properties of group headers, group footers, and fields include a Conditional Print Expression field (see Figure 359). Formulas that are written in this field influence the content of a field or the display of an entire region. Here, too, you can make use of the Function Wizard.

```
[Fieldname]="true"
```

causes the content of the named field to be displayed only if it is true.

Note

See also the database Example_Report_conditional_Overlay_Graphics.odb, which is included in the example databases for o this book.

Many forms of conditional display are not fully determined by the specified properties. For instance, if a graphical separator line is to be inserted after the tenth place of a list of competition results, you cannot simply use the following conditional display command for the graphic:

```
[Place]=10
```

This command does not work. Instead the graphic will continue to appear in the Details section after each subsequent record. See Bug 73707.

If you just want a rectangular shape overlaid at this location, that can be done using a graphical control, which can be given the address of a (monochrome) graphical file. In the general properties of this control, **Scaling > Autom.** should be selected. Then the graphic will fit the form and the condition will be fulfilled.

It is safer to bind the conditional display to a group footer rather than to the graphic, if this is not otherwise needed. The line is positioned in the group footer. Then the line does actually appear after the 10th place, when formulated as above. Because the condition is associated with the group footer, the line appears only if it is really needed. Otherwise the conditional display can lead to a blank appearing instead of the line.

Here you can also use the shapes provided by **Insert > Shapes > Standard shapes**, for example the horizontal line to blend with the group footer.

Conditional formatting

Conditional formatting can be used, for example in Figure 360, to format a calendar so that weekends are shown differently. Choose **Format > Conditional formatting** and enter:

```
WEEKDAY([Date])=1
```

and the corresponding formatting for Sundays.

If you use ‘Expression is’ in conditional formatting, you can enter a formula. As is usual in Calc, several conditions can be formulated and are evaluated sequentially. In the above example, first Sunday is tested for, then Saturday. Finally there might be a query about the content of the field. So for example the content ‘Holiday’ would lead to a different format.

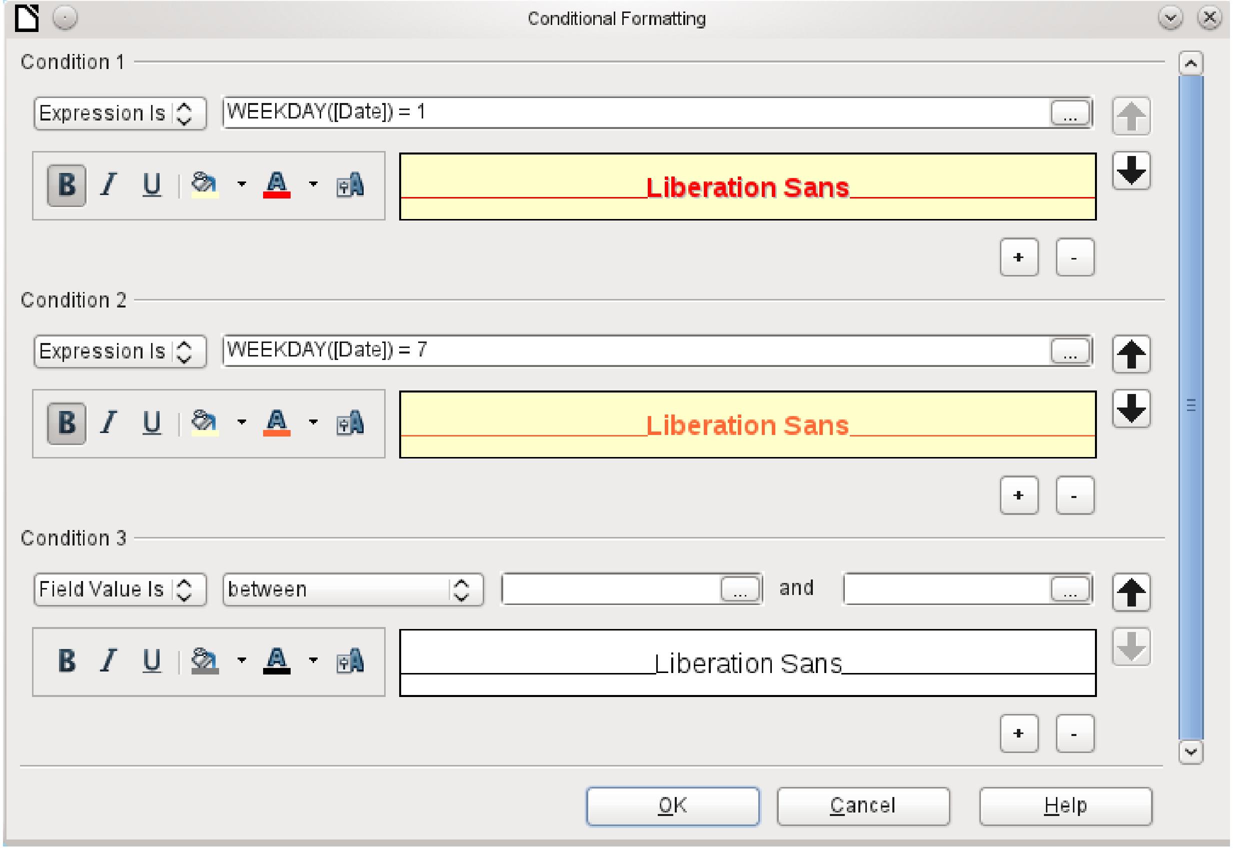


Figure 360: Conditional Formatting dialog

Note

If additional uncorrectable errors occur (formulas not implemented, too long text shown as an empty field, and so on), it is sometimes necessary to delete parts of the report or simply create it afresh.

Examples of reports created with the Report Builder

The Report Builder can be tricky to use, as some functions appear to be available but do not work as intended. In addition, the Help system does not currently address these problem areas. For that reason a few examples are provided here, showing how the Report Builder can be used for various types of report.

Printing bills

To create a bill requires the following considerations:

- The individual items must be numbered.
- Bills that require more than one page must be given page numbers.
- Bills that require more than one page should have a running total on each page, which is carried over to the next page.

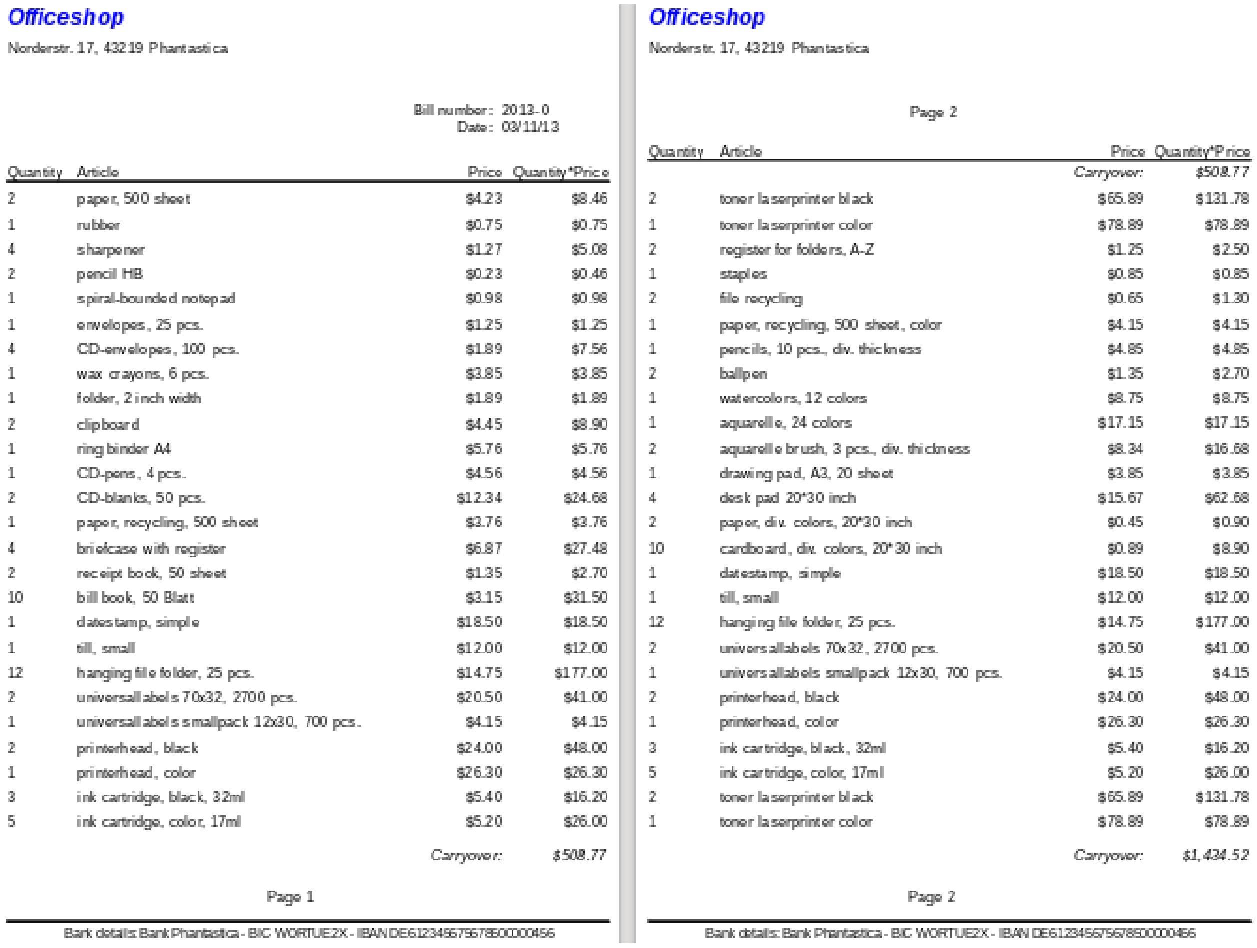
Several current bugs seem to make this impossible:

- Bug 51452: If a group is set to “Repeat Section”, a page break is inserted automatically before and after the group.
- Bug 51453: Groups with individual pagination are allowed for but do not actually work.

- Bug 51959: A group footer cannot be repeated. It can only occur at the end of the group, not at the end of each page. And if you choose “Repeat Section”, it disappears completely.

In addition, there are problems with inserting lines into reports. The built-in horizontal and vertical lines only appear in LibreOffice versions 4.0.5 and 4.1.1 respectively. You can use rectangles as a substitute, but they cannot be correctly positioned when there is a page break in the section.

The report in a simple form should look like Figure 361:



Officeshop

Norderstr. 17, 43219 Phantastica

Bill number: 2013-0
Date: 03/11/13

Quantity	Article	Price	Quantity*Price
2	paper, 500 sheet	$4.23	$8.46
1	rubber	$0.75	$0.75
4	sharpener	$1.27	$5.08
2	pencil HB	$0.23	$0.46
1	spiral-bounded notepad	$0.98	$0.98
1	envelopes, 25 pcs.	$1.25	$1.25
4	CD-envelopes, 100 pcs.	$1.89	$7.56
1	wax crayons, 6 pcs.	$3.85	$3.85
1	folder, 2 inch width	$1.89	$1.89
2	clipboard	$4.45	$8.90
1	ring binder A4	$5.76	$5.76
1	CD-pens, 4 pcs.	$4.56	$4.56
2	CD-blanks, 50 pcs.	$12.34	$24.68
1	paper, recycling, 500 sheet	$3.76	$3.76
4	briefcase with register	$6.87	$27.48
2	receipt book, 50 sheet	$1.35	$2.70
10	bill book, 50 Blatt	$3.15	$31.50
1	datestamp, simple	$18.50	$18.50
1	till, small	$12.00	$12.00
12	hanging file folder, 25 pcs.	$14.75	$177.00
2	universallabels 70x32, 2700 pcs.	$20.50	$41.00
1	universallabels smallpack 12x30, 700 pcs.	$4.15	$4.15
2	printerhead, black	$24.00	$48.00
1	printerhead, color	$26.30	$26.30
3	ink cartridge, black, 32ml	$5.40	$16.20
5	ink cartridge, color, 17ml	$5.20	$26.00
		Carryover:	*$508.77*

Page 1

Bark details:Bank Phantastica - BIC WORTUE2X - IBAN DE61234567567850000456

Officeshop

Norderstr. 17, 43219 Phantastica

Page 2

Quantity	Article	Price	Quantity*Price
		Carryover:	*$508.77*
2	toner laserprinter black	$65.89	$131.78
1	toner laserprinter color	$78.89	$78.89
2	register for folders, A-Z	$1.25	$2.50
1	staples	$0.85	$0.85
2	file recycling	$0.65	$1.30
1	paper, recycling, 500 sheet, color	$4.15	$4.15
1	pencils, 10 pcs., div. thickness	$4.85	$4.85
2	ballpen	$1.35	$2.70
1	watercolors, 12 colors	$8.75	$8.75
1	aquarelle, 24 colors	$17.15	$17.15
2	aquarelle brush, 3 pcs., div. thickness	$8.34	$16.68
1	drawing pad, A3, 20 sheet	$3.85	$3.85
4	desk pad 20*30 inch	$15.67	$62.68
2	paper, div. colors, 20*30 inch	$0.45	$0.90
10	cardboard, div. colors, 20*30 inch	$0.89	$8.90
1	datestamp, simple	$18.50	$18.50
1	till, small	$12.00	$12.00
12	hanging file folder, 25 pcs.	$14.75	$177.00
2	universallabels 70x32, 2700 pcs.	$20.50	$41.00
1	universallabels smallpack 12x30, 700 pcs.	$4.15	$4.15
2	printerhead, black	$24.00	$48.00
1	printerhead, color	$26.30	$26.30
3	ink cartridge, black, 32ml	$5.40	$16.20
5	ink cartridge, color, 17ml	$5.20	$26.00
2	toner laserprinter black	$65.89	$131.78
1	toner laserprinter color	$78.89	$78.89
		Carryover:	*$1,434.52*

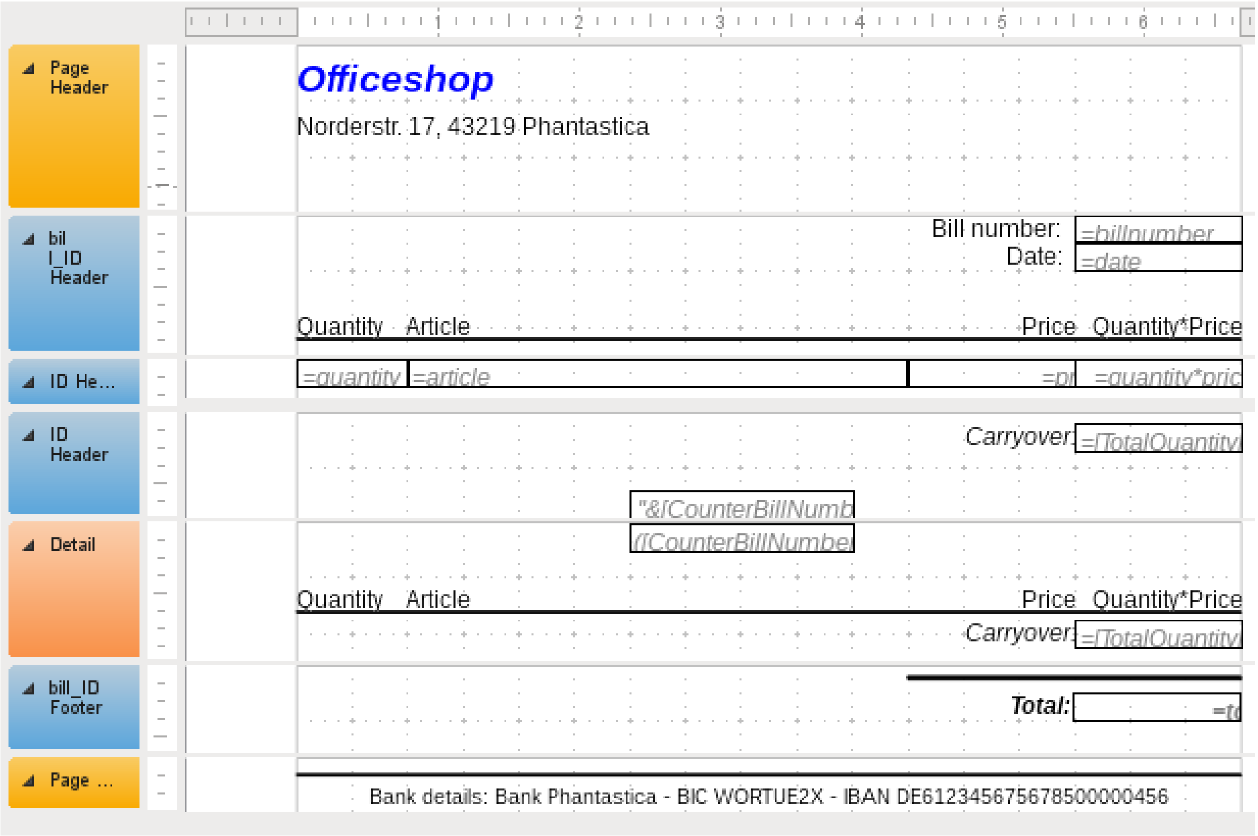
Page 2

Bank details: Bank Phantastica - BIC WORTUE2X - IBAN DE61234567567850000456

Figure 361: Initial draft of the Officeshop report

To overcome the restrictions described above, the creation of the corresponding bill requires exact attention to the page measurements in the final printed document. This example in Figure 362 starts out from a DIN-A4 format. The total height of each page is therefore 29.7 cm.

The report needs to be divided into several groups. Two groups relate to the same table field and contain the same table element.



Officeshop

Norderstr. 17, 43219 Phantastica

Bill number: =billnumber

Date: =date

Quantity Article Price Quantity*Price

=quantity =article =pr =quantity*pric

Carryover: =[TotalQuantity

"&[CounterBillNumb

([CounterBillNumbe

Quantity Article Price Quantity*Price

Carryover: =[TotalQuantity

Total: =to

Bank details: Bank Phantastica - BIC WORTUE2X - IBAN DE61234567567850000456

Figure 362: Editing the Officeshop report

Table 12 shows the division of the page into the various sections of the report.

Table 12: Division of page into sections for the Officeshop report

Position	Description	Value
A	Top margin	2.00 cm
B	Page header (appears on each page, contains no database content, only material such as a company logo or the supplier’s address).	3.00 cm
C	Group header for the bill (only items that belong to a bill number should be added later. The group header appears only at the beginning of the bill).	2.50 cm
D	Group header for the individual items. (The Detail section is required for a different purpose. Therefore a group comes here that also sorts the contents after input. This group contains only one value.)	0.70 cm
E	Group header, also bound to the items. (This section is only displayed if so many items occur that a further page is necessary. It contains the running total and the page number at the bottom of the page. There is a page break after this section,)	2.00 cm
F	Details section. (This section is only shown if so many items occur that a further page is needed. It contains the sum carried over, and the page number at the top of the page.)	2.50 cm
G	Group footer for the bill number. (Here follows the total sum for the bill, possibly with added VAT. The group footer appears only on the last page of the bill.)	1.60 cm

Position	*Description*	*Value*
H	Page footer (e.g. bank details)	1.00 cm
I	Page margin	1.00 cm

A page break follows only if there are too many items. For the items, we have the following free spaces as shown in Table 13:

Table 13: Free space for items in the Officeshop report

	29.70 cm	(DIN A 4)
-	2.00 cm	(Pos. A)
-	3.00 cm	(Pos. B)
-	2.50 cm	(Pos. C)
-	1.60 cm	(Pos. G)
-	1.00 cm	(Pos. H)
-	1.00 cm	(Pos. I)
=	18.60 cm	

The remaining free space is therefore at most 18.60 cm/0.70 cm = 26.57. Rounding up gives 26 item lines.

As soon as the 27th item appears, a page break must immediately follow. This indicates that the group header E and the Details section must be displayed. So we need a counter for the items (section C). When this counter hits 27, the Details section (F) is displayed.

The counter for the items is defined as follows:

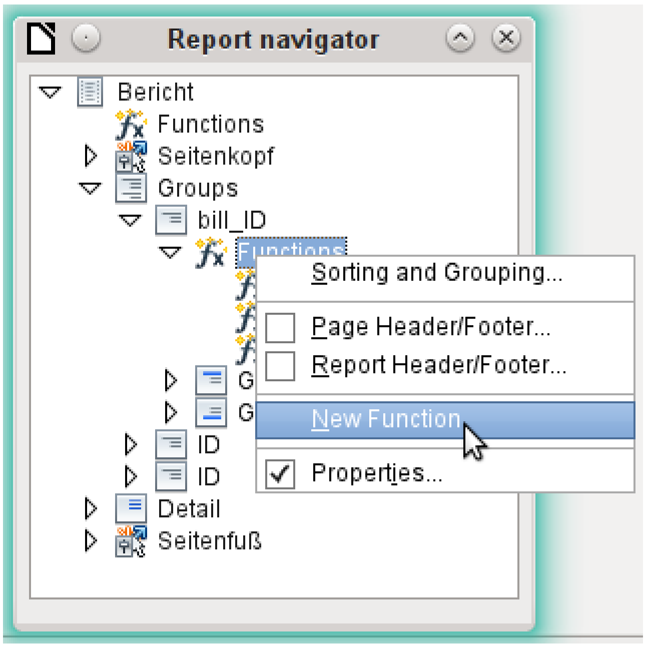


Figure 363: Creating a new function using the Report navigator

Using the Report Navigator, we look for the Bill_ID group (see Figure 363). Our new function is called CounterBillNumber. The formula is [CounterBillNumber] + 1. The initial value is 1. No subreports are bound in (there is no such function). Nor is it calculated in advance. For advance calculations we use a separate counter, CounterTotal.

The group heading E and the Details section F are both displayed if a total of more than 26 items are in the bill and the current record position has reached 26. The expression for the conditional display is therefore the same for both:

```
AND([CounterBillNumber]=26;[CounterTotal]>26)
```

The content of this section therefore appears only if at least 27 items are expected. Group header E appears on the first page. A page break follows, and the content of the Details section appears on the next page.

Now we must calculate the number of sections that must appear on the first page as shown in Table 14.

Table 14: Calculating the number of sections on the first page

	29.70 cm	(DIN A 4)
-	2.00 cm	(Pos. A)
-	3.00 cm	(Pos. B)
-	2.50 cm	(Pos. C)
-	18.20 cm	(Pos. D * 26)
-	1.00 cm	(Pos. H)
-	1.00 cm	(Pos. I)
=	2.00 cm	

The group footer is not required for the first page. There are 26 item lines in total. The group header E can therefore occupy at most 2 cm on the first page. These 2 cm must accommodate the running total and the page number. To ensure a correct page break in all circumstances, this section should actually be a bit smaller than 2 cm. In the example, we use 1.90 cm.

The Details section comes at the top of the next page. As the group header for the items (C) does not occur on this page, the Details section can take up as much space as the header did, namely 2.50 cm. Then begin the next lot of items with the same arrangement as on the previous page.

The sum carried over is calculated by simply adding up the previous items.

The Report Navigator is used to find the Bill_number group. The new function to be created will be called TotalPrice. The formula is [Price] + [TotalPrice]. The initial value is [Price]. No subreports are bound in. Nor does this figure need to be calculated in advance.

The sum to be carried over is displayed both in the Group header E and in the Details section. In the group header, it is right at the top of the page. On the first page it appears at the bottom. In the Details section, the sum is right at the bottom. It appears on page 2 directly below the table header.

The query to determine the page number is similar to the one for determining the display of the group header and the details section:

```
IF([CounterBillNumber]=26;"Page 1";"")
```

This gives the page number for the first page. Further IF-queries can be used for the other pages.

The page number for the following page is easily set to “Page 2” in the same way.

If the formulas are continued in the same way, they can cover as many pages as you like.

The expression for conditional display changes from

```
AND([CounterBillNumber]=26;[CounterBillComplete]>26)
```

to

```
AND(MOD([CounterBillNumber];26)=0;
[CounterBillComplete]>[CounterBillNumber])
```

The group header E and the Details section F appear only when dividing the item counter by 26 gives no remainder and the total number of items is greater than the item counter.

The expression for the page number changes from

```
IF([CounterBillNumber]=26;"Page 1";"")
```

to

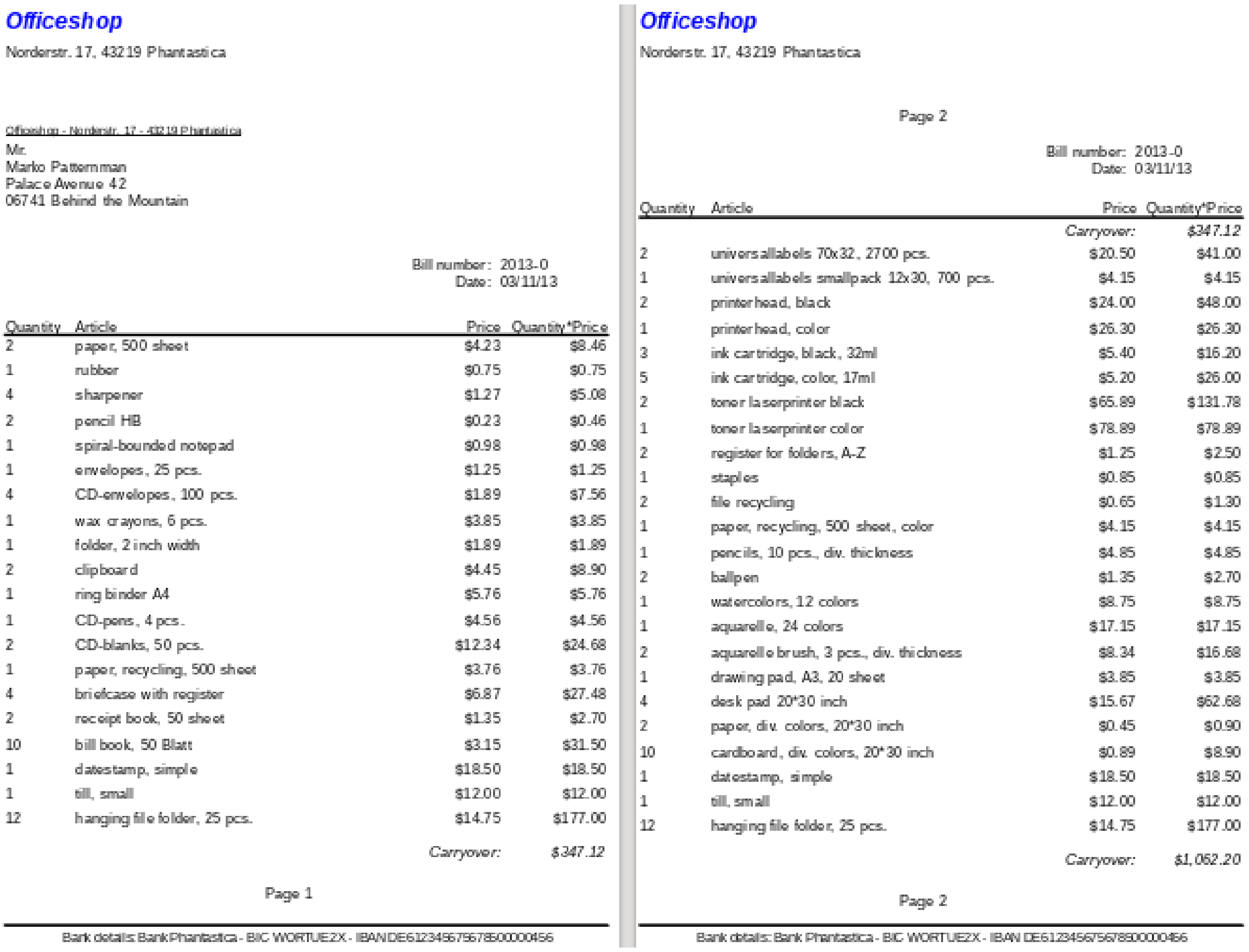
```
"Pag "&[CounterBillNumber]/26
```

for the current page and

```
"Page "&([CounterBillNumber]/26)+1
```

for the following page.

The report shown in Figure 364 using these settings is still not quite ready for use:



Officeshop

Norderstr. 17, 43219 Phantastica

Officeshop - Norderstr. 17 - 43219 Phantastica
Mr.
Marko Pattemman
Palace Avenue 42
06741 Behind the Mountain

Bill number: 2013-0
Date: 03/11/13

Quantity	Article	Price	Quantity*Price
2	paper, 500 sheet	$4.23	$8.46
1	rubber	$0.75	$0.75
4	sharpener	$1.27	$5.08
2	pencil HB	$0.23	$0.46
1	spiral-bounded notepad	$0.98	$0.98
1	envelopes, 25 pcs.	$1.25	$1.25
4	CD-envelopes, 100 pcs.	$1.89	$7.56
1	wax crayons, 6 pcs.	$3.85	$3.85
1	folder, 2 inch width	$1.89	$1.89
2	clipboard	$4.45	$8.90
1	ring binder A4	$5.76	$5.76
1	CD-pens, 4 pcs.	$4.56	$4.56
2	CD-blanks, 50 pcs.	$12.34	$24.68
1	paper, recycling, 500 sheet	$3.76	$3.76
4	briefcase with register	$6.87	$27.48
2	receipt book, 50 sheet	$1.35	$2.70
10	bill book, 50 Blatt	$3.15	$31.50
1	datestamp, simple	$18.50	$18.50
1	till, small	$12.00	$12.00
12	hanging file folder, 25 pcs.	$14.75	$177.00
		Carryover:	*$347.12*

Page 1

Bank details: Bank Phantastica - BIC WORTUE2X - IBAN DE6123456756788500000456

Officeshop

Norderstr. 17, 43219 Phantastica

Page 2

Bill number: 2013-0
Date: 03/11/13

Quantity	Article	Price	Quantity*Price
		Carryover:	*$347.12*
2	universal labels 70x32, 2700 pcs.	$20.50	$41.00
1	universal labels smallpack 12x30, 700 pcs.	$4.15	$4.15
2	printerhead, black	$24.00	$48.00
1	printerhead, color	$26.30	$26.30
3	ink cartridge, black, 32ml	$5.40	$16.20
5	ink cartridge, color, 17ml	$5.20	$26.00
2	toner laserprinter black	$65.89	$131.78
1	toner laserprinter color	$78.89	$78.89
2	register for folders, A-Z	$1.25	$2.50
1	staples	$0.85	$0.85
2	file recycling	$0.65	$1.30
1	paper, recycling, 500 sheet, color	$4.15	$4.15
1	pencils, 10 pcs., div. thickness	$4.85	$4.85
2	ballpen	$1.35	$2.70
1	watercolors, 12 colors	$8.75	$8.75
1	aquarelle, 24 colors	$17.15	$17.15
2	aquarelle brush, 3 pcs., div. thickness	$8.34	$16.68
1	drawing pad, A3, 20 sheet	$3.85	$3.85
4	desk pad 20*30 inch	$15.67	$62.68
2	paper, div. colors, 20*30 inch	$0.45	$0.90
10	cardboard, div. colors, 20*30 inch	$0.89	$8.90
1	datestamp, simple	$18.50	$18.50
1	till, small	$12.00	$12.00
12	hanging file folder, 25 pcs.	$14.75	$177.00
		Carryover:	*$1,062.20*

Page 2

Bank details: Bank Phantastica - BIC WORTUE2X - IBAN DE6123456756788500000456

Figure 364: Revised draft of the Officeshop report

Because of the address field, there are fewer items on the first page of the bill than on the second page. The Details section right at the top of the second page is therefore significantly smaller than the group header for the bill (C).

To allow for different numbers of items on the first two pages, the formulas must be adjusted.

The following calculation ensures that the corresponding sections are correctly displayed.

```
AND(MOD([CounterBillNumber]-20;24)=0;
[CounterBillComplete]>[CounterBillNumber])
```

We subtract the number of items on the first page from the item counter. This difference is divided by the possible total number of items on the second page. If the division is exact with no remainder, the first condition for the display of the group header E and the Details section F has been fulfilled. In addition, as previously shown, the current value of the item counter must be less than than the expected total. Otherwise there would be room for the calculated total sum on the current page.

The possible total number of items on the second page is now smaller because this page now contains the bill number and the date.

The page number can now be calculated more simply:

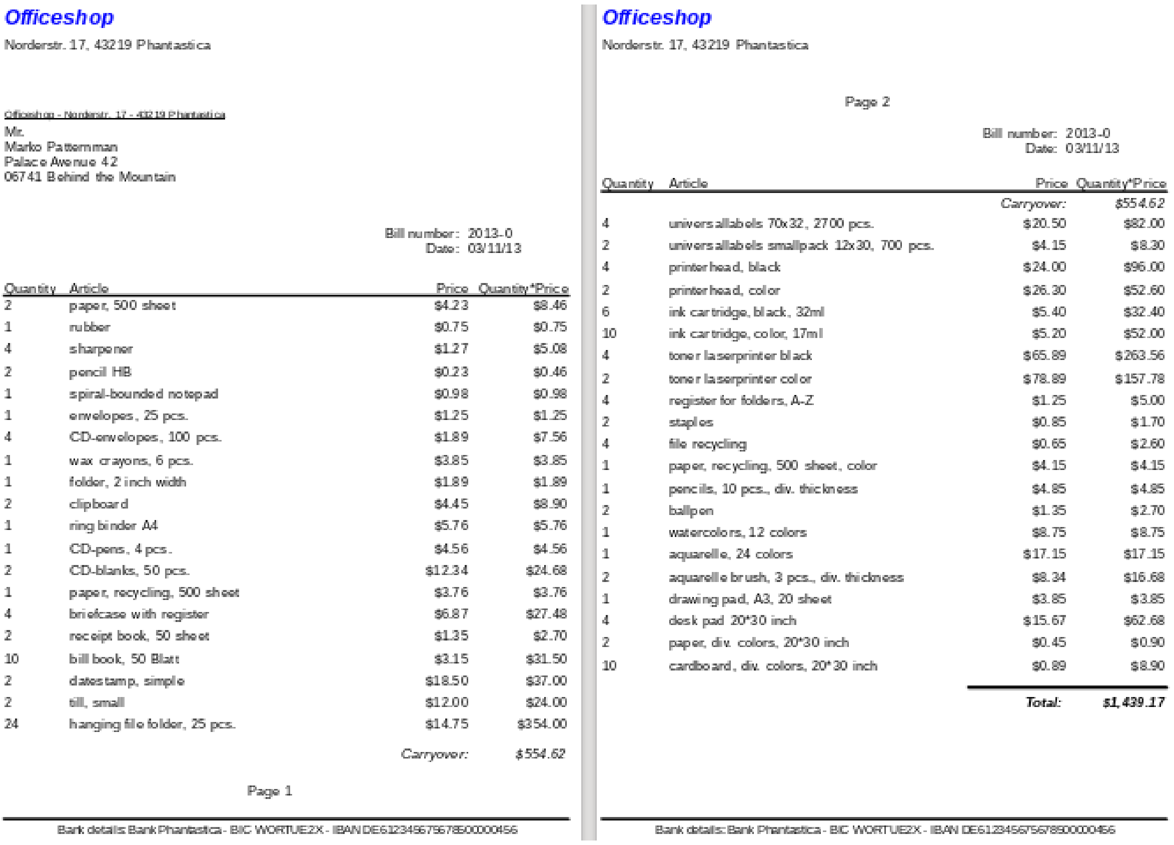
```
"Page "&INT([CounterBillNumber]/24)+1
```

The INT function rounds down to the nearest integer. The first page contains a maximum of 20 items. The division therefore gives a result of <1 for this first page. This rounds down to zero. So we need to add 1 to the calculated page number so that 1 appears on the first page. Similarly for page 2.

The report still shows an aesthetic fault. A careful look at the bill items shows that the three bottom items on the pages are the same. This is because records have simply been copied. They are not actually the same records, but different ones for the same product which have been processed independently of one another. It would be better here to group by product type so that each product appears only once with the total number ordered.

Basically you should try to remove as many calculations and groupings as possible from the Report Builder. Instead of using the Report Builder’s groups, it is better to use the grouping functions in the query editor. In order for the Report Builder to process the query easily, turn it into a view. Otherwise the Report Builder will try to improve the query with its own grouping and sorting functions, which can rapidly lead to quite impractical coding.

The final result is shown in Figure 365:



Officeshop

Norderstr. 17, 43219 Phantastica

Officeshop - Norderstr. 17 - 43219 Phantastica

Mr.
Marko Pattemman
Palace Avenue 42
06741 Behind the Mountain

Bill number: 2013-0
Date: 03/11/13

Quantity	Article	Price	Quantity*Price
2	paper, 500 sheet	$4.23	$8.46
1	rubber	$0.75	$0.75
4	sharpener	$1.27	$5.08
2	pencil HB	$0.23	$0.46
1	spiral-bounded notepad	$0.98	$0.98
1	envelopes, 25 pcs.	$1.25	$1.25
4	CD-envelopes, 100 pcs.	$1.89	$7.56
1	wax crayons, 6 pcs.	$3.85	$3.85
1	folder, 2 inch width	$1.89	$1.89
2	clipboard	$4.45	$8.90
1	ring binder A4	$5.76	$5.76
1	CD-pens, 4 pcs.	$4.56	$4.56
2	CD-blanks, 50 pcs.	$12.34	$24.68
1	paper, recycling, 500 sheet	$3.76	$3.76
4	briefcase with register	$6.87	$27.48
2	receipt book, 50 sheet	$1.35	$2.70
10	bill book, 50 Blatt	$3.15	$31.50
2	datestamp, simple	$18.50	$37.00
2	till, small	$12.00	$24.00
24	hanging file folder, 25 pcs.	$14.75	$354.00
		Carryover:	*$554.62*

Page 1

Bank details: Bank Phantastica - BIC WORTUE2X - IBAN DE612345675678500000456

Officeshop

Norderstr. 17, 43219 Phantastica

Page 2

Bill number: 2013-0
Date: 03/11/13

Quantity	Article	Price	Quantity*Price
		Carryover:	*$554.62*
4	universallabels 70x32, 2700 pcs.	$20.50	$82.00
2	universallabels smallpack 12x30, 700 pcs.	$4.15	$8.30
4	printerhead, black	$24.00	$96.00
2	printerhead, color	$26.30	$52.60
6	ink cartridge, black, 32ml	$5.40	$32.40
10	ink cartridge, color, 17ml	$5.20	$52.00
4	toner laserprinter black	$65.89	$263.56
2	toner laserprinter color	$78.89	$157.78
4	register for folders, A-Z	$1.25	$5.00
2	staples	$0.85	$1.70
4	file recycling	$0.65	$2.60
1	paper, recycling, 500 sheet, color	$4.15	$4.15
1	pencils, 10 pcs., div. thickness	$4.85	$4.85
2	ballpen	$1.35	$2.70
1	watercolors, 12 colors	$8.75	$8.75
1	aquarelle, 24 colors	$17.15	$17.15
2	aquarelle brush, 3 pcs., div. thickness	$8.34	$16.68
1	drawing pad, A3, 20 sheet	$3.85	$3.85
4	desk pad 20*30 inch	$15.67	$62.68
2	paper, div. colors, 20*30 inch	$0.45	$0.90
10	cardboard, div. colors, 20*30 inch	$0.89	$8.90
		Total:	***$1,439.17***

Bank details: Bank Phantastica - BIC WORTUE2X - IBAN DE612345675678500000456

Figure 365: Final draft of the Officeshop report

Here all the items occur only once. The original 12 hanging folders at the bottom of page 1 are now 24 hanging folders. Quantity*Price has been recalculated accordingly.

Printing reports for the current record in a form

Especially in the kind of bill production shown in the previous example, it can be useful to prepare and preview a new print after each entry of an item. The content of the bill should be determined usIng a form, and the printing of the individual documents should follow.

Reports cannot be launched with a filter by using macros. However the query which is used to make the report can be filtered beforehand. This is done either by a parameterized query using the form

```
SELECT * FROM "bill" WHERE "ID" = :ID
```

or by a query that is supplied with data using a one-line filter table:

```
SELECT * FROM "bill" WHERE "ID" = (SELECT "Integer" FROM "Filter" WHERE
"ID" = TRUE)
```

In a query with parameters, the content must be inserted into the corresponding dialog field after launching it.

When a filter table is used to control the process, its content to be written by a macro. Therefore a separate entry is no longer necessary, which makes life easier for the user. The process is described below.

Building the filter table

The filter table should contain only one record at a time. This means that its primary key can be a Yes/No field. Other fields in the table are named in such a way that it is clear what type of content they hold. In the example as shown in Table 15, the field that is to filter the primary key of the Bill table is called Integer, as the key itself is of this type. For other filtering possibilities, other fields can be added later. The Integer filter can be used at different times for several different tables, since the old content is always overwritten with new before printing. But this multiple use only works in a single-user database (Base with internal HSQLDB). In a multi-user database, there is always the possibility that some other user will change the filter value in one of the normal tables while the query that uses the filter is being carried out,

Table 15: Filter table provided to filter the primary key of the Bill table

Fieldname	***Fieldtyp***
ID	Yes/No [BOOLEAN]
Integer	Integer [INTEGER]

This table is filled at the start with one record. For this purpose, the ID field must be set to have the value Yes (“TRUE” in SQL).

Creating the macro to launch the filtered report

To call up a single report, the form must contain somewhere the primary key of the Bill table. This primary key is read and transferred to the filter table by a macro. Then the macro launches the desired report.

```
SUB Filter_and_Print
   DIM oDoc AS OBJECT
   DIM oDrawpage AS OBJECT
   DIM oForm AS OBJECT
   DIM oField AS OBJECT
   DIM oDatasorce AS OBJECT
   DIM oConnection AS OBJECT
   DIM oSQL_Command AS OBJECT
   DIM stSQL AS STRING
   oDoc = thisComponent
   oDrawpage = oDoc.Drawpage
   oForm = oDrawpage.Forms.getByName("MainForm")
   oField = oForm.getByName("fmtID")
   oDatasource = ThisComponent.Parent.CurrentController
   If NOT (oDatasorce.isConnected()) THEN
      oDatasource.connect()
   END IF
   oConnection = oDatasource.ActiveConnection()
   oSQL_Command = oConnection.createStatement()
   stSql = "UPDATE ""Filter"" SET ""Integer"" = '"+oField.GetCurrentValue()+"' WHERE
      ""ID"" = TRUE"
   oSQL_Command.executeUpdate(stSql)
   ThisDatabaseDocument.ReportDocuments.getByName("bill").open
END SUB
```

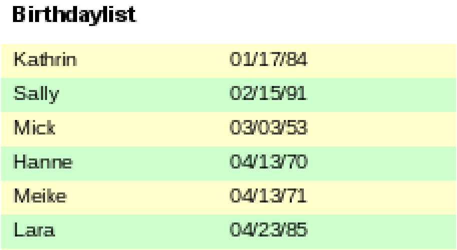
In this example, the form is called MainForm. The primary key is called fmtID. This key field does not have to be visible in order for the macro to access it. The value of the field is read out and the `UPDATE` command writes it into the table. Then the report is launched. The view to which the report refers is expanded into a condition:

```
… WHERE "bill_ID" = IFNULL((SELECT "Integer" FROM "Filter" WHERE "ID" =
TRUE),"bill_ID") …
```

The Integer field is read out. If it has no value, it is set to bill_ID. This means that all records are displayed, not just the filter record. So from the same view, all the stored records can be printed.

Alternate background coloring of lines

When you are reading individual lines in a table within a report, it is easy for to the eye to slip up or down a line. Coloring in the background of at least one line helps to prevent this. In the example shown in Figure 366, the lines are simply colored alternately. The result looks like this:



Birthdaylist

Kathrin	01/17/84
Sally	02/15/91
Mick	03/03/53
Hanne	04/13/70
Meike	04/13/71
Lara	04/23/85

Figure 366: Example showing alternate lines colored to increase usability

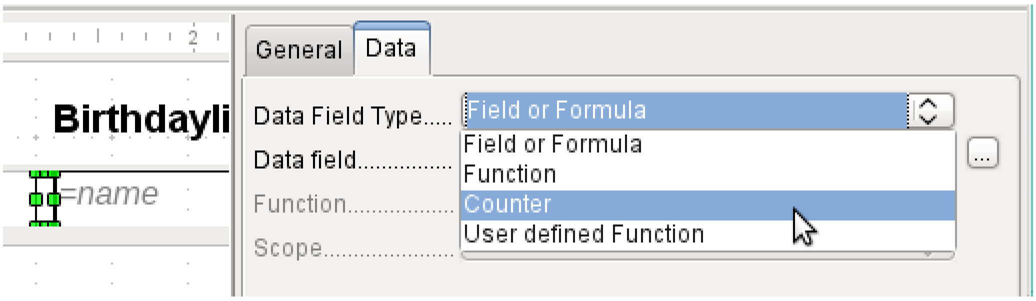
The basis of the report is a query with names and dates. The original table is queried with sorting by date (month and day) to show the sequence of birthdays through the year. This is done by:

```
... ORDER BY MONTH("birthday") ASC, DAY("birthday") ASC
```

To get the alternating colors, we need to create a function which can later use some value to set conditions, that in turn determine the background color. We create a text field in the report and, using **Properties > Data > Data Field Type**, we define a counter as shown in Figure 367.

Note

The database Example_Report_Rows_Color_change_Columns.odb is included in the example databases for this book.


Birthdayli

=name

General | Data

Data Field Type..... Field or Formula

- Field or Formula
- Function
- Counter
- User defined Function

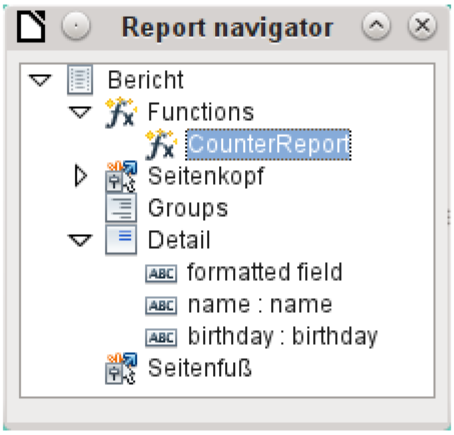
Data field...............

Function.................

Scope....................

Figure 367: Setting the Data Field Type property of a text field to Counter

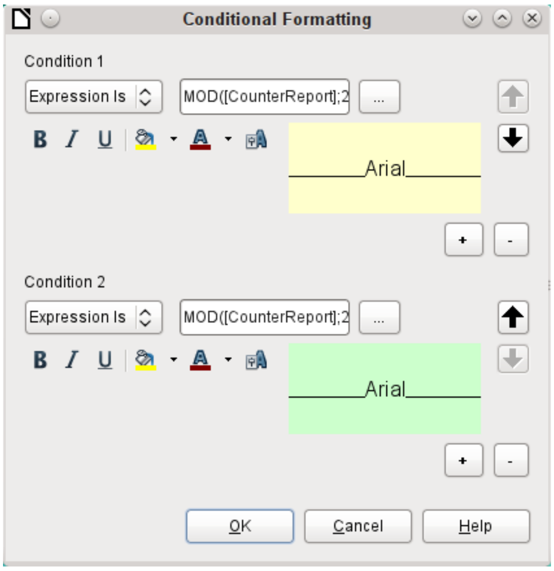
We need the name of the function for the conditional formatting. The actual counter does not need to appear in the expression. The name can be read directly from the field. If the field is deleted again, the function name is accessible using **View > Report Navigator** (see Figure 368).


Report navigator

```
Bericht
  Functions
    CounterReport
  Seitenkopf
  Groups
  Detail
    formatted field
    name : name
    birthday : birthday
  Seitenfuß
```

Figure 368: Function name highlighted in the Report navigator

Now the counter must be used to give each text field its own format as shown in Figure 369. The condition is an expression that is not directly connected to the field. Therefore it is set up as `Condition 1 > Expression is > MOD([CounterReport];2)>0`. `MOD` calculates the remainder from a division. For all odd numbers, this is greater than 0; for even numbers it is 0 precisely. Lines 1, 3 and 5 will therefore be assigned this format.


Conditional Formatting

Condition 1

Expression Is | MOD([CounterReport];2

Arial

Condition 2

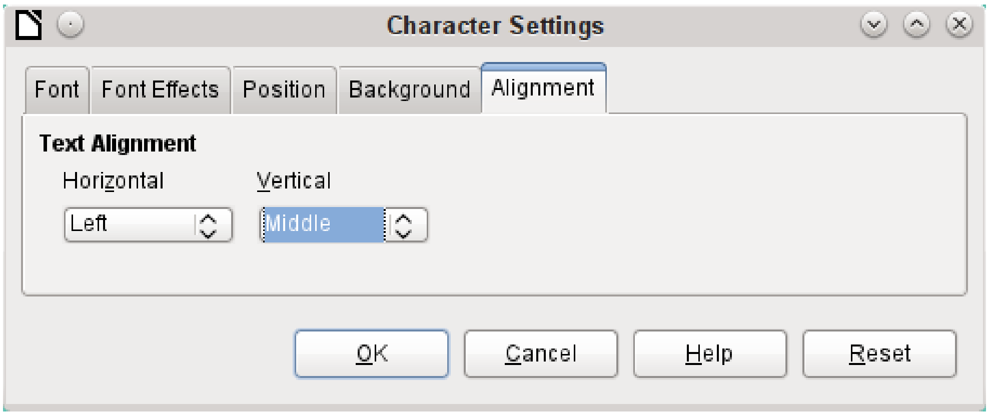
Expression Is | MOD([CounterReport];2

Arial

Figure 369: Conditional Formatting to alternate colors between adjacent lines

The second condition is formulated as the exact opposite and a corresponding format assigned. Actually this second condition could be omitted and a default format set in the properties of each field. The conditional format specified in the first condition would then replace this default whenever the condition was satisfied.

As the conditional format overwrites all default formatting, a text alignment for this format must be included using the character settings shown in Figure 370.



Text Alignment

Horizontal: Left

Vertical: Middle

Figure 370: Using the Character Settings dialog for text alignment

Here the letters are to be vertically centered on the colored text field. The horizontal alignment is the standard one, but an indent needs to be added, so that the letters are not crowded together at the left side of the text field.

Experiments with adding spaces to the content of a query or formula have not led to proper indentation of the text. The leading spaces are simply cut out.

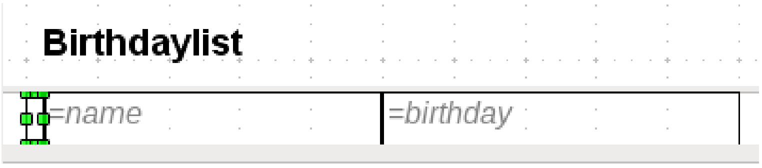
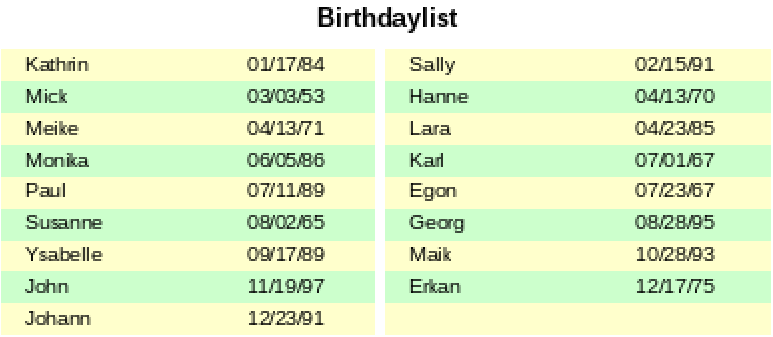


Figure 371: Using a separate text field to achieve consistent indentation

A more successful method is to position a text field before the actual text but without binding it to a data field. This text field is conditionally formatted in the same way as the others, so that a consistent visible indentation appears on printing (see Figure 371).

Two-column reports

With clever query techniques, you can make a report with multiple columns such that the horizontal sequence corresponds to successive records as shown in Figure 372:



Birthdaylist

Kathrin	01/17/84	Sally	02/15/91
Mick	03/03/53	Hanne	04/13/70
Meike	04/13/71	Lara	04/23/85
Monika	06/05/86	Karl	07/01/67
Paul	07/11/89	Egon	07/23/67
Susanne	08/02/65	Georg	08/28/95
Ysabelle	09/17/89	Maik	10/28/93
John	11/19/97	Erkan	12/17/75
Johann	12/23/91		

Figure 372: Example showing a two-column birthday list

The first record goes into the left column, the second into the right. Records are sorted according to the sequence of birthdays within the year.

Sorting by birthday makes the query for this report rather long. If the sort was by the primary key of the underlying table, it would be a lot shorter. The sort criterion uses a recurrent text block that is explained further below.

Here is the query that underlies the report:

```
SELECT "T1"."name" "name1", "T1"."birthday" "birthday1", "T2"."name" "name2",
"T2"."birthday" "birthday2"
FROM
   ( SELECT "name", "birthday", "rowsNr" AS "row" FROM
      ( SELECT "a".*,
         ( SELECT COUNT( "ID" ) FROM "birthdays" WHERE
         RIGHT( '0' || MONTH( "birthday" ), 2 ) ||
         RIGHT( '0' || DAY( "birthday" ), 2 ) || "ID"
         <= RIGHT( '0' || MONTH( "a"."birthday" ), 2 ) ||
         RIGHT( '0' || DAY( "a"."birthday" ), 2 ) || "a"."ID" )
         AS "rowsNr"      FROM "birthdays" AS "a" )
   WHERE MOD( "rowsNr", 2 ) > 0 )
AS "T1"
LEFT JOIN
   ( SELECT "name", "birthday", "rowsNr" - 1 AS "row" FROM
      ( SELECT "a".*,
         ( SELECT COUNT( "ID" ) FROM "birthdays" WHERE
         RIGHT( '0' || MONTH( "birthday" ), 2 ) ||
         RIGHT( '0' || DAY( "birthday" ), 2 ) || "ID"
         <= RIGHT( '0' || MONTH( "a"."birthday" ), 2 ) ||
         RIGHT( '0' || DAY( "a"."birthday" ), 2 ) || "a"."ID" )
         AS "rowsNr"
      FROM "birthdays" AS "a" )
   WHERE MOD( "rowsNr", 2 ) = 0 )
AS "T2"
ON "T1"."row" = "T2"."row"
ORDER BY "T1"."row"
```

There are two identical subqueries in this query. The first two columns relate to the subquery with the alias T1 and the last two to the subquery T2.

The subqueries provide another field (rowsNr) in addition to those in the Birthdays table, which makes it possible to distinguish lines and to sort the records. The main query uses this together with the line numbering facility in queries (see Chapter 5, Queries).

```
RIGHT( '0' || MONTH( "birthday" ), 2 ) || RIGHT( '0' || DAY( "birthday" ),
2 ) || "ID"
```

This formula allows a unique sequence of records to be established. As the example records are to be sorted by date, it would be easy to think that only the date should be used in the comparison. It is more difficult in practice because it is not the birth dates themselves but their sequence within a single year that counts. Additional problems are caused by identical date values which prevent a unique sequence from being established. Hence sorting is not only by month and day but, within that, by the unique primary key. And to prevent month 10 from coming before month 2, a leading zero is placed before each month using | | when the sort criterion is put together; if the month already has 2 digits, this is subsequently removed using `RIGHT( … , 2 )`.

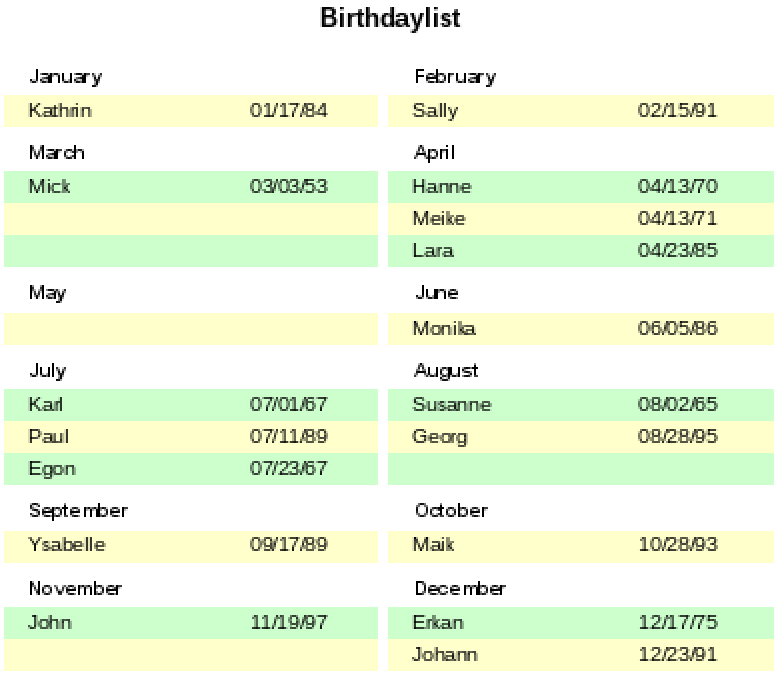
`SELECT COUNT( "ID" )` gives the number of records whose combination of month, day and primary key is less than or equal to that for the current record in the Birthdays table. Here we have a correlating subquery (see chapter on “Queries”).

`MOD( "rowsNr", 2 )` determines if this number is odd or even. `MOD` gives the remainder from a division, in the example, division by 2. This causes the rowsNr to give the values ‘1’ and ‘0’ alternately. This in turn distinguishes the queries for T1 and T2.

In the next query level of T2, Row is defined as `"rowsNr"–1`. In this way T1 and T2 are directly comparable.

T1 is linked to T2 using `LEFT JOIN`, so that all the dates in the Birthdays table are shown for odd record numbers too. When T1 and T2 are combined, direct reference can now be made to the actual rows: `"T1"."row" = "T2"."row"`.

Finally the whole content is sorted on the value of `row`, which is the same for T1 and T2. The report can also use this directly using its grouping function shown in Figure 373.



Birthdaylist

January		February	
Kathrin	01/17/84	Sally	02/15/91
March		April	
Mick	03/03/53	Hanne	04/13/70
		Meike	04/13/71
		Lara	04/23/85
May		June	
		Monika	06/05/86
July		August	
Karl	07/01/67	Susanne	08/02/65
Paul	07/11/89	Georg	08/28/95
Egon	07/23/67		
September		October	
Ysabelle	09/17/89	Maik	10/28/93
November		December	
John	11/19/97	Erkan	12/17/75
		Johann	12/23/91

Figure 373: Birthday list example using two columns to show birthdays grouped by month

It is much more complicated to create query techniques for making subdivisions in addition to the 2-column format. In such cases, empty lines occur in the middle of the report, whereas in the previous 2-column example, they occur only at the end. The Report Builder does not work well with this sort of query. Therefore instead of this, we will use two linked views.

First we create the following view:

```
SELECT "a"."ID", "a"."name", "a"."birthday",
   MONTH( "a"."birthday" ) AS "monthnumber",
      ( SELECT COUNT( "ID" ) FROM "birthdays"
         WHERE MONTH( "birthday" ) = MONTH( "a"."birthday" )
            AND RIGHT( '0' || DAY( "birthday" ), 2 ) || "ID" <=
            RIGHT( '0' || DAY( "a"."birthday" ), 2 ) || "a"."ID" )
   AS "monthcounter"
FROM "birthdays" AS "a"
```

All fields of the Birthdays table are included. In addition, the month is included as a number. The Birthdays table is given the alias “a”, so that the table can be accessed with a correlating subquery.

The subquery counts all records within a month whose date values have a smaller or equal day number. For identical dates, the primary key determines which comes first. This technique is the same as in the previous example.

The report_month_two_columns view accesses the monthnumber view. Only extracts of this view are given here:

```
SELECT
   "Tab1"."name" AS "name1",
   "Tab1"."birthday" AS "birthday1",
   1 AS "monthnumber1",
   IFNULL("Tab1"."monthcounter",999) AS "monthcounter1",
   'January' AS "month1",
   "Tab2"."name" AS "name2",
   "Tab2"."birthday" AS "birthday2",
   2 AS "monthnumber2",
   IFNULL("Tab2"."monthcounter",999) AS "monthcounter2",
   'February' AS "month2"
FROM
   (SELECT * FROM "monthnumber" WHERE "monthnumber" = 1) AS "Tab1"
   RIGHT JOIN (SELECT * FROM "monthnumber" WHERE "monthnumber" = 2) AS
   "Tab2" ON "Tab1"."monthcounter" = "Tab2"."monthcounter"
UNION
   SELECT "Tab1"."name" AS "name1",
   "Tab1"."birthday" AS "birthday1",
   1 AS "monthnumber1",
   IFNULL("Tab1"."monthcounter",999) AS "monthcounter1",
   'January' AS "month1",
   "Tab2"."name" AS "name2",
   "Tab2"."birthday" AS "birthday2",
   2 AS "monthnumber2",
   IFNULL("Tab2"."monthcounter",999) AS "monthcounter2",
   'February' AS "month2"
FROM
   (SELECT * FROM "monthnumber" WHERE "monthnumber" = 1) AS "Tab1"
   LEFT JOIN (SELECT * FROM "monthnumber" WHERE "monthnumber" = 2) AS
   "Tab2" ON "Tab1"."monthcounter" = "Tab2"."monthcounter"
UNION
...
ORDER BY "monthnumber1", "monthcounter1", "monthcounter2"
```

First the subquery reads from monthnumber all dates for which monthnumber = 1. This selection is given the alias Tab1 . At the same time, all dates for monthnumber = 2 are read into Tab2. These tables are linked with a `RIGHT JOIN`,so that all records from Tab2 are displayed but only those records from Tab1 with the same monthcounter as Tab2.

The columns of the view must have different names so that each column is provided with an alias. Also a 1 is entered directly as the monthnumber for Tab1 and January as the month1. These entries also appear when there is no record from Tab1 but still some records in Tab2. If there is no monthcounter available, the value 999 is entered. As Tab1 is linked to Tab2 by a `RIGHT JOIN`, it may happen that when there are fewer records in Tab1, empty fields are shown instead. Since the sorting later on would put such records ahead of all records with content, a very high number is entered instead.

The creation of columns for Tab2 is similar. Here though it is not necessary to use the code `IFNULL("Tab2"."monthcounter",999)`, since, with a `RIGHT JOIN` for Tab2, all rows from Tab2 will be displayed, but it will no longer be the case that more rows could be created from Tab1 than from Tab2.

Precisely this problem is solved by the linking of two queries. Using `UNION`, all records from the first query and all records from the second query are displayed. The records from the second query appear only if they are not identical with a previous record. This means that `UNION` works like `DISTINCT`.

Using `UNION`, the same query is asked again, only Tab1 and Tab2 are now linked with a `LEFT JOIN`. This causes all records from Tab1 to be displayed even if Tab2 contains fewer records than Tab1.

For months 3 and 4, 5 and 6, and so on, exactly equivalent queries are used and again attached to the previous query using a `UNION`.

Finally the result of the view is sorted by monthnumber1, monthcounter1, and monthcounter2. There is no need to sort by monthnumber2, because monthnumber1 already gives the same record sequence.

Sources of errors in reports

The Report Builder sometimes hides errors whose exact cause cannot easily be determined afterwards. Here are a few sources of error and useful counter-measures.

The content of a field from a query does not appear

A database is set up to simulate the sale of stock. A query calculates a total price from the number of items bought and the unit price.

```
SELECT "sales"."sum", "stock"."stock", "stock"."Price",
"sales"."sum"*"stock"."Price" FROM "sales", "stock"
WHERE "sales"."stock_ID" = "stock"."ID"
```

This query serves as the basis of the report. If however the field `"sales"."sum"*"stock"."Price"` is called up in the report, it remains empty. If on the other hand it is provided with an alias in the query, the Report Builder can easily access it:

```
SELECT "sales"."sum", "stock"."stock", "stock"."Price",
"sales"."sum"*"stock"."Price" AS "TPrice"
FROM "sales", "stock" WHERE "sales"."stock_ID" = "stock"."ID"
```

The field now accesses "Tprice" and presents the corresponding value.

A report cannot be produced

Sometimes a report can be prepared but then not produced or even saved. A rather uninformative error message appears:

“Report could not be produced. An exception of type com.sun.star.lang.WrappedTargetExeption was discovered”

Here it can be helpful to read the additional information attached to the message. If you see there a reference to SQL, it means that the Report Builder cannot interpret correctly the SQL code in the data source.

It may help here to use the Report Navigator to set Data > Analyze SQL command > No. However, this solution causes existing groups to stop functioning.

A better way is to use a view rather than a query as the basis for your report. This is set up in such a way that it looks like a table to the Report Builder and can be edited without any difficulty. Even the sorting requirements of the view function smoothly.

Base Guide 7.3

Chapter 7
Linking to Databases

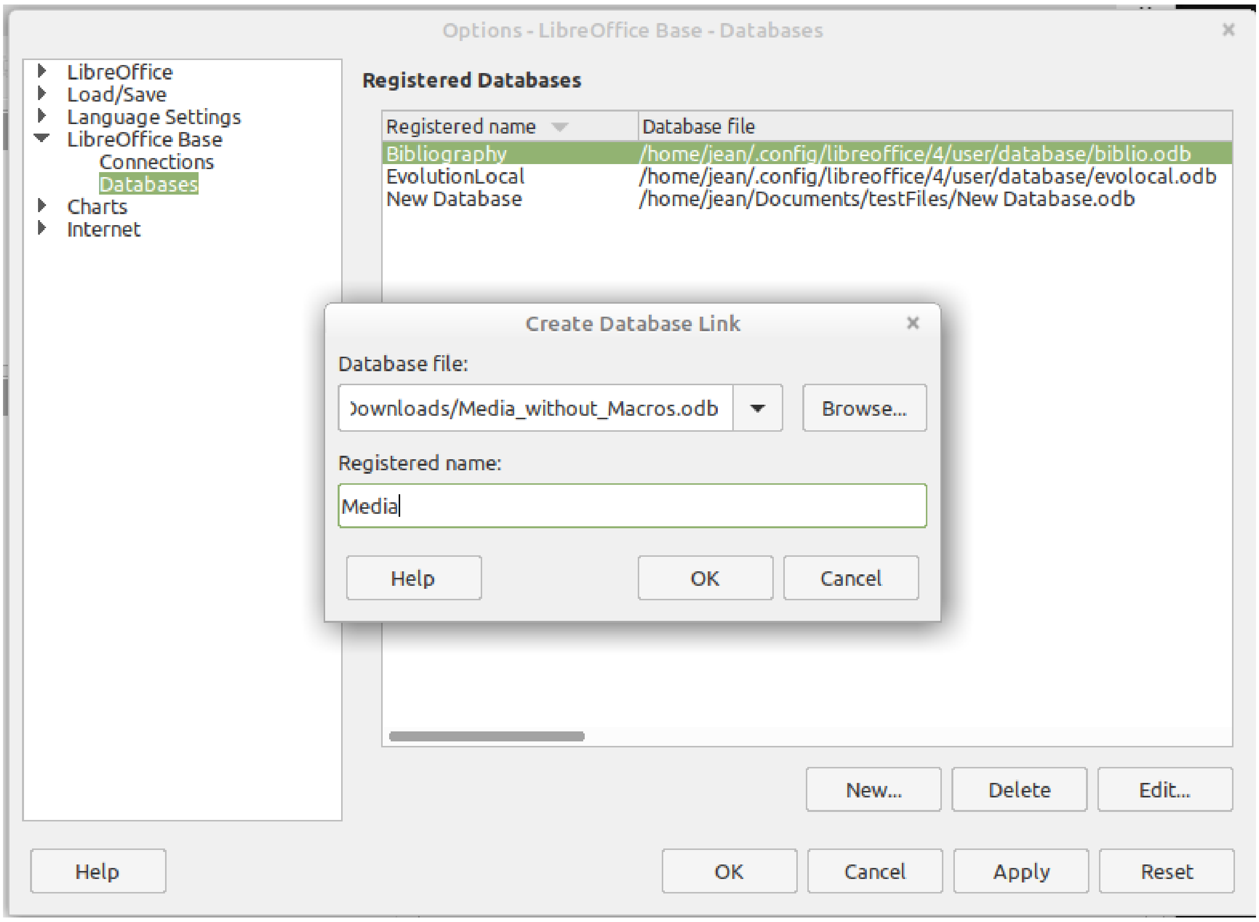
General notes on database linkage

With Base, you can use documents in LibreOffice Writer and Calc in various ways as data sources. This means that the use of Base is not necessarily tied to the registration of databases in the configuration of LibreOffice. External forms can also interact directly with Base, provided that the path to the data sources is supplied.

Registration of databases

Many functions, such as printing labels or using data for form letters, require the registration of a database in the configuration of LibreOffice.

Using **Tools > Options > LibreOffice Base > Databases > New**, you can register a database for subsequent use by other LibreOffice components.



Registered name	Database file
Bibliography	/home/jean/.config/libreoffice/4/user/database/biblio.odb
EvolutionLocal	/home/jean/.config/libreoffice/4/user/database/evolocal.odb
New Database	/home/jean/Documents/testFiles/New Database.odb

Database file:)ownloads/Media_without_Macros.odb

Registered name: Media

Figure 374: Registering a database for use by other LibreOffice components

Find the database using a file browser and connect it to LibreOffice in a similar way as for a simple form. Give the database itself a suitably informative registered name, for example the name of the database file. The name serves as an alias, which can also be used in queries to the database.

Data source browser

The data source browser in Writer and Calc provides access to tables and queries of all registered databases under their registered names. To open the browser, use **View > Data Sources**, or press the *Ctrl+Shift+F4* keys, or click the icon on the Standard toolbar.

The Data Sources icon is not usually visible on the Standard toolbar. To make it visible, right-click the Save button to open this toolbar. Scroll down to *Visible Buttons*. This opens a list of all of the buttons. Scroll down to near the bottom to find the Data Sources button. Click it to make it visible at the right end of the toolbar.

Tip

If you are using a laptop, you may need to press *Ctrl+fn+shift+F4*. The fn key (function) permits the use of the F keys for more than one feature.

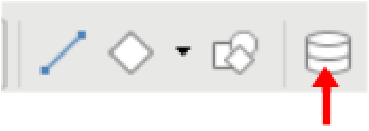


Figure 375: Data Sources button on Standard toolbar

Registered data sources are shown on the left side of the data source browser, which is located by default at the top of the workspace. The Bibliography data source is included in LibreOffice by default. Other data sources are listed by their registered names.

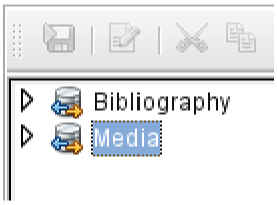
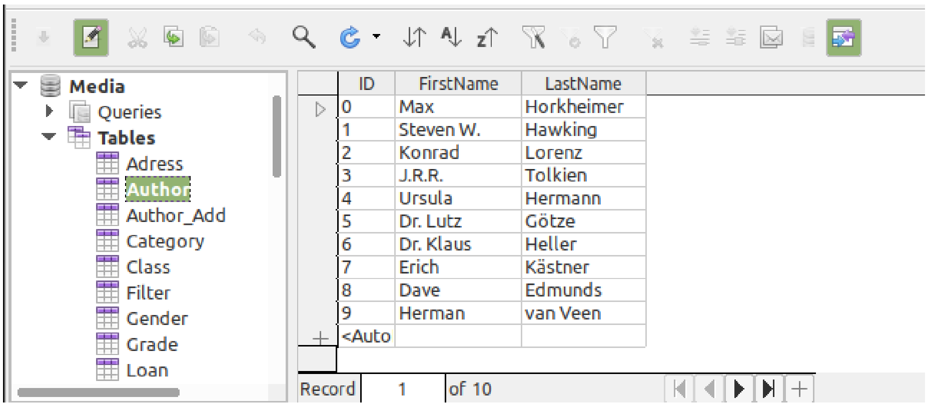


Figure 376: Registered data sources shown in name tree on left side of data source browser

Click on the expansion sign in front of the database name to open the database and show sub-folders for queries and tables. Other sub-folders of the database are not made available here. Internal forms and reports can only be accessed by opening the database itself.

Only when you click on the Tables folder is the database actually accessed. For databases protected by a password, the password must be entered at this point.

To the right of the name tree, you can see the table you have selected. It can be edited just as in Base. However, direct entry into tables should be carried out with caution in very complex relational databases, as the tables are linked together with foreign keys. For example, the database shown below has separate tables for street names, postcodes, and towns.



ID	FirstName	LastName
0	Max	Horkheimer
1	Steven W.	Hawking
2	Konrad	Lorenz
3	J.R.R.	Tolkien
4	Ursula	Hermann
5	Dr. Lutz	Götze
6	Dr. Klaus	Heller
7	Erich	Kästner
8	Dave	Edmunds
9	Herman	van Veen
<Auto		

Figure 377: Select table in name tree to view constituent data on right side of data source browser

For a proper view of the data (but without the ability to edit), queries or views are more suitable.

Many of the icons in the toolbar (Figure 378) will be familiar from data entry into tables. (The icons on your display may be different from those shown in the figure.)

The main new icons are those in the last section: Data to Text, Data to Fields, Mail Merge, Data Source of Current Document, Explorer On/Off. Their use is described below, using the Reader table of the Media database.

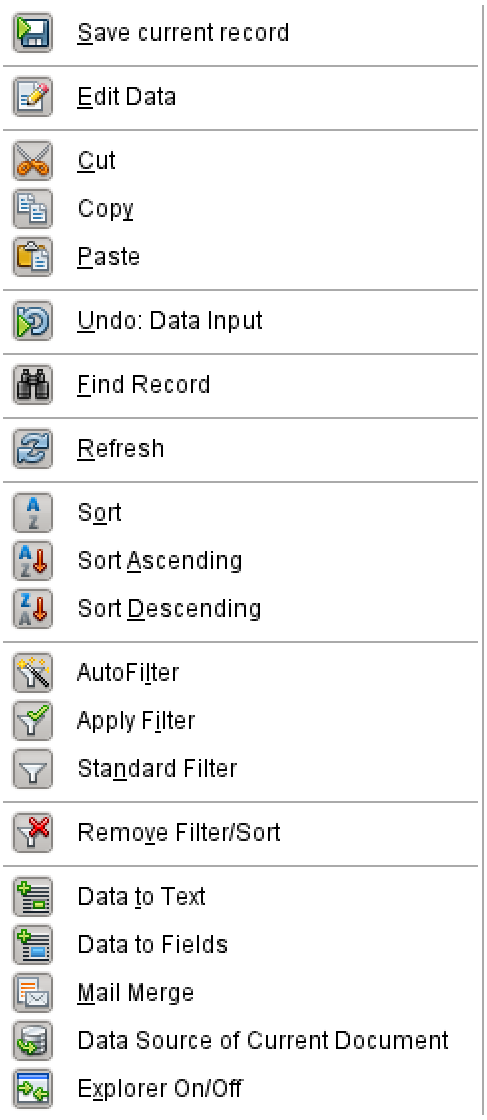


Figure 378: Default allocation of buttons on Table Data toolbar of data source browser

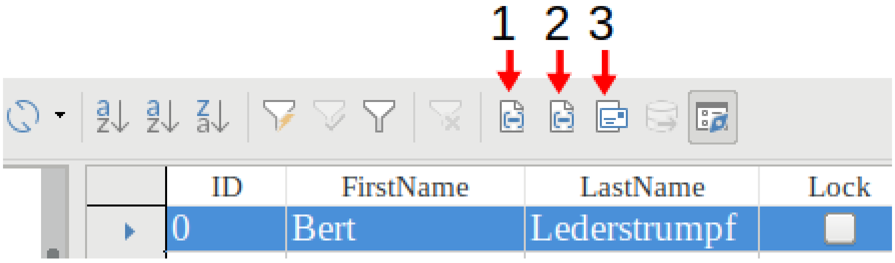
Data to Text

Tip

With this method, data can be inserted directly into specific places in a text document or specific cells of a spreadsheet. While the data could be typed into these locations, inserting it guarantees the accuracy of it. This is important when using mail merge which is discussed later.

When sending the same document to different people, this guarantees all will receive the exact same data.

Select one or more records to activate the Data to Text and Data to Fields functions.



ID	FirstName	LastName	Lock
0	Bert	Lederstrumpf	□

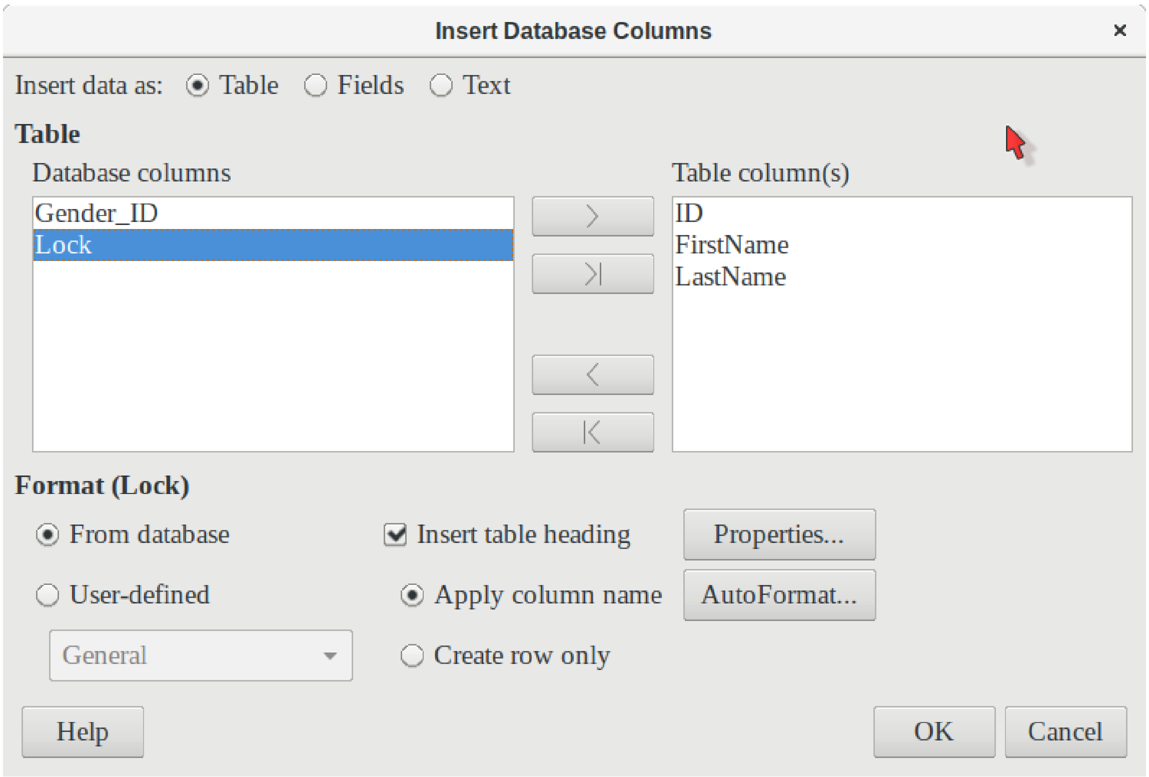
1	*Data to Text*	*2*	*Data to Fields*	*3*	*Mail Merge*

Figure 379: Select data record to activate Data to Text icon on Table Data toolbar

If you now choose **Data to Text**, a Wizard appears to carry out the necessary formatting (Figure 380).

The three choices for entering data as text are: as a table, as single fields, or as ordinary text.

Figure 380 shows the option **Insert Data as Table**. In the case of numeric and date fields, the database format can be changed to a chosen format. Otherwise, formatting is carried out automatically when the table fields are selected. The sequence of fields is adjusted by using the arrow keys.



Insert data as: ● Table ○ Fields ○ Text

Table

Database columns: Gender_ID, Lock (selected)

Table column(s): ID, FirstName, LastName

Format (Lock)

● From database

○ User-defined

General

☑ Insert table heading

● Apply column name

○ Create row only

Figure 380: Insert Database Columns dialog with Table radio button selected

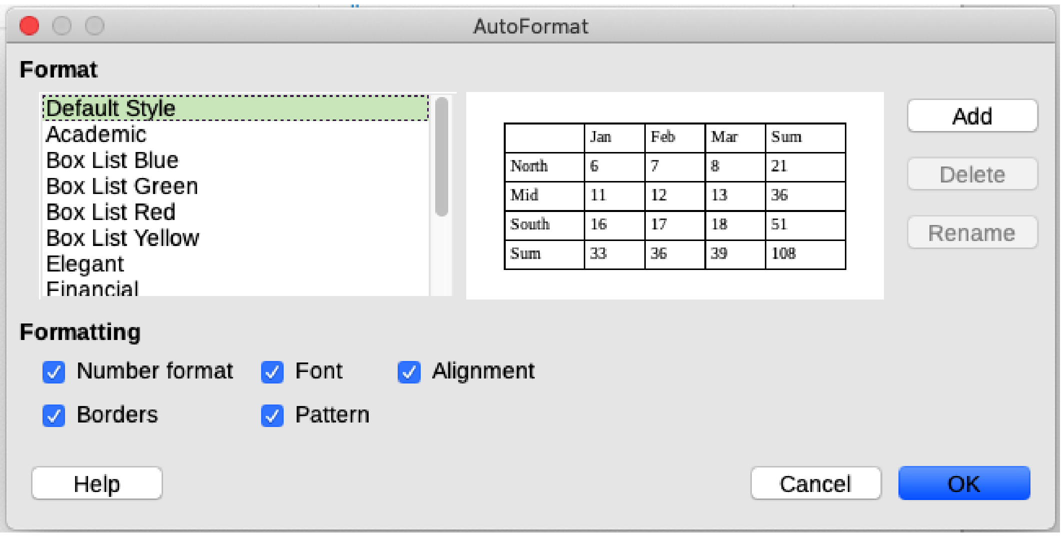
As soon as table columns have been selected, the **Properties** button for the table is activated. This allows you to set the usual table properties for Writer (table width, column width, and so on).

The **Insert table heading** checkbox determines if a table heading is required. If it is not checked, no separate row will be reserved for headings.

The row chosen for the table heading can be taken from the column names, or the record may be written out with space left for the headings to be edited in later. Choose the **Create row only** option.

You can use the **AutoFormat** button to open a dialog with several pre-formatted table styles. Apart from the suggested Default Style, all the formats can be renamed. (You can also add autoformats; to do this, first create a table in the required format. Then select the table and click the **Add** button to add its format to the list.) You can also format the table in Writer by selecting it and choosing a format from the Table Styles list in the Styles deck of the sidebar; the list of table styles is the same as the list of formats in the AutoFormat dialog.

To create the table with the selected records and columns, click **OK** in the Insert Database Columns dialog.



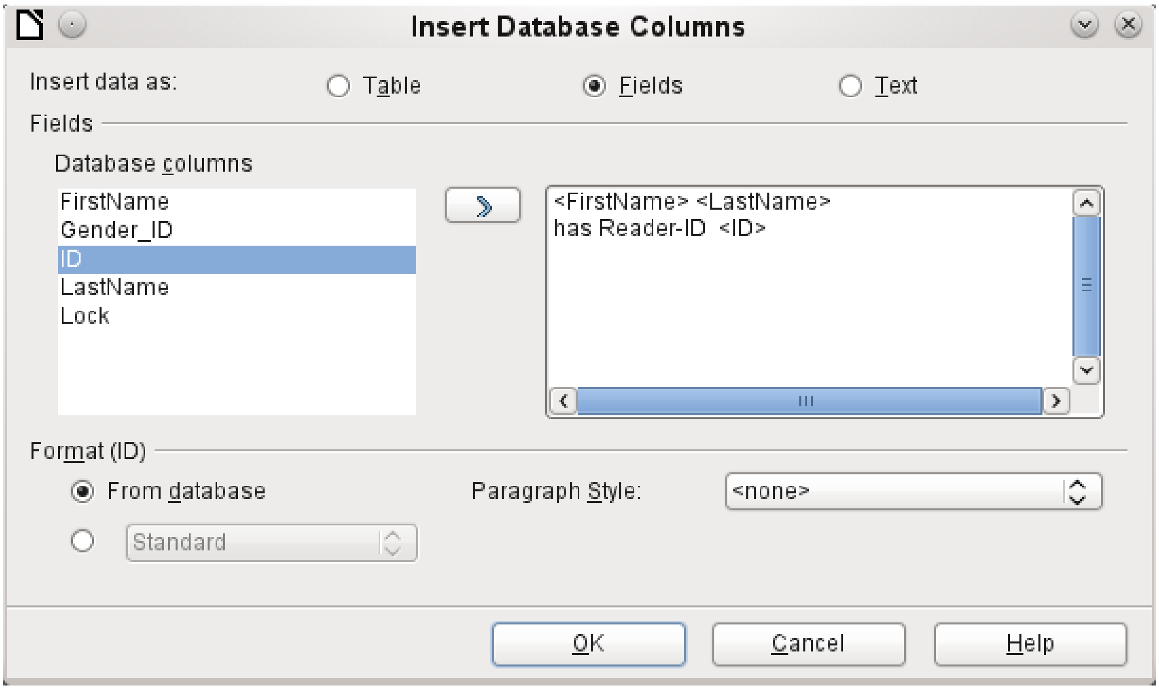
Default Style
Academic
Box List Blue
Box List Green
Box List Red
Box List Yellow
Elegant
Financial

	Jan	Feb	Mar	Sum
North	6	7	8	21
Mid	11	12	13	36
South	16	17	18	51
Sum	33	36	39	108

☑ Number format ☑ Font ☑ Alignment
☑ Borders ☑ Pattern

Figure 381: AutoFormat dialog

Insert Data as Fields provides the possibility of using a mini-editor to position the various table fields successively in the text. The text created in this way can also be provided with a paragraph style. In this case too, the formatting of dates and numbers can be specified separately, or can be read directly from the table settings in the database.



Insert data as: ○ Table ● Fields ○ Text

FirstName
Gender_ID
ID
LastName
Lock

<FirstName> <LastName>
has Reader-ID <ID>

● From database
○ Standard

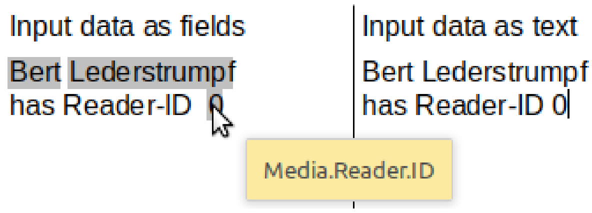
Paragraph Style: <none>

Figure 382: Insert Database Columns dialog with Fields radio button selected

The fields inserted into the text in this way can subsequently be deleted singly or used for a mail merge.

If you choose **Insert data as Text**, the only difference from using fields is that fields remain linked to the database. When you insert as text, only the content of the specified fields is transferred and not the link to the actual database.

The results of the two procedures are compared below.



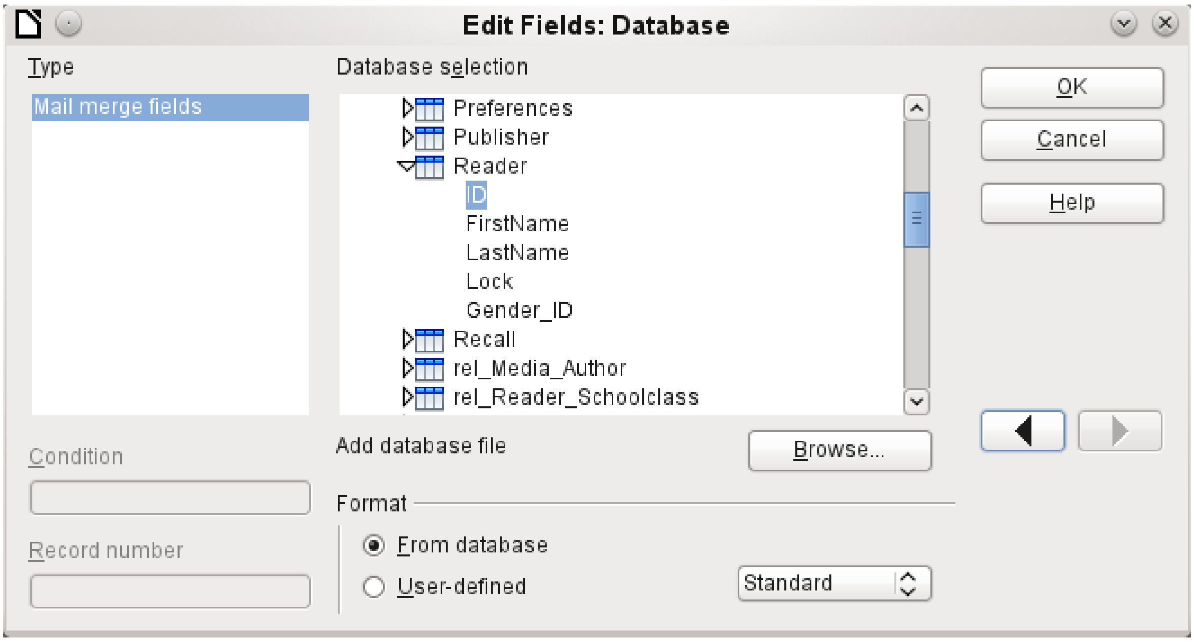
Input data as fields	Input data as text
Bert Lederstrumpf has Reader-ID 0	Bert Lederstrumpf has Reader-ID 0

Figure 383: Comparison of Data as Fields and Data as Text options

The fields have a gray background. If you hover the mouse cursor over the fields, a tooltip shows that the fields are linked to the Media database, to the Reader table and, within this table, to the ID field.

So, for example, a double-click on the ID field opens the following overview. This makes it clear which field was created through the Insert Data as Fields procedure. It is the same field type that is shown by **Insert > Fields > More Fields > Database**.

It is simpler to create such a field by selecting the column header of the table in the data source browser and dragging it into the document with the mouse. You can create a form letter directly in this way.



Edit Fields: Database

Type: Mail merge fields

Database selection:

- Preferences
- Publisher
- Reader
  - ID
  - FirstName
  - LastName
  - Lock
  - Gender_ID
- Recall
- rel_Media_Author
- rel_Reader_Schoolclass

Condition:

Record number:

Add database file

Format

● From database

○ User-defined

Standard

Figure 384: Double-click an inserted field to access Edit Fields dialog

Data to Fields

Tip

This method is useful when sending a document to several people in which each will be sent some data that is specific only for them. This is done through mail merge.

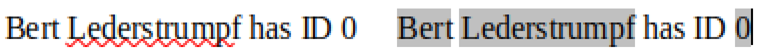
For example, libraries send out notices to people listing the media they have borrowed and not returned on time. The list will usually be different for each person, but all will receive a warning of some kind. Of course, the type of warning will depend upon the length of time since the media was due. Everyone who falls with a particular time period will receive the same warning.

To insert data to fields (see Figure 380):

3) Click on the left of one of the test rows to highlight it.
4) Click the **Data to Text** button to open the **Insert Database Columns** wizard.
5) Choose the option button, **Fields**.
6) Move the fields of the database that you want to use from the left list to the right one in the order you desire. Text can be added as well as in Figure 382.
7) To apply a specific paragraph style to these fields, select it from the Paragraph styles drop-down list.
8) Click **OK**.

Tip

Inserting data to text is done the same way as inserting data to fields. The only difference is whether you select Text or Fields respectively. The difference is in the appearance in the document or spreadsheet as shown below. The left one is data to text; the right one is data to fields.

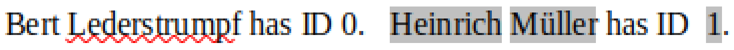


Bert Lederstrumpf has ID 0 Bert Lederstrumpf has ID 0

Figure 385: Left side inserted as text; right side inserted as fields

Tip

Once a field has been inserted, the values it displays can be changed. Select the record (row) with ID = 1 and then click the **Data to Fields** button.



Bert Lederstrumpf has ID 0. Heinrich Müller has ID 1.

Figure 386: Data inserted as fields can change

Mail merge

The Mail Merge button launches the Mail Merge Wizard. A form letter assembles its data from different tables, so you need first to launch the database. In the database, you then create a new query to make the required data available.

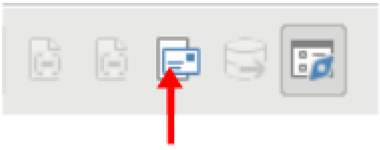


Figure 387: Mail Merge icon on Table Data toolbar

To launch the database, right-click on the database itself or on one of its tables or queries; this immediately refreshes the display in the data source browser. After that the Mail Merge Wizard can be called up by clicking the corresponding button.

Data source of current document

Click on the **Data Source of Current Document** button to open a direct view of the table which forms the basis for the data inserted into the document. In the above example, the Reader table from the Media database appears.

Explorer on/off

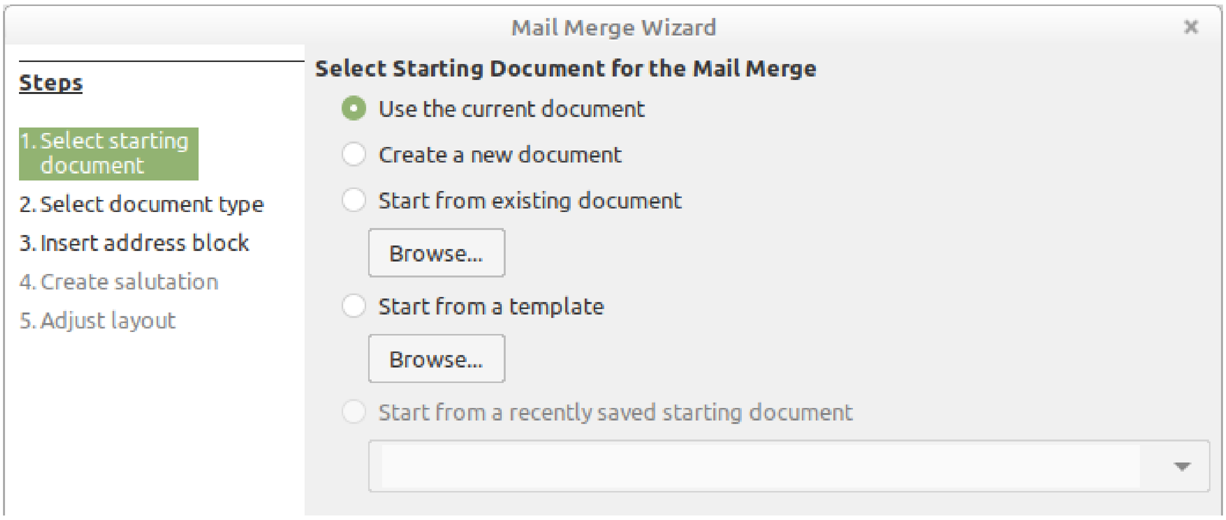
Toggling the **Explorer On/Off** button shows or hides the directory tree on the left of the table view. This allows more space, if necessary, for a display of the data. To access another table, you need to switch the Explorer back on.

Creating mail merge documents

The Mail Merge Wizard is also accessible from the database browser. This Wizard allows the address field and the salutation to be constructed from a data source in small steps. In principle you can create these fields without using the Wizard. Here we will work through the steps of the Wizard as an example. The following steps use the Wizard to do this. This time, a query will be the data source, specifically Media’s *Readeradresses*. Search for it in the Query drop-down list as you did earlier for the Reader table in the Table drop-down list.

Tip

Remove any text document in Writer if it has links to a database. You can not establish a new link to that document when the old one is still active. Begin with a new document, which can be untitled, or a template letter that contains no links.



Mail Merge Wizard

Steps

1. Select starting document
2. Select document type
3. Insert address block
4. Create salutation
5. Adjust layout

Select Starting Document for the Mail Merge

● Use the current document

○ Create a new document

○ Start from existing document

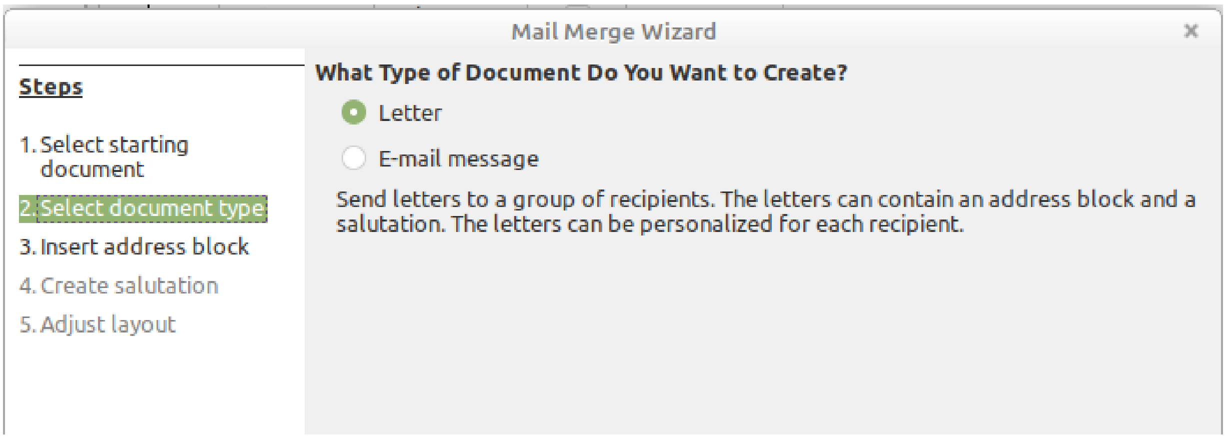
○ Start from a template

○ Start from a recently saved starting document

Figure 388: Mail Merge Wizard - step 1

The *Starting document* for the form letter is the document to which the *database fields will be linked*.

The *Merged document* is the one containing the data for the various people who are to receive the form letters. In the merged document there is *no linkage* to the data source. It is similar to the output of Insert Data as Text.



What Type of Document Do You Want to Create?

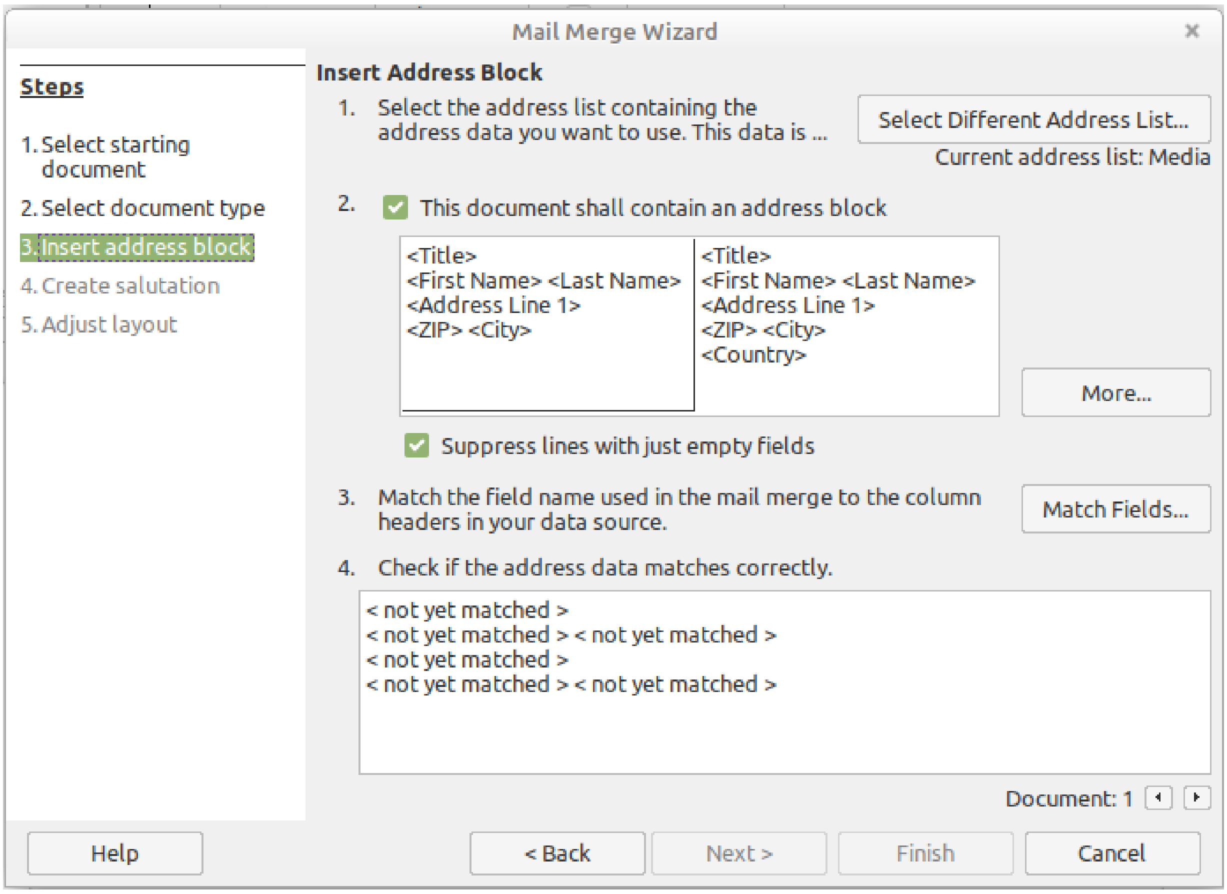
● Letter

○ E-mail message

Send letters to a group of recipients. The letters can contain an address block and a salutation. The letters can be personalized for each recipient.

Figure 389: Mail Merge Wizard - step 2

The Mail Merge Wizard can produce either letters or emails using records from the database. In this example we will create letters using the Reader table of the Media database.



Insert Address Block

1. Select the address list containing the address data you want to use. This data is ...

Current address list: Media

2. ☑ This document shall contain an address block

```
<Title>                      <Title>
<First Name> <Last Name>     <First Name> <Last Name>
<Address Line 1>             <Address Line 1>
<ZIP> <City>                 <ZIP> <City>
                             <Country>
```

☑ Suppress lines with just empty fields

3. Match the field name used in the mail merge to the column headers in your data source.

4. Check if the address data matches correctly.

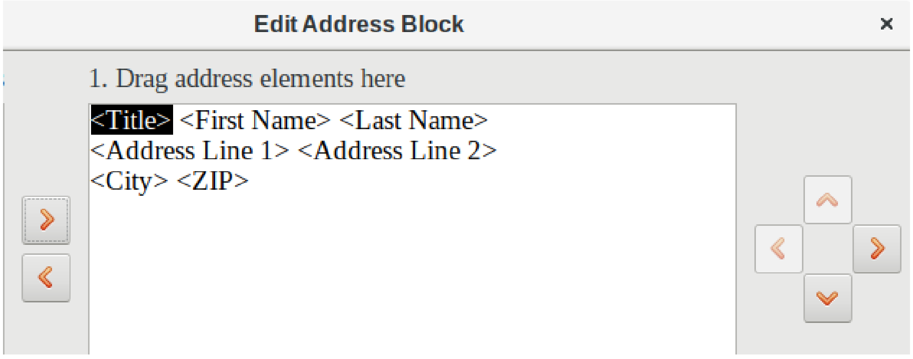
```
< not yet matched >
< not yet matched > < not yet matched >
< not yet matched >
< not yet matched > < not yet matched >
```

Document: 1

Figure 390: Mail Merge Wizard – step 3

The entry of the address block allows the most extensive configuration. The suggested address list comes from the currently selected query or table in the currently selected database.

Step 3 determines the overall look of the address block, which can be customized further by clicking the **More** button. See Figure 390. The left Address is already selected, and this block will be used.



<Title> <First Name> <Last Name>
<Address Line 1> <Address Line 2>
<City> <ZIP>

Figure 391: Changing contents of address block using Edit Address Block dialog

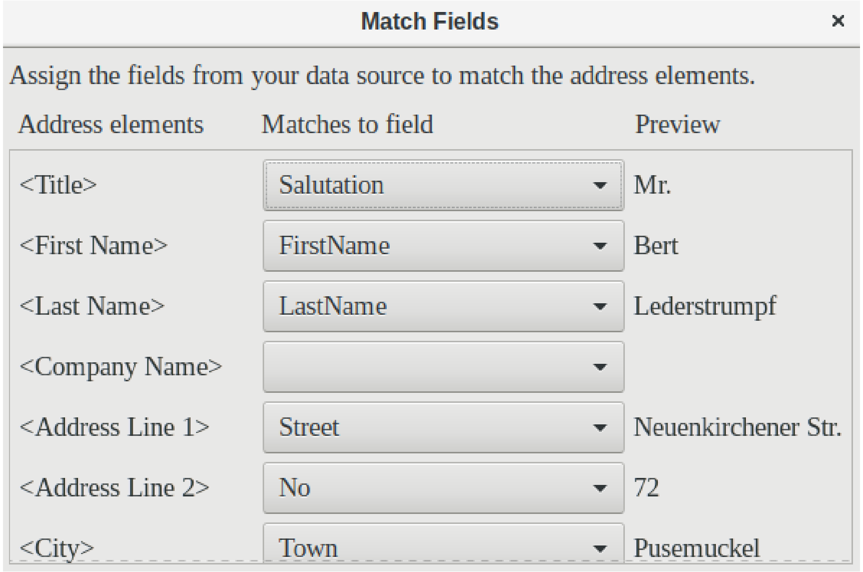
One element needs to be added: <Address Line 2>. To do this:

1) Click the **More** button.
2) Drag the <Address Line 2> element in the Address elements list to place it to the right of <Address Line 1>.
3) If there is not a space between these two elements, click the right arrow on the right side of the dialog to create the space.
4) Click **OK**.
5) Select Address Block: click **OK**.

The <Title> element needs to be moved down one line placing it before <First Name> <Last Name>. Make sure to put a space between <Title> and First Name>. Use the four arrows on the right to move first <Last Name> and then <First Name>.

Step 3 serves to link the named fields in the address block to the correct fields in the database. The Wizard initially recognizes only those database fields which have exactly the same names as those the Wizard uses. In this example, none of the fields do match, so all will have to be selected from the drop-down lists in this step.

- For <Title>, select Salutation.
- For <FirstName> select First Name.
- For <LastName> select Last Name
- For <Address Line 1> select Street.
- For <Address Line 2> select No.
- For <City> select Town.
- For <Zip> select Postal code.



Match Fields

Assign the fields from your data source to match the address elements.

Address elements	Matches to field	Preview
<Title>	Salutation	Mr.
<First Name>	FirstName	Bert
<Last Name>	LastName	Lederstrumpf
<Company Name>		
<Address Line 1>	Street	Neuenkirchener Str.
<Address Line 2>	No	72
<City>	Town	Pusemuckel

Figure 392: Matching data source fields to address elements using the Match Fields dialog

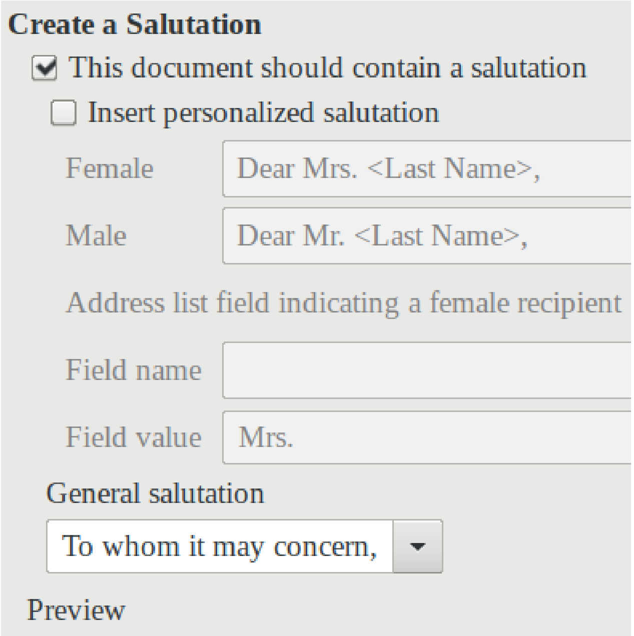
Here the address elements are associated with the corresponding elements from the query of the database successfully transferred by the Mail Merge Wizard. Again the first record in the query is used for the preview.

The database settings are essentially ended with Step 4. Here, it is just a matter of choosing which field the gender of the recipient should be taken from. This field has already been named, so that only the field content for a female recipient still needs to be specified.

Note

Because the wizard has a bug at this point, the personal salutation is created using Data to Text as described above. Specifically, the field, Salutation, will provide the proper title for each person. The rest of the salutation is created by typing it onto the merge document.

1) To finish this page, uncheck *Insert personalized salutation*.
2) Make no change in the *General salutation*. It will be replaced later, but it is needed to identify where the salutation should be in the letter.



Create a Salutation

☑ This document should contain a salutation

□ Insert personalized salutation

Female Dear Mrs. <Last Name>,

Male Dear Mr. <Last Name>,

Address list field indicating a female recipient

Field name

Field value Mrs.

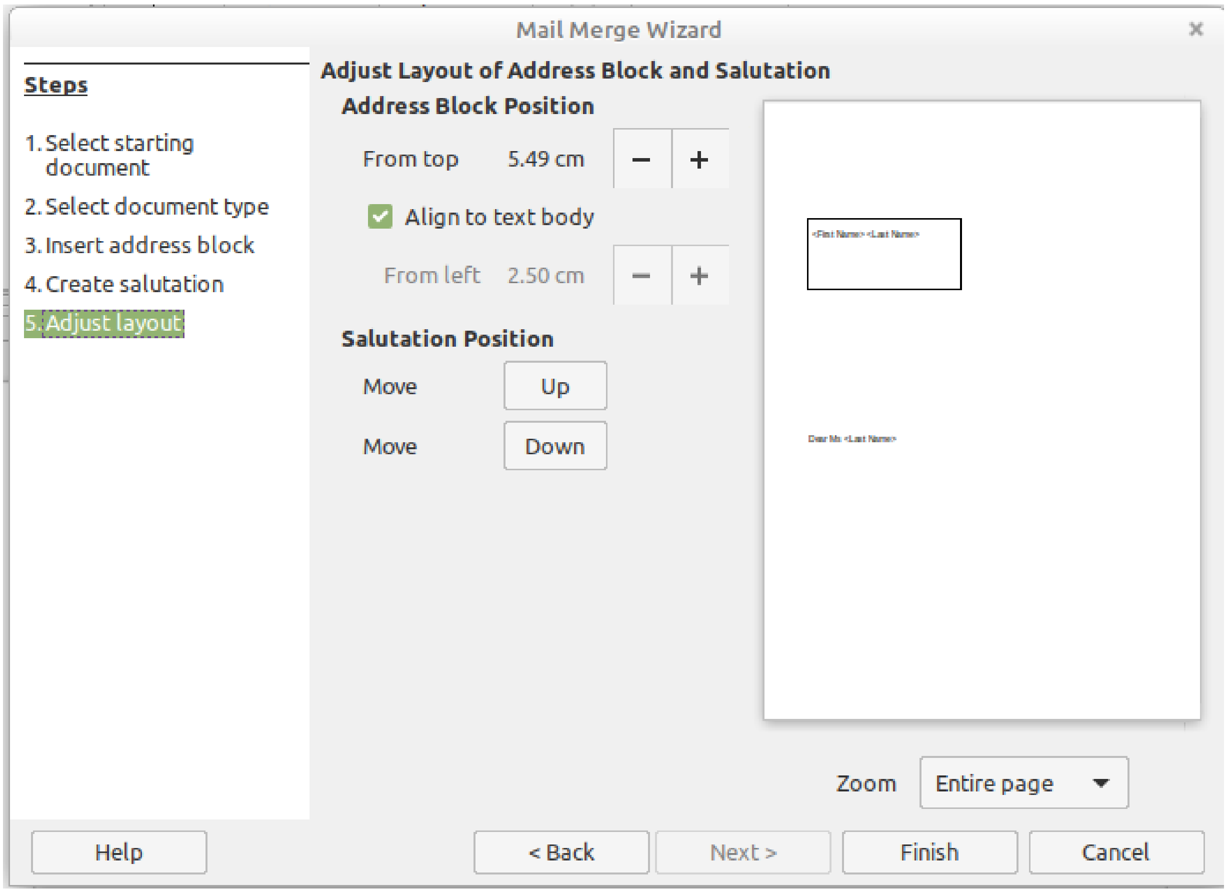
General salutation

To whom it may concern,

Preview

Figure 393: Creating a salutation at step 4 of Mail Merge Wizard

Click **Next >**. In Step 5, you can adjust the position of the address block and salutation on the page. (See Figure 394.) Then click **Finish**.



Mail Merge Wizard

Steps

1. Select starting document
2. Select document type
3. Insert address block
4. Create salutation
5. Adjust layout

Adjust Layout of Address Block and Salutation

Address Block Position

From top 5.49 cm

☑ Align to text body

From left 2.50 cm

Salutation Position

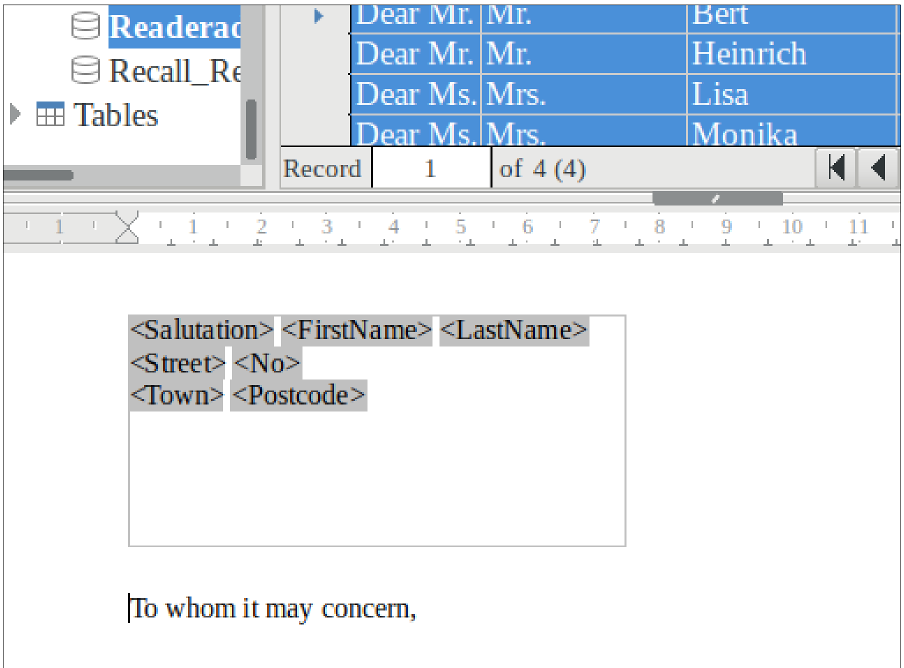
Move Up

Move Down

Zoom Entire page

Figure 394: Mail Merge Wizard – step 5

Now to finish the layout on the mail merge document. It contains the Address Block fields where you have placed them.



Dear Mr.	Mr.	Bert
Dear Mr.	Mr.	Heinrich
Dear Ms.	Mrs.	Lisa
Dear Ms.	Mrs.	Monika

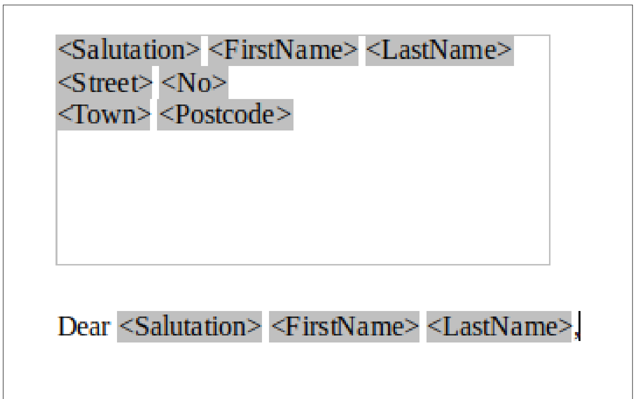
<Salutation> <FirstName> <LastName>
<Street> <No>
<Town> <Postcode>

To whom it may concern,

Figure 395: Writer document after completing interaction with the Mail Merge Wizard

Now use Data to Text to replace the salutation.

1) Replace *To whom it may concern* with *Dear*.
2) Drag-and-drop <Salutation> to one space after Dear.
3) Drag-and-drop <FirstName> to one space after <Salutation>.
4) Drag-and-drop <LastName> to one space after <FirstName>.

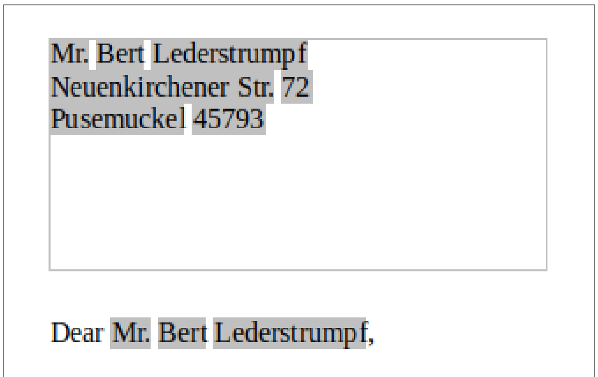


<Salutation> <FirstName> <LastName>
<Street> <No>
<Town> <Postcode>

Dear <Salutation> <FirstName> <LastName>,

Figure 396: Updated salutation

Select the top record in the Data Source window at its beginning. Then click the Data to Text button to see data entered into the fields.

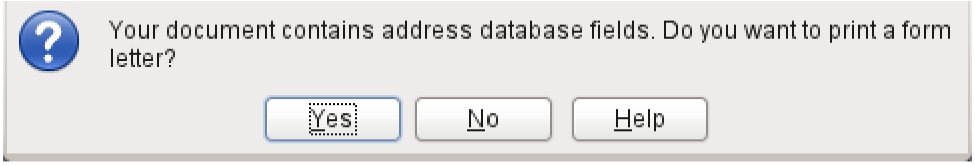


Mr. Bert Lederstrumpf
Neuenkirchener Str. 72
Pusemuckel 45793

Dear Mr. Bert Lederstrumpf,

Figure 397: Data from first data source record shown in Writer document

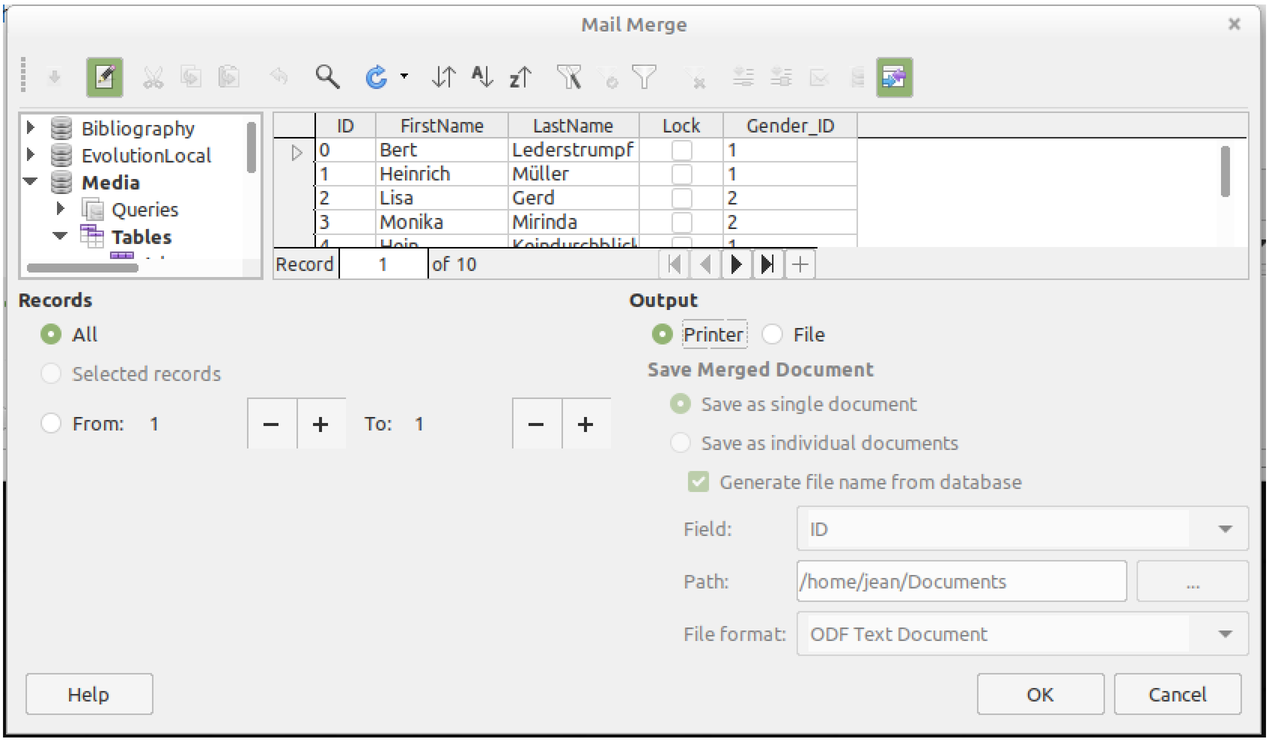
You now have a Writer document into which you can type the contents of the letter. To merge the fields and print the letters, choose **File > Print** from the Menu bar. The following message pops up. Click **Yes**.



Your document contains address database fields. Do you want to print a form letter?

Figure 398: Printing form letters confirmation dialog

The Mail Merge dialog (Figure 399) is now displayed, where you can optionally select records to include or exclude, and choose to print the letters or save them to a file. For more details, see Chapter 14, Mail Merge, in the *Writer Guide*.



ID	FirstName	LastName	Lock	Gender_ID
0	Bert	Lederstrumpf	□	1
1	Heinrich	Müller	□	1
2	Lisa	Gerd	□	2
3	Monika	Mirinda	□	2
4	Hein	Keindurchblick	□	1

Record 1 of 10

Records

● All
○ Selected records
○ From: 1 − + To: 1 − +

Output

● Printer ○ File

Save Merged Document

● Save as single document
○ Save as individual documents
☑ Generate file name from database

Field: ID
Path: /home/jean/Documents ...
File format: ODF Text Document

Figure 399: Mail Merge dialog

Label printing

File > New > Labels launches the Labels Wizard. It opens a dialog, which includes all questions of formatting and content for labels, before the labels themselves are produced. The settings in this dialog are saved in the personal settings for the user.

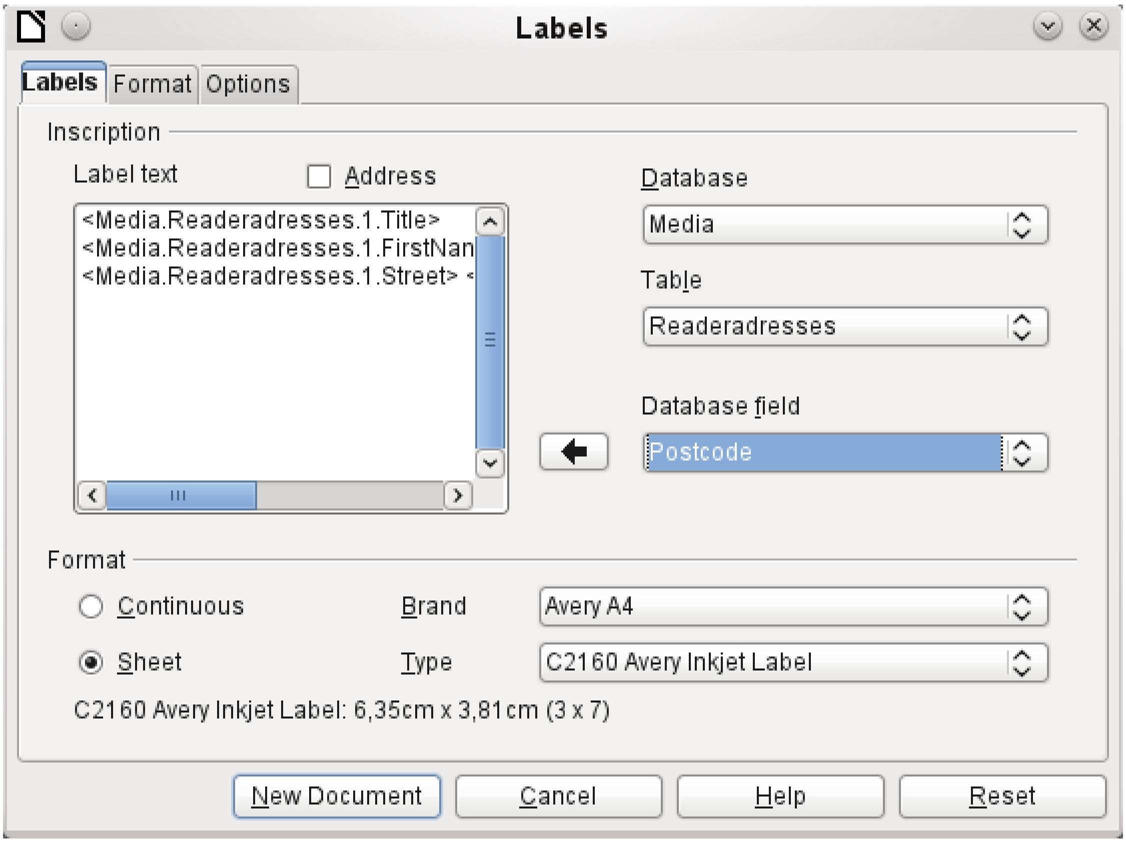
The basic settings for the content are in the Labels tab (Figure 400). If for Label text you check the Address box, all the labels will have the same content, taken from the LibreOffice settings for the user of the program.

As an example, we will again use the Addresses database. Although the next selection field is headed Tables, Tables and Queries are both listed here, just as in the data source browser.

Use the arrow buttons to insert individual database fields into the editor. The name for the database field Surname is set here to <Addresses.MailMergeQuery.1.Surname>. The sequence is thus <database.Table.1.database field>.

You can work with the keyboard in the editor. So for example, you can insert a line break at the beginning, so that the labels will not be printed directly on the top edge but the content can be printed as completely and clearly visible.

The format can be selected in the Labels tab. Here many label brands are incorporated so that most other settings in the Format tab are not necessary.



Inscription

Label text ☐ Address

```
<Media.Readeradresses.1.Title>
<Media.Readeradresses.1.FirstNan
<Media.Readeradresses.1.Street> <
```

Database: Media

Table: Readeradresses

Database field: Postcode

Format

○ Continuous

● Sheet

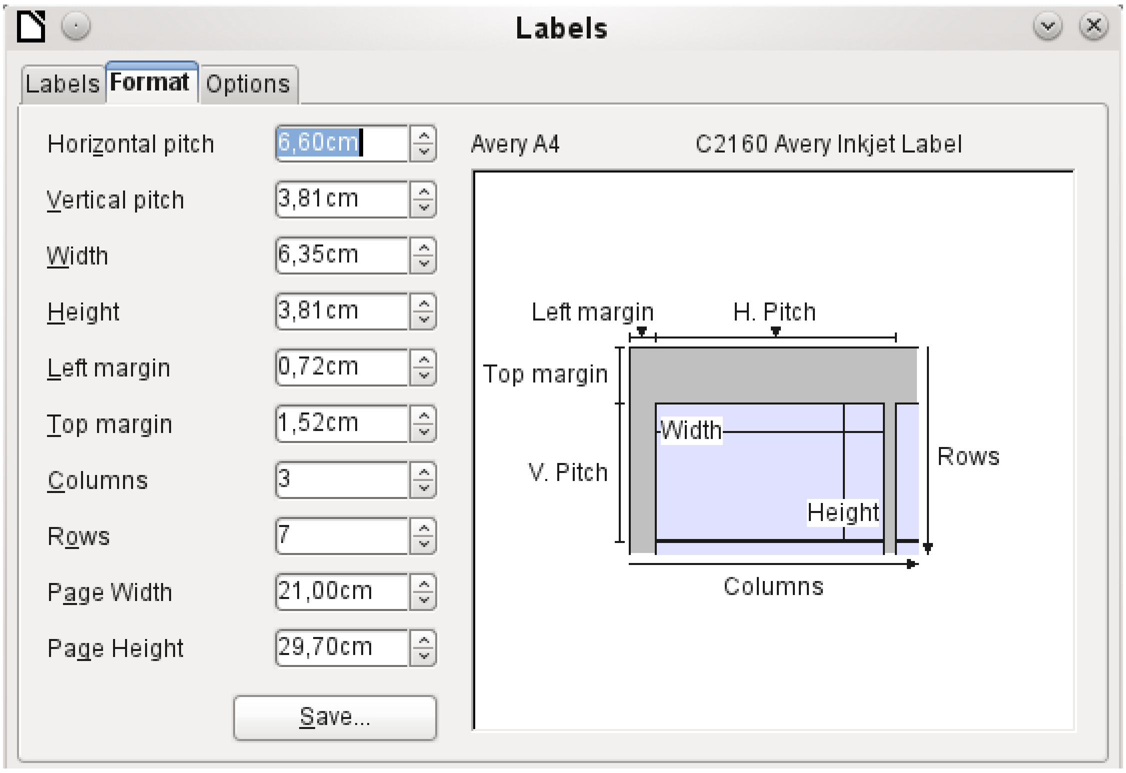
Brand: Avery A4

Type: C2160 Avery Inkjet Label

C2160 Avery Inkjet Label: 6,35cm x 3,81cm (3 x 7)

Figure 400: Labels dialog, Labels tab

Use the Format tab (Figure 401) to set the label size accurately. The settings are only significant when the make and type of the labels is not known. Note that, to print labels 7.00 cm wide, you need a page width a little bigger than 3*7.00 cm = 21.00 cm. Only then will three labels be printed in a row on the page.



Labels

Labels | Format | Options

Horizontal pitch 6,60cm

Vertical pitch 3,81cm

Width 6,35cm

Height 3,81cm

Left margin 0,72cm

Top margin 1,52cm

Columns 3

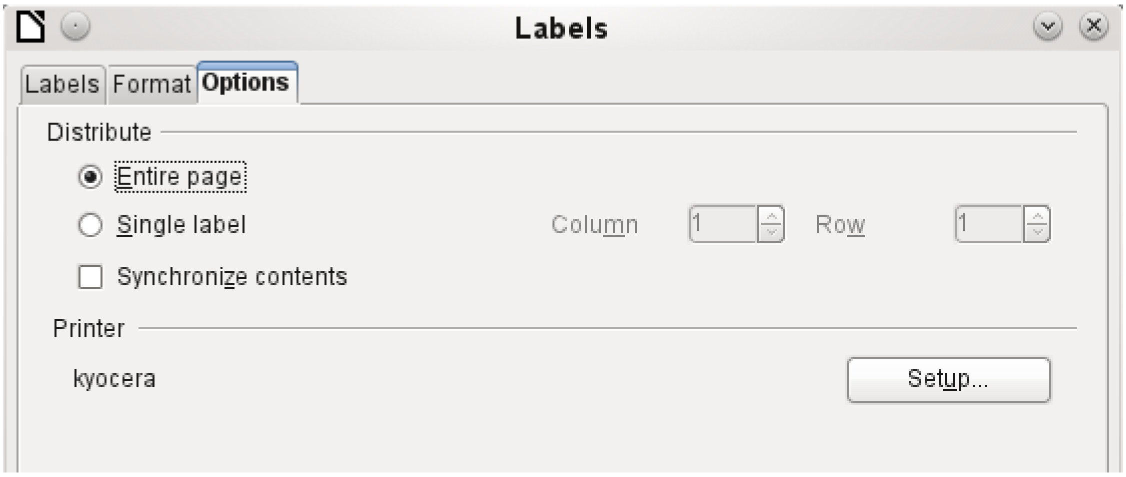
Rows 7

Page Width 21,00cm

Page Height 29,70cm

Avery A4 C2160 Avery Inkjet Label

Figure 401: Labels dialog, Format tab



Labels

Labels | Format | Options

Distribute

● Entire page

○ Single label Column 1 Row 1

□ Synchronize contents

Printer

kyocera

Figure 402: Labels dialog, Options tab

On the Options tab you can specify whether only a single label or a whole page of labels will be produced. The page will then be filled with data from successive records of the database, beginning with the first record. If there are more records than will fit on the page, the next page will automatically be filled with the next set of records.

The Synchronize contents checkbox links all the labels together so that subsequent changes in layout of any label will be applied to all the others. To transfer the edited content, use the Synchronize button, which appears during label production if you have selected this checkbox.

Use the **New Document** button to create a document containing the selected fields.

When you initiate the printing process, the confirmation dialog shown in Figure 398 appears (as for form letters). Choose **Yes** to fill the address database fields with the corresponding content.

The source of the data for the label printing is not found automatically; only the database is pre-selected. The actual query must be specified by the user, because in this case we are not dealing with a table.

When the query is selected and the corresponding records chosen (in this case *All*), the printing can begin. It is advisable, especially for the first tests, to choose *Output to a File*, on the Mail Merge dialog (Figure 399), which will save the labels as a document. The option to save in several documents is not appropriate for label printing but rather for letters to different recipients which can then be worked on subsequently.

Direct creation of mail merge and label documents

Instead of using the Wizard, you can produce mail merge and label documents directly.

Mail merge using the mouse

Mail merge fields can be taken from the database browser using the mouse.

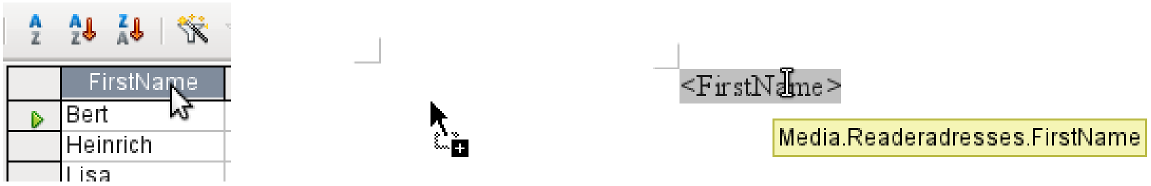


Figure 403: Using the mouse to create a mail merge

Select the table header with the left mouse button. Hold the button down and drag the cursor through the text document. The cursor changes its shape to an insert symbol. The MailMerge field is inserted into the text document, here shown in the complete description which is made visible using **View > Field names**.

Creating form letters by selecting fields

Mail merge fields can be inserted using **Insert > Field > More Fields > Database**.

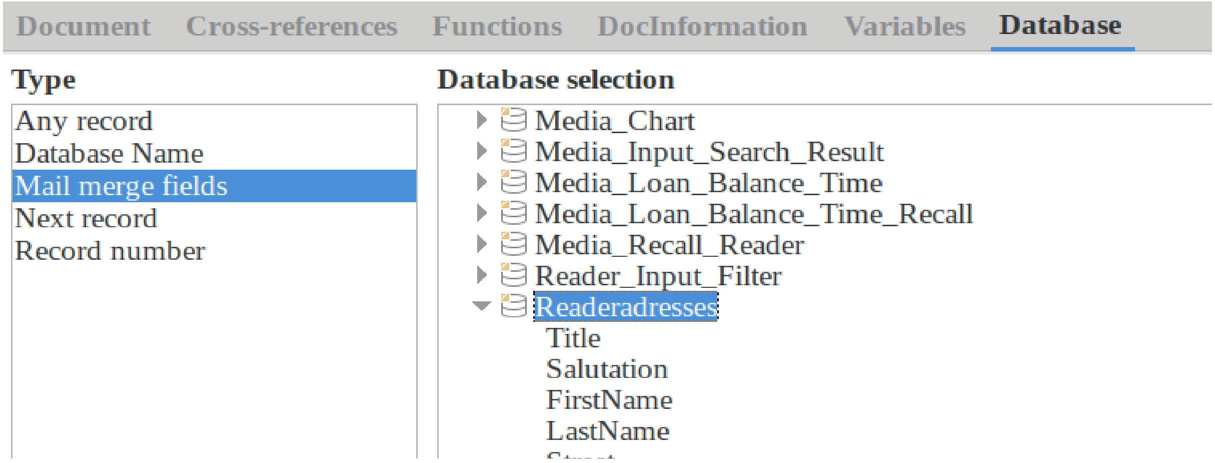


Figure 404: Using the Fields dialog, Database tab to insert mail merge fields

Here all tables and queries in the selected database are available. Using the **Insert** button, you can insert the various fields one after another directly into the text at the current cursor position.

If you want to create a salutation, which is usual in form letters, you can use a hidden paragraph or hidden text: **Insert > Fields > More Fields > Functions > Hidden paragraph**. For both variants take care that the condition you formulate will not be fulfilled, since you want the paragraph to be visible.

For the formula Dear Ms <Surname>, to appear only when the person is female, a sufficient condition is:

```
[Media.Readeraddresses.Salutation] ! = "Mrs."
```

Now the only remaining problem is that there may be no surname. Under these circumstances, "Dear Sir/Madam," should appear so this is the condition you must insert. The overall condition is:

```
[Media.Readeraddresses.Salutation] ! "Mrs." OR NOT
[Media.Readeraddresses.Salutation]
```

That excludes the possibility of this paragraph appearing when the person is not female or there is no entered surname.

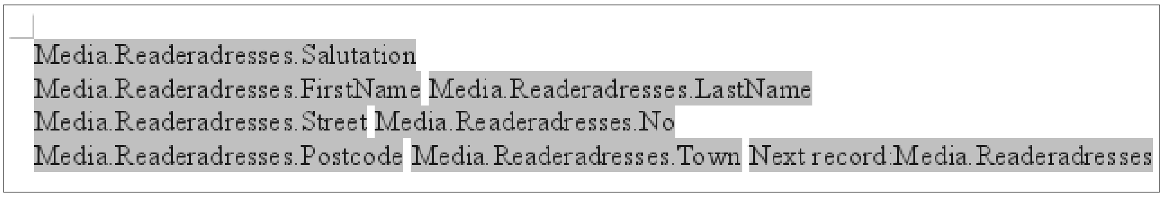
In the same way you can create entries for the masculine gender and for missing entries for the two remaining types of salutation.

Naturally you can create a salutation in the address field in exactly the same way, wherever the gender is specified.

Further information is given in the LibreOffice Help under Hiding Text and *Conditional Text*.

Of course it would be still simpler if someone who understands databases were to put the whole salutation right into the query. This can be done using a correlated subquery (see Chapter 5, Queries).

Particularly interesting for labels is the field type **Next record**. If this field type is chosen at the end of a label, the next label will be filled with data from the following record. Typical labels for sequential label printing look like the following figure when you use **View > Field Names** to make the corresponding field designations visible:



Media.Readeradresses.Salutation
Media.Readeradresses.FirstName Media.Readeradresses.LastName
Media.Readeradresses.Street Media.Readeradresses.No
Media.Readeradresses.Postcode Media.Readeradresses.Town Next record:Media.Readeradresses

Figure 405: Typical field selection for sequential label printing

For the last label on the page, you must allow for the fact that the next record is automatically called up following a page break. Here the field type *Next record* should not occur. Otherwise a record will be missed out because a double record jump occurs.

Tip

Creating mail merge letters is also possible directly from a database form. The only requirement is that the database be registered in LibreOffice.

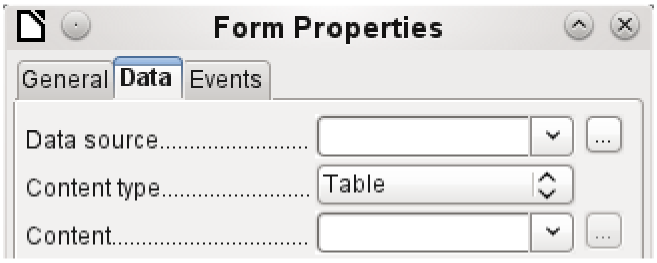
When a mail merge is carried out, be sure to choose **View > Normal**. This ensures that the elements are correctly positioned on the page. If a form is then printed, the usual mail merge query appears.

This type of mail merge has the advantage that you do not need any files other than the *.odb file in order to print.

External forms

If simple form properties available in LibreOffice are to be used in other components such as Writer and Calc, you only need to display the Form Design toolbar, using **View > Toolbars >**

Form design, then open the Form Navigator. You can build a form or, as described in Chapter 4, Forms, create a form field. The Data tab of the Form Properties dialog looks a little different from the one you see when forms are built directly in an ODB database file.

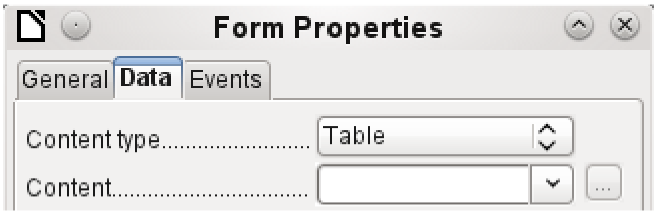


Data source

Content type: Table

Content

Figure 406: Upper part of Form Properties dialog, Data tab with an external data source



Content type: Table

Content

Figure 407: Upper part of Form Properties dialog, Data tab with an internal data source.

The Data source must be selected separately when using an external form. Use the button to the right of the data source listbox to open the file browser. Any ODB file can be selected. In addition, the field for the data source contains a link, beginning with `file:///`.

If instead you look in the listbox contents, you will see databases already registered in LibreOffice under their registered names.

The forms are created in exactly the same way as in Base itself.

The forms produced in this way are by default shown in edit mode each time the file is opened, not write-protected as in Base. To prevent accidental modification of the form, you can use **File > Properties > Security** to open the file read-only. You can even protect the file from alteration using a password. In office systems, it is also possible to declare the whole file as write-protected. This still allows entry into the fields of the form, but not movement of fields or the entering of text between them.

Tip

Forms can also be rapidly created by using drag and drop. To do this, open the database, find the relevant table or query, and select the table headers.

In Writer, select appropriate field headings with the left mouse button, hold down the Shift and Ctrl keys, and the mouse cursor will turn into a link symbol. Then drag the headings into the Writer document.

You can drag the fields into Calc files without the use of the additional keys. The copy symbol appears as a mouse cursor.

In both cases an entry field is created with the associated label. The link to the data source is created with the first actual entry of data, So data entry into such a form can commence immediately after the drag-and-drop operation.

Advantages of external forms

Base need not be opened first in order to work with the database. Therefore you do not need an extra open window in the background.

In a database that is already complete, existing database users can subsequently be sent the improved form without problems. They can continue to use the database during the development of further forms, and do not need to copy complicated external forms from one database into another.

Forms for a database can be varied to suit the user. Users who do not have the authority to correct data or make new entries can be sent a current data set by other users, and simply replace their *.odb file to have an up-to-date view. This could, for example, be useful for a database for an organization where all committee members get the database but only one person can edit the data; the others can still view the addresses for their respective departments.

Disadvantages of external forms

Users must always install forms and Base with the same directory structure. That is the only way that access to the database can be free from errors. As the links are stored relative to the form, it is sufficient to store the database and its forms in a common directory.

Only forms can be created externally, not queries or reports. A simple glance at a query therefore must go through a form. A report on the other hand requires the opening of the database. Alternatively it might be possible to create it, at least partially, using mail merge.

Database use in Calc

Data can be used in Calc for calculation purposes. For this purpose it is first necessary to make the data accessible in a Calc worksheet.

Entering data into Calc

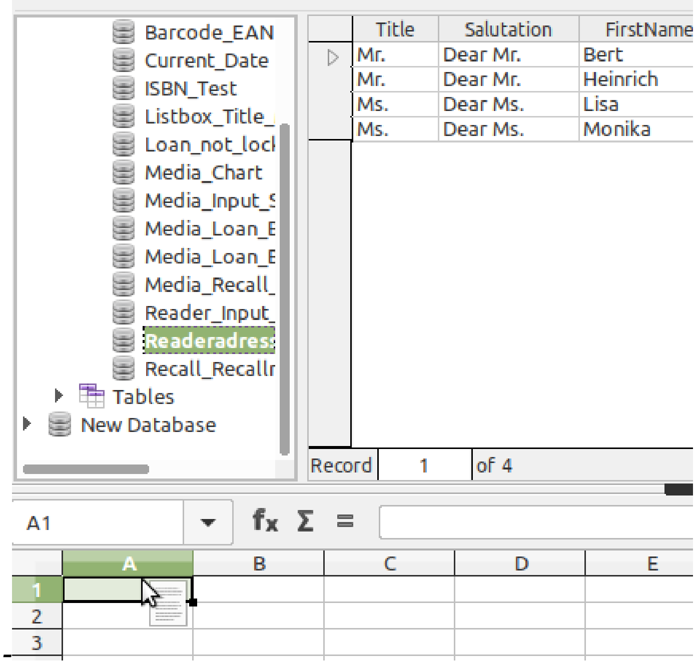
There are various ways of entering data into Calc.

Select a table with the left mouse button and drag it into a Calc worksheet. The cursor sets the left upper corner of the table. The table is created complete with field names. The data source browser in this case does not offer the options of Data to Text or Data to Fields.

Data dragged into Calc in this way shows the following properties:

Not only the data are imported, but the field properties too are read and acted on during the import. Fields such as house numbers, which were declared as text fields, are formatted as text after insertion into Calc.

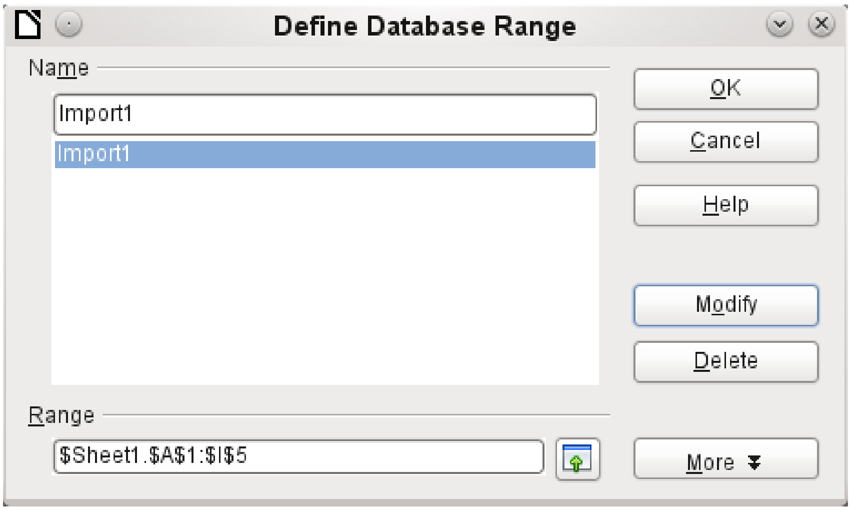
The import becomes a Calc range, which by default is assigned the name Import1. The data can later be accessed using this range. **Data > Refresh range** allows the range, where appropriate, to be supplied with new data from the database.



Title	Salutation	FirstName
Mr.	Dear Mr.	Bert
Mr.	Dear Mr.	Heinrich
Ms.	Dear Ms.	Lisa
Ms.	Dear Ms.	Monika

Record 1 of 4

Figure 408: Dragging a data source table to a Calc sheet



Name: Import1

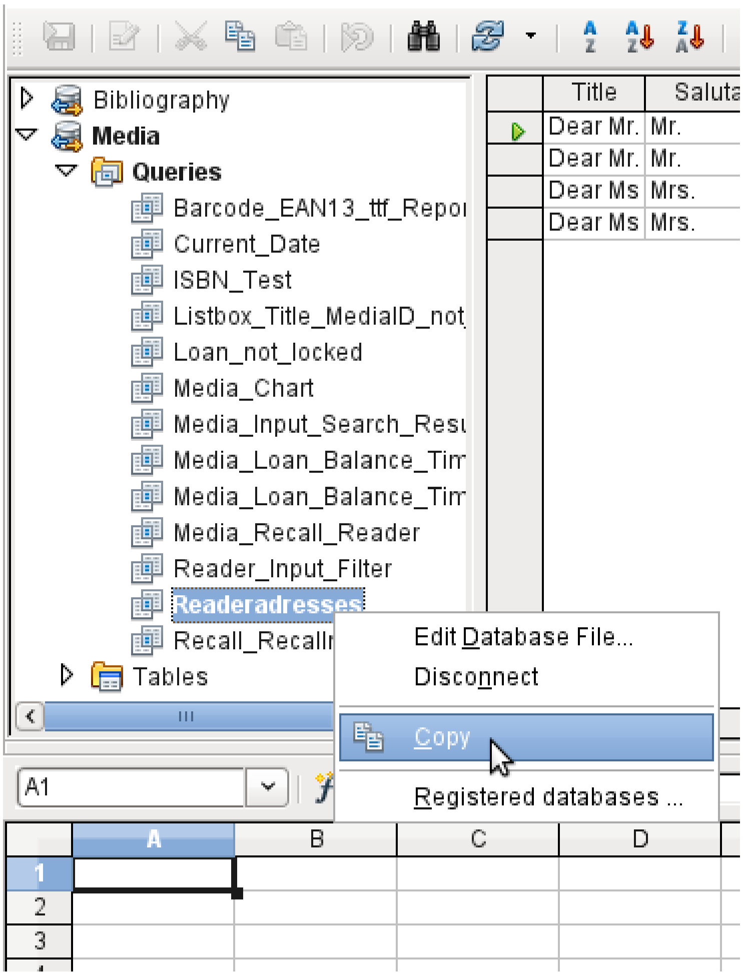
Import1

Range: $Sheet1.$A$1:$I$5

Figure 409: Define Database Range dialog showing the automatically create Import1 database range

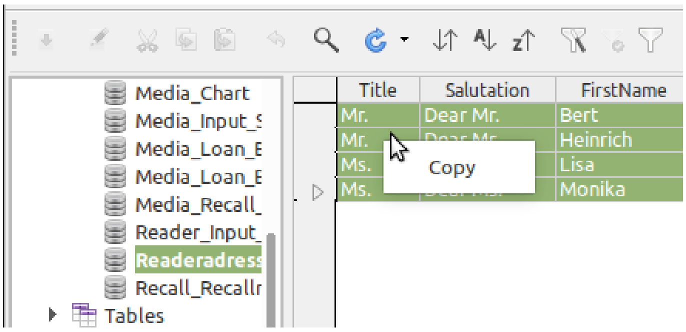
The imported data is not formatted except as the properties of the database fields require.

You can also use the context menu of a table to make a *copy* of the data. In this case, however, there is no import but merely a copy. The properties of the data fields are not read with them but are determined by Calc. In addition the field names are formatted as table headers.



Title	Saluta
Dear Mr.	Mr.
Dear Mr.	Mr.
Dear Ms	Mrs.
Dear Ms	Mrs.

Figure 410: Copying table from left side of data source browser

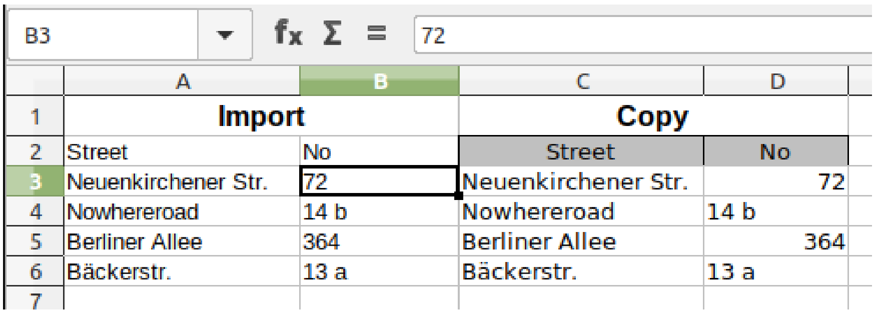


Title	Salutation	FirstName
Mr.	Dear Mr.	Bert
Mr.	Dear Mr	Heinrich
Ms.		Lisa
Ms.	Dear Ms.	Monika

Figure 411: Copying table from right side of data source browser

You see the difference especially in database fields that are formatted as text. On import, Calc turns these into text fields, which are aligned to the left like other text. These numbers can then no longer be used in calculations.

If you export them again, the data remain as they were.



B3 | 72

	A	B	C	D
1	Import		Copy	
2	Street	No	Street	No
3	Neuenkirchener Str.	72	Neuenkirchener Str.	72
4	Nowhereroad	14 b	Nowhereroad	14 b
5	Berliner Allee	364	Berliner Allee	364
6	Bäckerstr.	13 a	Bäckerstr.	13 a
7				

Figure 412: Comparison of imported and copied data in Calc

Tip

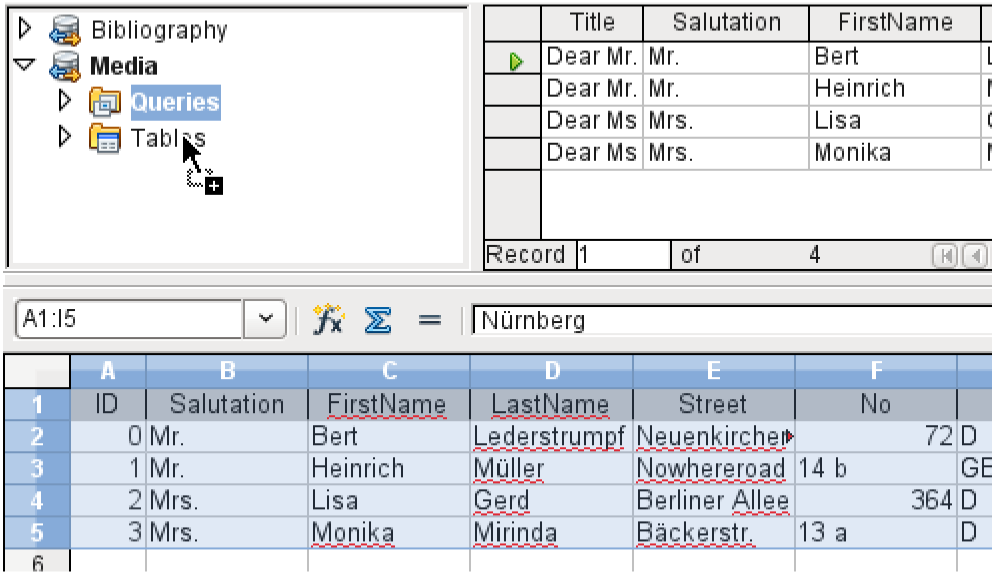
Importing data into Calc overwrites the previous contents and also any previous formatting. If data is to be exported consistently into the same table, you should use a separate sheet for data import. The data are then read into the other sheet by using the term `tablename.fieldname`. The fields in this sheet can be formatted suitably without risk of the formatting being overwritten.

Tip

Records can also be copied directly from the database using the clipboard, or by drag and drop using the mouse. If a table or query is dragged into a Calc spreadsheet, the whole of the content is inserted. If the table or query is opened and one or more records selected, only these records together with the field names are copied when you drag them over.

Exporting data from Calc into a database

Select the data in the Calc worksheet. Hold the left mouse button down and drag the data that you want to turn into a database into the table area of the database browser.



Title	Salutation	FirstName
Dear Mr.	Mr.	Bert
Dear Mr.	Mr.	Heinrich
Dear Ms	Mrs.	Lisa
Dear Ms	Mrs.	Monika

Record 1 of 4

A1:I5 | Nürnberg

	A	B	C	D	E	F	
1	ID	Salutation	FirstName	LastName	Street	No	
2	0	Mr.	Bert	Lederstrumpf	Neuenkirchen	72	D
3	1	Mr.	Heinrich	Müller	Nowhereroad	14 b	GE
4	2	Mrs.	Lisa	Gerd	Berliner Allee	364	D
5	3	Mrs.	Monika	Mirinda	Bäckerstr.	13 a	D
6							

Figure 413: Copying data from Calc into a database table

The cursor changes its appearance, showing that something can be inserted.

The first window of the Import Wizard opens. The further steps with the wizard are described in Chapter 3, Tables, in the section “Importing data from other sources”.

Converting data from one database to another

In the explorer of the data source browser, tables can be copied from one database to another by selecting the source table with the left mouse button, holding the button down, and dragging it into the target database in the table container. This causes the dialog for copying tables to be displayed.

In this way, for example, read-only databases (data sources such as address books from an email program or a spreadsheet table) can be used as a basis for a database in which the data become editable. Also data can be directly copied when changing to another database program (for example changing from PostgreSQL to MySQL).

If you wish the new database to have different relationships from the original one, you can arrange this by using appropriate queries. Those who are not sufficiently expert can instead use Calc. Just drag the data into a spreadsheet and prepare them for import into the target database using the facilities that Calc provides.

For the cleanest possible import into a new database, the tables should be prepared in advance. This allows problems of formatting and those involving the creation of primary keys to be recognized well in advance.

Importing records into a table using the clipboard

If records are available in tabular form, they can be inserted into Base using the clipboard and the wizard.

In Base, a right-click on the destination table begins the import. In the context menu under **Copy** are the commands **Import** and **Import content**. If you choose **Paste**, the Import Wizard will have already selected the table and *Append data*. **Paste special** gives only a query for an import filter. The available options are HTML and RTF.

If instead you right-click in the table container, the Import Wizard gives you only the choice of creating a new table.

Importing PDF records

If you want to import data from various external sources, it is best to choose a format that prevents your form from being modified during data entry. Using Writer, you can create forms in PDF format, put them online, and have the completed forms returned to you, for example as email attachments. All that is lacking is the simplest possible entry of the data into Base. The example illustrates such a means of import.

Note

The database Example_PDFFormular_Import.odb for this report is included in the example databases for this book.

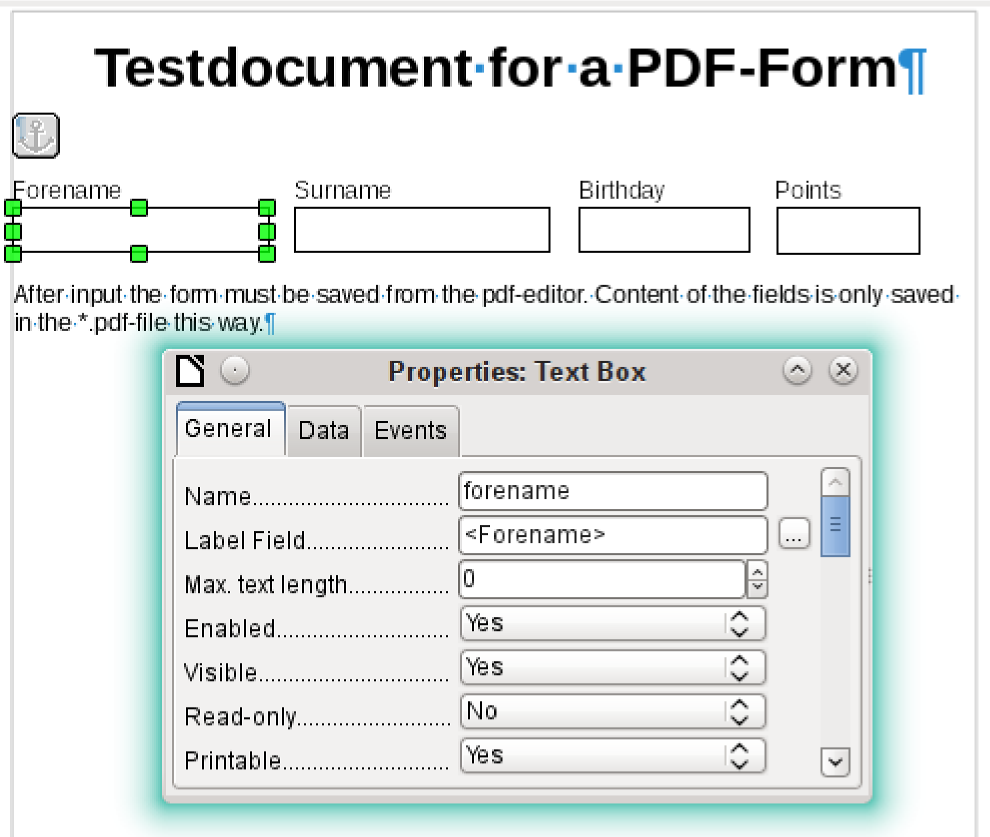
Creating a PDF form

A PDF form is created as an external form with no link to the database. Using **View > Toolbars > Form Controls**, the necessary elements for the form are displayed and can be inserted as required.

Unfortunately PDF format makes no distinction between numeric fields, date fields, and text fields. For the example provided here, it is sufficient to use text fields for all the entries. Other field formats within the Writer form will inevitably be lost during PDF export.

Basically PDF forms can have the following fields:

- Buttons
- Text fields
- Checkboxes
- Comboboxes
- Listboxes



Testdocument for a PDF-Form

After input the form must be saved from the pdf-editor. Content of the fields is only saved in the *.pdf-file this way.

Name	forename
Label Field	<Forename>
Max. text length	0
Enabled	Yes
Visible	Yes
Read-only	No
Printable	Yes

Figure 414: PDF test document to be read into Base

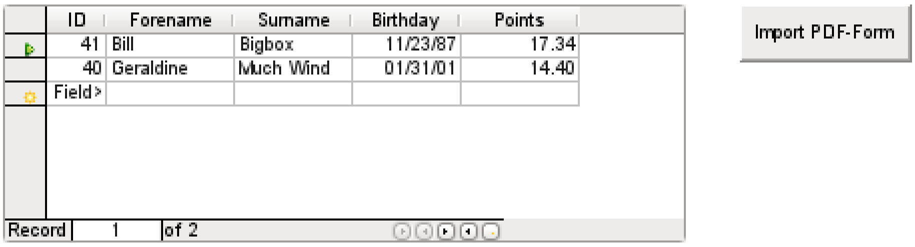
The test form contains a total of 4 text fields. In **Properties: Text box > General > Name**, you should always choose the field name used in the database table when using the following import method in order to avoid problems with field names and field content.

Help messages are shown when the records are read, but do not appear in every PDF viewer.

To ensure that the form actually contains the records, it should be saved in the PDF viewer after data input, using the **File > Save as** menu option. The actual command for doing this might vary between viewers. Without this procedure, the viewer will show the records after the form has been opened on your own computer, but it actually reads them from the viewer’s temporary storage file and not directly from the PDF file. If the form is then transferred to another computer, it will be empty.

Reading the records from the PDF form

The form for the Base database is very simple in appearance. It is linked to the table and shows the records that have just been read in. The most recent entries are shown in the table control above.



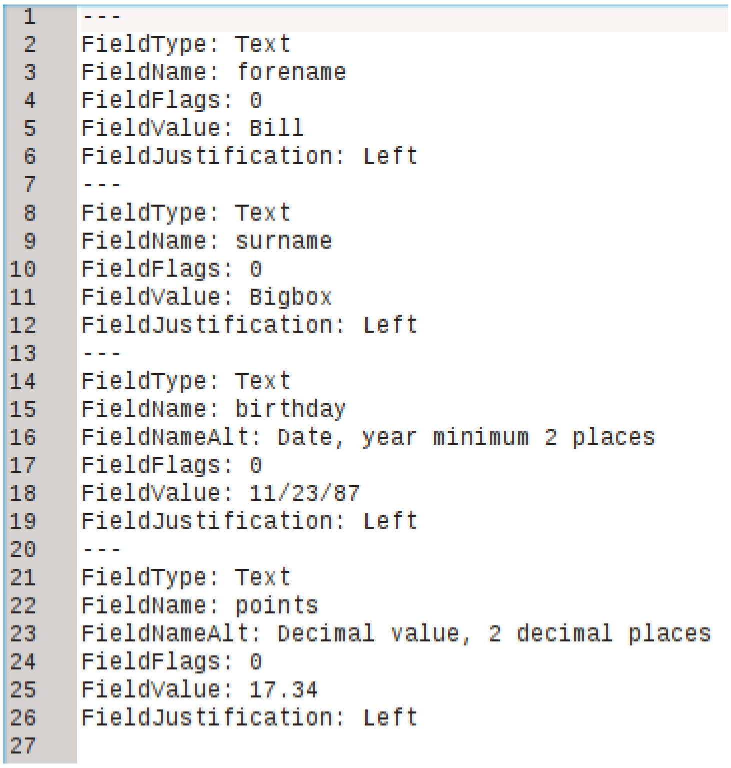
ID	Forename	Surname	Birthday	Points
41	Bill	Bigbox	11/23/87	17.34
40	Geraldine	Much Wind	01/31/01	14.40
Field>				

Figure 415: Form data read into Base from PDF document

The macro for reading the records in is entered under **Properties: Button > Events > Execute Action**.

To read the records out, we use the open source program `pdftk`. The program is freely available for both Windows and Linux. Linux distributions mostly have it as a package in their repositories. Windows users will find it at https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/

The records read out using `pdftk` are written into a text file, which looks like this:



```
---
FieldType: Text
FieldName: forename
FieldFlags: 0
FieldValue: Bill
FieldJustification: Left
---
FieldType: Text
FieldName: surname
FieldFlags: 0
FieldValue: Bigbox
FieldJustification: Left
---
FieldType: Text
FieldName: birthday
FieldNameAlt: Date, year minimum 2 places
FieldFlags: 0
FieldValue: 11/23/87
FieldJustification: Left
---
FieldType: Text
FieldName: points
FieldNameAlt: Decimal value, 2 decimal places
FieldFlags: 0
FieldValue: 17.34
FieldJustification: Left

```

Each field is represented by five to six lines in the file. For the macro, the important lines are **FieldName** (should be the same as FieldName in the destination table), **FieldValue** (content of the field after saving the PDF file), and **FieldJustification** (last line of the entry).

The whole import process is controlled by macros. The PDF form needs to be on the same path as the database. The records are read out of it into the text file, and then read into the database from this text file. This continues for all the PDF files in the folder. Old records should therefore be removed from the folder as far as possible, because the function does not check for duplication.

```
SUB PDF_Form_Import(oEvent AS OBJECT)
   DIM inNumber AS INTEGER
   DIM stRow AS STRING
   DIM i AS INTEGER
   DIM k AS INTEGER
   DIM oDatasource AS OBJECT
   DIM oConnection AS OBJECT
   DIM oSQL_Command AS OBJECT
   DIM oResult AS OBJECT
   DIM stSql AS STRING
   DIM oDB AS OBJECT
   DIM oFileAccess AS OBJECT
   DIM inFields AS INTEGER
   DIM stFieldName AS STRING
   DIM stFieldValue AS STRING
   DIM stFieldType AS STRING
   DIM stDir AS STRING
   DIM stDir2 AS STRING
   DIM stPDFForm AS STRING
   DIM stFile AS STRING
   DIM stTable AS STRING
   DIM inNull AS INTEGER
   DIM aFiles()
   DIM aNull()
   DIM stCommand AS STRING
   DIM stParameter AS STRING
   DIM oShell AS OBJECT
```

After the variables have been declared, the number of fields in the PDF form is given. The count begins at 0, so a value of 3 actually means a total of four fields. Using this count, it can be determined if all the data for a record has been read, so that it is ready to be transferred into the table.

```
inFields = 3
stTable = "Name"
oDatasource = ThisComponent.Parent.CurrentController
If NOT (oDatasource.isConnected()) THEN
   oDatasource.connect()
END IF
oConnection = oDatasource.ActiveConnection()
oSQL_Command = oConnection.createStatement()
```

The database connection is made. The path to the database file in the file system is read. Using this path, the contents of the folder are read into the array aFiles. A loop checks each filename in the array to see if it ends in .pdf. Upper and lower case are not distinguished, as the results of the search are all converted into lower case using Lcase.

```
oDB = ThisComponent.Parent
stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
aFiles = oFileAccess.getFolderContents(stDir, False)
FOR k = 0 TO uBound(aFiles())
   IF LCase(Right(aFiles(k),4)) = ".pdf" THEN
      stDir2 = ConvertFromUrl(stDir)
      stPDFForm = ConvertFromUrl(aFiles(k))
```

To determine the command for reading out the data, it is necessary to understand the operating system’s file address conventions. Therefore the original URL beginning with file:// must be adapted to the current system. The command for starting the pdftk program depends on the operating system. It might carry the suffix .exe or perhaps a complete path to the program like *C:\ Program Files (x86)\pdftk\pdftk.exe* or the suffix might not be required at all. `GetGuiType` is used to determine the type of system in use: 1 stands for Windows, 3 for macOS, and 4 for Linux. The following steps only distinguish between Windows and the rest.

After this the Shell() function is used to pass the appropriate launch command for pdftk to the console. The argument True ensures that LibreOffice will wait until the shell process terminates.

```
IF GetGuiType = 1 THEN '()
   stCommand = "pdftk.exe"
ELSE
   stCommand = "pdftk"
END IF
stParameter = stPDFForm & " dump_data_fields_utf8 output "
   & stDir2 & "PDF_Form_Data.txt"
Shell(stCommand,0,stParameter,True)
stFile = stDir & "PDF_Form_Data.txt"
i = -1
inNumber = FreeFile
```

The FreeFile function determines which is the next free data channel available in the operating system. This channel is read out as an integer number and used to connect directly to the PDF data file that has just been created. The `INPUT` instruction is used to read the file. This takes place outside LibreOffice. The external records are then read into LibreOffice.

```
OPEN stFile FOR INPUT AS inNumber
   DO WHILE NOT Eof(inNumber)
      LINE INPUT #inNumber, stRow
```

The PDF data file is now read line by line. Whenever the term **FieldName** occurs, the remaining content of the line is taken as the name of the field in the PDF form and also, because of the way the form was defined, the name of the database field into which the data should be written.

All the field names are combined directly for use in the later SQL commands. What this means in practice is that the field names are enclosed in double quotes and separated by commas.

In addition, for each field name a query determines the field type in the table. Date and decimal values must be transferred in a different way to text.

```
             IF instr(stRow, "FieldName: ") THEN
                IF stFieldName = "" THEN
                   stFieldName = """"+mid(stRow,12)+""""
                ELSE
                   stFieldName = stFieldName & ","""" +
mid(stRow,12)+""""
                END IF
                stSql = "SELECT TYPE_NAME FROM
INFORMATION_SCHEMA.SYSTEM_COLUMNS
                     WHERE TABLE_NAME = '" + stTable + "' AND
                     COLUMN_NAME = '" + mid(stRow,12) + "'"
                oResult = oSQL_Command.executeQuery(stSql)
                WHILE oResult.next
                   stFieldType = oResult.getString(1)
                WEND
             END IF
```

As for the field names, so for the field values. However these must not be double-quoted but must be prepared in accordance with the requirements of SQL code. This means that text must be single-quoted, dates converted to conform with SQL conventions, and so on. This is done by the extra external SQL_Value function.

```
IF instr(stRow, "FieldValue: ") THEN
   IF stFieldValue = "" THEN
      stFieldValue = SQL_Value(mid(stRow,13), stFieldType)
   ELSE
```

```
                        stFieldValue = stFieldValue & "," &
                           SQL_Value(mid(stRow,13), stFieldType)
                     END IF
                  END IF
```

If the term **FieldJustification** is found, this marks the end of the combined block of field name and properties. The counter i, which will subsequently be compared with the previously declared field counter inFields, is therefore incremented by 1.

When i and inFields become equal, the SQL command can be put together. However you must ensure that empty records are not created from empty forms. Therefore there is a previous check for all field values being NULL. In such cases, the SQL command is launched immediately. Otherwise, the record is included for insertion into the Name table. After this, the variables are restored to their default values and the next PDF form can be read.

```
                  IF instr(stRow, "FieldJustification:") THEN
                     i = i + 1
                  END IF
                  IF i = inFields THEN
                     aNull = Split(stFieldValue,",")
                     FOR n = 0 TO Ubound(aNull())
                        IF aNull(n) = "NULL" THEN inNull = inNull + 1
                     NEXT
                     IF inNull < inFields THEN
                        stSql ="INSERT INTO """ + stTable + """(" + stFieldName
   + ")"

                        stSql = stSql + "VALUES (" + stFieldValue + ")"
                        oSQL_Command.executeUpdate(stSql)
                     END IF
                     stFieldName = ""
                     stFieldValue = ""
                     stFieldType = ""
                     i = -1
                     inNull = 0
                  END IF
               LOOP
            CLOSE inNumber
```

At the end of the procedure, one *PDF_Form_Data.txt* file remains. This is deleted. Then the form is reloaded so that the records that were read in can be displayed.

```
            Kill(stFile)
         END IF
      NEXT
      oEvent.Source.Model.Parent.reload()
   END SUB
```

If the text contains a `"'"`, this will be seen as an end-of-text marker during insertion by SQL. The SQL code for the insertion command fails if any more text follows without being enclosed in single quotes. To prevent this, each single quote within the text must be masked by another single quote. This is the job of the String_to_SQL function.

```
   FUNCTION String_to_SQL(st AS STRING) AS STRING
      IF InStr(st,"'") THEN
         st = Join(Split(st,"'"),"''")
      END IF
      String_to_SQL = st
   END FUNCTION
```

Dates in the PDF file are read as text. They cannot be checked in advance for correct entry.

When dates are written in English, the day, month and year are separated by dots or, more often, hyphens. The day and the month may have a single digit or two. The year may have two digits or four.

In SQL code, dates must begin with a four-digit year and be written YYYY-MM-DD. The entered dates therefore need to go through a conversion process.

The entered date is split up into day, month, and year parts. The day and month are given a leading zero and then right-truncated to two digits. This ensures a two-digit figure in all cases.

If the year part already has four digits (greater than 1000), the value is not changed. Otherwise, if the year is greater than 30, the date is assumed to belong to the last century and needs to have 1900 added. All other dates are assigned to the current century.

```
FUNCTION Date_to_SQLDate(st AS STRING) AS STRING
   DIM stDay AS STRING
   DIM stMonth AS STRING
   DIM stDate AS STRING
   DIM inYear AS INTEGER
   stDay = Right("0" & Day(CDate(st)), 2)
   stMonth = Right("0" & Month(CDate(st)), 2)
   inYear = Year(CDate(st))
   IF inYear = 0 THEN
      inYear = Year(Now())
   END IF
   IF inYear > 1000 THEN
   ELSEIF inYear > 30 THEN
      inYear = 1900 + inYear
   ELSE
      inYear = 2000 + inYear
   END IF
   stDate = inYear & "-" & stMonth & "-" & stDay
   Date_to_SQLDate = stDate
END FUNCTION
```

The SQL_Value function combines this function with the NULL settings shown below, and so gives correctly formatted values for input into the database to its calling function.

Empty fields yield a NULL value. The corresponding field in the table will also be empty.

```
FUNCTION SQL_Value(st AS STRING, stType AS STRING) AS STRING
   DIM stValue AS STRING
   IF st = "" THEN
      SQL_Value = "NULL"
```

If this is a date field, and the content is to be recognizable as a date, its content must be converted into the SQL date format. If it is not recognizable as a date, the field should remain empty.

```
   ELSEIF stType = "DATE" THEN
      IF isDate(st) THEN
         SQL_Value = "'" & Date_to_SQLDate(st) & "'"
      ELSE
         SQL_Value = "NULL"
      END IF
```

A decimal field might contain commas instead of decimal points, with decimal places following them. In Basic and SQL, the decimal separator is always a dot. Therefore numbers containing a comma must be converted. The field must contain a number, so other characters such as units must be removed. This is carried out by the Val() function.

```
   ELSEIF stType = "DECIMAL" THEN
```

```
        stValue = Str(Val(Join(Split(st,","),".")))
```

All other content is treated as text. Single quotes are masked with a further single quote and the whole term is enclosed again in single quotes.

```
      ELSE
        SQL_Value = "'" & String_to_SQL(st) & "'"
      END IF
   END FUNCTION
```

For further details on macro construction, see Chapter 9, Macros. This example simply shows that it is possible to transfer data from PDF forms into Base without having to copy the values field by field using the clipboard. The construction of the above procedure has deliberately been kept very general and would need to be adapted to particular situations.

Base Guide 7.3

Chapter 8
Database Tasks

General remarks on database tasks

This chapter describes some solutions for problems that arise for many database users.

Data filtering

Data filtering using the GUI is described in Chapter 3, Tables. Here we describe a solution to a problem that many users have raised: how to use listboxes to search for the content of fields in tables, which then appear filtered in the underlying form section and can be edited.

The basis for this filtering is an editable query (see Chapter 5, Queries) and an additional table, in which the data to be filtered are stored. The query shows from its underlying table only the records that correspond to the filter values. If no filter value is given, the query shows all records.

The following example starts from a ‘*Media’ example* table that includes, among others, the following fields: ID (primary key), Title, Category. The field types are INTEGER, VARCHAR, and VARCHAR respectively.

First we require a ‘Filter’ example table. This table contains a primary key and two filter fields (of course you can have more if you want): ID (primary key), Filter_1, Filter_2. As the fields of the ‘Media’ example table, which is to be filtered, are of the type VARCHAR, the fields Filter_1 and Filter_2 are also of this type. ID can be the smallest numeric type, TINYINT because the Filter table will never contain more than one record.

You can also filter fields that occur in the ‘Media’ example table only as foreign keys. In that case, you must give the corresponding fields in the ‘Filter’ example table the type appropriate for the foreign keys, usually INTEGER.

The following query is certainly editable:

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

All records from the Media example table are displayed, including the primary key.

```
SELECT * FROM "Media"
WHERE "Title" = IFNULL( ( SELECT "Filter_1" FROM "Filter" ), 'Title' )
```

If the field Filter is not NULL, those records are displayed for which the Title is the same as Filter_1. If the field Filter is NULL, the value of the Title field is used instead. As Title is the same as "Title", all records are displayed. This assumption does not hold, however, if the Title field of any record is empty (contains NULL). That means that those records will never be displayed that have no title entry. Therefore we need to improve the query:

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

WHERE "T" = IFNULL( ( SELECT "Filter_1" FROM "Filter" ), 'T' )
```

Tip

IFNULL(expression, ‘value’) requires the expression has the same field type as the value.

If the expression has the field type VARCHAR, use two single quotes as the value.

If it has DATE as its field type, enter a date as the value that is not contained in the field of the table to be filtered. Use this format: {D 'YYYY-MM-DD'}.

If it is any of the numerical field types, use the NUMERIC field type for the value. Enter a number that does not appear in the field of the table to be filtered.

This variant will lead to the desired goal. Instead of filtering Title directly, a field is filtered which carries the alias T. This field has no content either but it is not NULL. In the conditions only the field T is considered. All records are therefore displayed even if Title is NULL.

This command is available only directly with SQL and you cannot do this using the GUI. To make it editable in the GUI, further modification is required:

```
SELECT "Media".* , IFNULL( "Media"."Title", '' ) AS "T"
FROM "Media"
WHERE "T" = IFNULL( ( SELECT "Filter_1" FROM "Filter" ), 'T' )
```

If the relationship of the table to the fields is now set up, the query becomes editable in the GUI. As a test, you can put a title into "Filter"."Filter_1".

As "Filter"."ID" sets the value '0', the record is saved and the filtering can be comprehended. If "Filter"."Filter_1" is emptied, the GUI treats that as NULL. A new test yields a display of all the media. In any case, before a form is created and tested, just one record with a primary key should be entered into the Filter table. It must be only one record, since sub-queries as shown above can only transmit one value.

The query can now be enlarged to filter two fields:

```
SELECT "Media".* , IFNULL( "Media"."Title", '' ) AS "T", IFNULL(
"Media"."Category", '' ) AS "K"
FROM "Media"
WHERE "T" = IFNULL( ( SELECT "Filter_1" FROM "Filter" ), "T" ) AND "K" =
IFNULL( ( SELECT "Filter_2" FROM "Filter" ), "K" )
```

This concludes the creation of the editable query. Now for the basic query for the two listboxes:

```
SELECT DISTINCT "Title", "Title"
FROM "MediaExample" ORDER BY "Title" ASC
```

The listbox should show the Title and then also transmit that Title to the Filter_1 field in the Filter table that underlies the form. Also no duplicate values should be shown (DISTINCT condition). And the whole thing should of course be sorted into the correct order.

A corresponding query is then created for the Category field, which is to write its data into the Filter_2 field in the Filter table.

If one of these fields contains a foreign key, the query is adapted so that the foreign key is passed to the underlying Filter table.

The form consists of two parts. Form 1 is the form based on the Filter table. Form 2 is the form based on the query. Form 1 has no navigation bar and the cycle is set to Current record. In addition, the *Allow additions* property is set to *No*. The first and only record for this form already exists.

Form 1 contains two listboxes with appropriate labels. Listbox 1 returns values for Filter_1 and is linked to the query for the Title field. Listbox 2 returns values for Filter_2 and relates to the query for the Category field.

Form 2 contains a table control field, in which all fields from the query can be listed except for the fields T and K. The form would still work if these fields were present; they are omitted to avoid a confusing duplication of field contents. In addition Form 2 contains a button, linked to the *Update form* function. An additional navigation bar can be built in to prevent screen flicker every time the form changes, due to the navigation bar being present in one form and not in the other.

Once the form is finished, the test phase begins. When a listbox is changed, the button on Form 2 is used to store this value and update Form 2. This now relates to the value which the listbox provides. The filtering can be made retrospective by choosing an empty field in the listbox.

Searching for data

The main difference between searching for data and filtering data is in the query technique. The aim is to deliver, in response to free language search terms, a resulting list of records that may only partially contain these actual terms. First the similar approaches to the table and form are described.

Searching with LIKE

The table for the search content may be the same one that already contains the filter values. The Filter table is simply expanded to include a field named Searchterm. So, if required, the same table can be accessed and, using the forms, simultaneously filtered and searched. Searchterm has the field type VARCHAR.

The form is built just as for filtering. Instead of a listbox, we need a text entry field for the search term, and also perhaps a label field with the title Search. The field for the search term can stand alone in the form or together with the fields for filtering, if both functions are desired.

The difference between filtering and searching lies in the query technique. While filtering uses a term that already occurs in the underlying table, searching uses arbitrary entries. (After all, the listbox is constructed from the table content.)

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

This query normally leads to an empty result list for these reasons:

- When entering search terms, people seldom know completely and accurately what the title is. Therefore the correct title does not get displayed. To find the book "The Hitchhiker’s Guide to the Galaxy" it should be sufficient to put "Hitchhiker" into the Search field or even just "Hit".
- If the Searchterm field is empty, only records are displayed in which there is no title. This only happens if the item does not have a title, or someone did not enter its title.

The last condition can be removed if the filtering condition is:

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

With this refinement of the filtering (what happens if the title is NULL?) we get a result more in line with expectations. But the first condition is still not fulfilled. Searching should work well when only fragmentary knowledge is available. The query technique must therefore use the LIKE condition:

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

or better still:

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

LIKE, coupled with %, means that all records are displayed which have the search term anywhere within them. % is a wildcard for any number of characters before or after the search term. Various questions still remain after this version of the query has been built:

- It is common to use lower case letters for search terms. So what result do I get if I type "hitch" instead of "Hitch"?
- What other conventions in writing need to be considered?
- What about fields that are not formatted as text fields? Can you search for dates or numbers with the same search field?

- And what if, as in the case of the filter, you want to prevent NULL values in the field from causing all the records to be displayed?

The following variant covers one or two of these possibilities:

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

The condition changes the search term and the field content to lower case. This also allows whole sentences to be compared.

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

The IFNULL function must occur only once, so that when the Searchterm is NULL, LOWER("Title") LIKE LOWER("Title") is queried. And as the title should be a field that cannot be NULL, in such cases all records are displayed. Of course, for multiple field searches, this code becomes correspondingly longer. In such cases it is better to use a macro, to allow the code to cover all the fields in one go.

But does the code still work with fields that are not text? Although the LIKE condition is really tailored to text, it also works for numbers, dates, and times without needing any alterations. So in fact text conversion need not take place. However, a time field that is a mixture of text and numbers cannot interact with the search results – unless the query is broadened, so that a single search term is subdivided across all the spaces between the text and numbers. This, however, will significantly bloat the query.

Tip

The queries that are used for filtering and searching records can be directly embedded in the form.

The whole condition that has been put together above can be entered into the row filter using the form properties.

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

then becomes a form that uses the content of the Media table.

Under "Filter" we have

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

In the filter entry, take care that the condition is put in brackets and works with the term "Table"."Field".

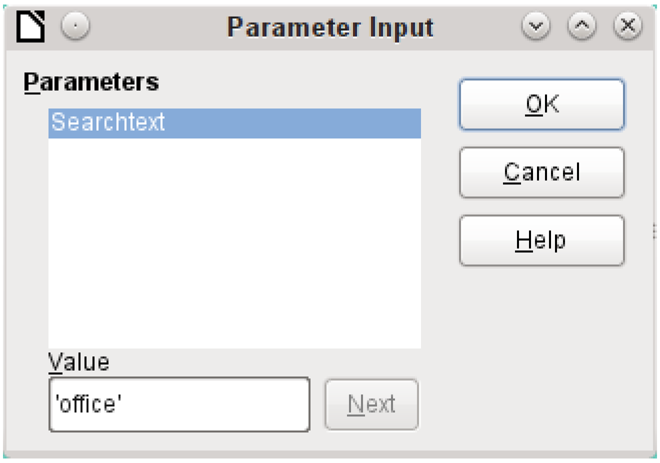
The advantage of this variant is that the filter can be switched on and off when the form is open.

Searching with LOCATE

Searching with LIKE is usually satisfactory for databases with fields containing text in amounts that can be easily scanned by eye. But what about Memo fields, which can contain several pages of text? In that case, the search needs to determine where the specified text can be found.

To locate text exactly, HSQLDB has the LOCATE function. LOCATE takes a search term (the text you want to look for) as its argument. You can also add a position to be searched. In short: LOCATE(Search term, Database text field, Position).

The following explanation uses a table called Table. The primary key is called ID and must be unique. There is also a field called Memo which was created as a field of type Memo (LONGVARCHAR). This Memo field contains a few sentences from this handbook.



Searchtext

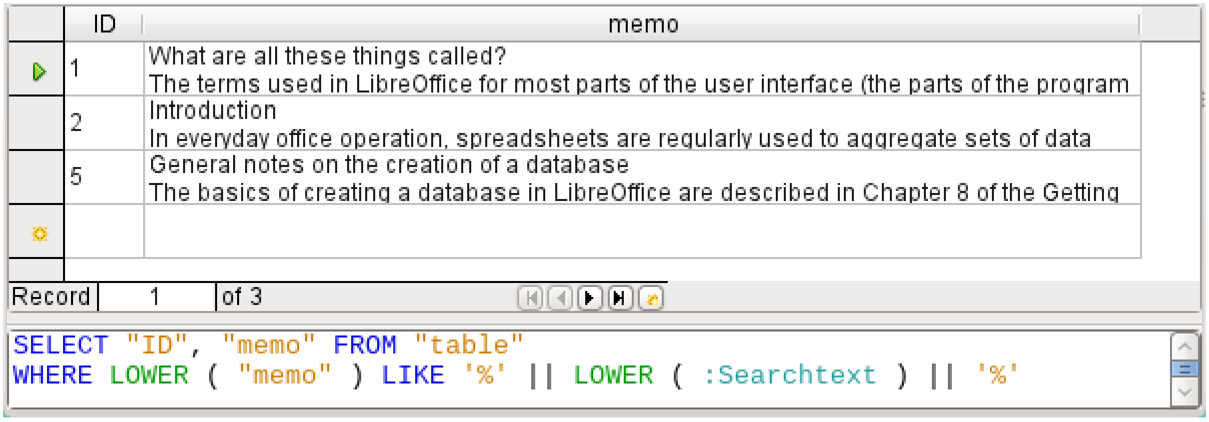
Value
'office'

Figure 416: Enter the string 'office' in the Parameter Input dialog

Note

The screenshots for this chapter come from Example_Autotext_Searchmark_Spelling.odb, which is included in the example databases for this book.

The query examples are presented as parametered queries. The search text to be entered is always ‘office’.



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

Record 1 of 3

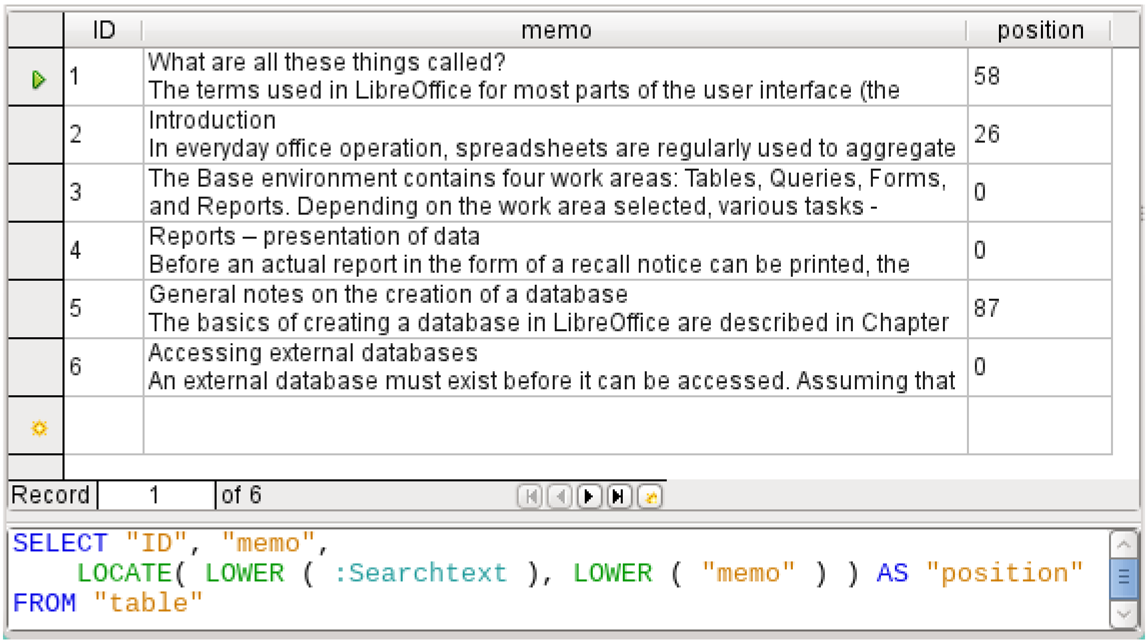
```
SELECT "ID", "memo" FROM "table"
WHERE LOWER ( "memo" ) LIKE '%' || LOWER ( :Searchtext ) || '%'
```

Figure 417: Search results and SQL code using LIKE

First we use LIKE. LIKE can only be used in conditions. If the search text is found anywhere, the corresponding record is shown. The comparison is between a lower-case version of the field content, using LOWER("Memo") and a lower-case version of the search text using LOWER(:Searchtext), to make the search case-insensitive. The longer the text in the memo field, the harder it becomes to see the term in the retrieved text.

LOCATE shows you more accurately just where your search term occurs. In records 1 and 2, the term does not occur. In this case LOCATE gives the position as ‘0’. It is easy to confirm the figure given for record 5: the string ‘Office’ begins at position 6. Naturally it would also be possible to display the results from LOCATE in the same way as for LIKE.

In the Hits column, the search results are displayed more accurately. The previous query has been used as a basis for this one. This allows the word “Position” to be used in the outer query instead of having to repeat LOCATE(LOWER(:Searchtext),LOWER("Memo")) each time. In principle, this is no different from saving the previous query and using it as a source for this one.



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

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

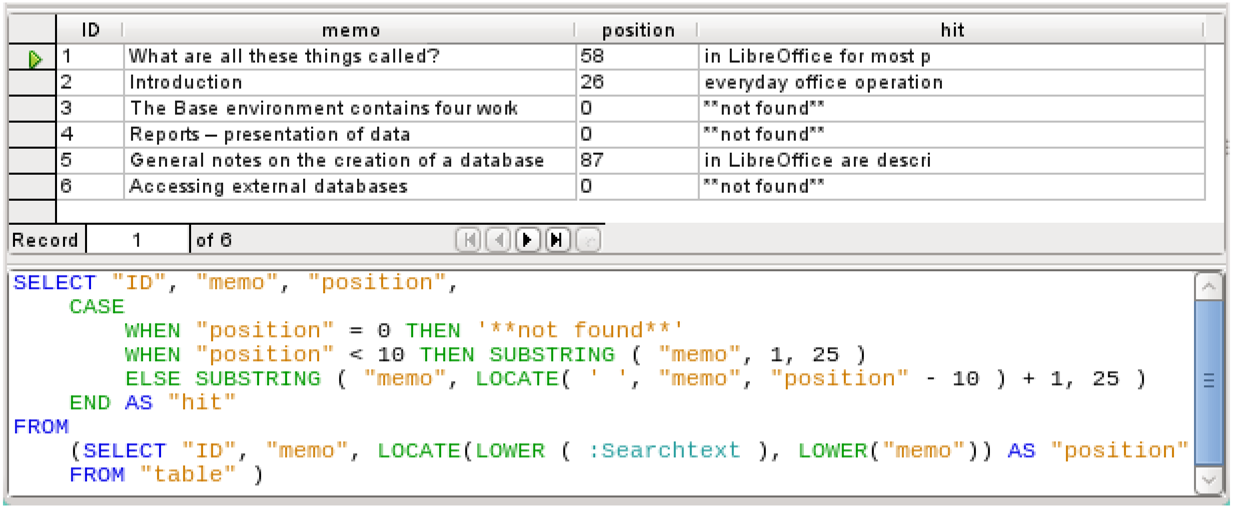
Figure 418: Search results and SQL code using LOCATE (version 1 – no attempt to show text around location of match)

`"Position" = 0` means that there is no result. In this case `'**not found**'` is displayed.

`"Position" < 10` means that the search term occurs right at the beginning of the text. 10 characters can easily be scanned by eye. Therefore the entire text is displayed. Here instead of `SUBSTRING("Memo",1)`, we could have used just "Memo".

For all other hits, the search looks for a space ' ' up to 10 characters before the search term. The displayed text does not start in the middle of a word but after a space. `SUBSTRING("Memo",LOCATE(' ',"Memo","Position"-10)+1)` ensures that the text starts at the beginning of a word which lies 10 characters at most before the search term.

In practice we would want to use more characters, as there are many words longer than that, and the search term might lie within another word with more than 10 characters in front of it. LibreOffice contains the search term “office” with the “O” as the sixth character. If the search term had been “hand”, Record 4 would have been fatal for the display. It contains the word “LibreOffice-Handbooks” which has 12 characters to the left of “hand”. If a maximum of 10 characters to the left was searched for spaces, the first one found would have been the character following the comma. This would be shown in the “Hits” column as beginning with ‘the built-in help system...’



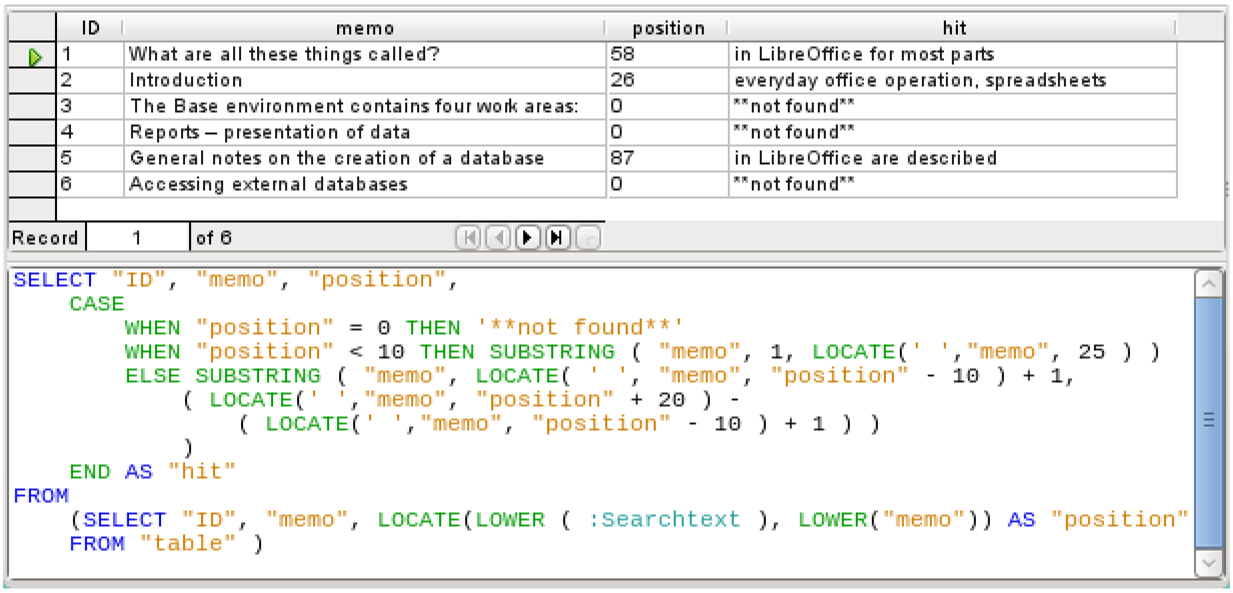
ID	memo	position	hit
1	What are all these things called?	58	in LibreOffice for most p
2	Introduction	26	everyday office operation
3	The Base environment contains four work	0	**not found**
4	Reports – presentation of data	0	**not found**
5	General notes on the creation of a database	87	in LibreOffice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

Record 1 of 6

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

Figure 419: Search results and SQL code using LOCATE (version 2 – displays text around location of match but partial words appear in hit column)

The query technique is the same as for the previous query. Only the length of the hit to be displayed has been reduced to 25 characters. The SUBSTRING function requires as arguments the text to be searched, then the starting position for the result, and as a third optional argument, the length of the text string to be displayed. Here it has been set quite short for demonstration purposes. An advantage of shortening it is that storage requirements for large numbers of records are reduced, and the location can be easily seen. A visible disadvantage of this type of string shortening is that the cut is made in strict accordance with the 25 character limit, without consideration of where words begin.



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

Record 1 of 6

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

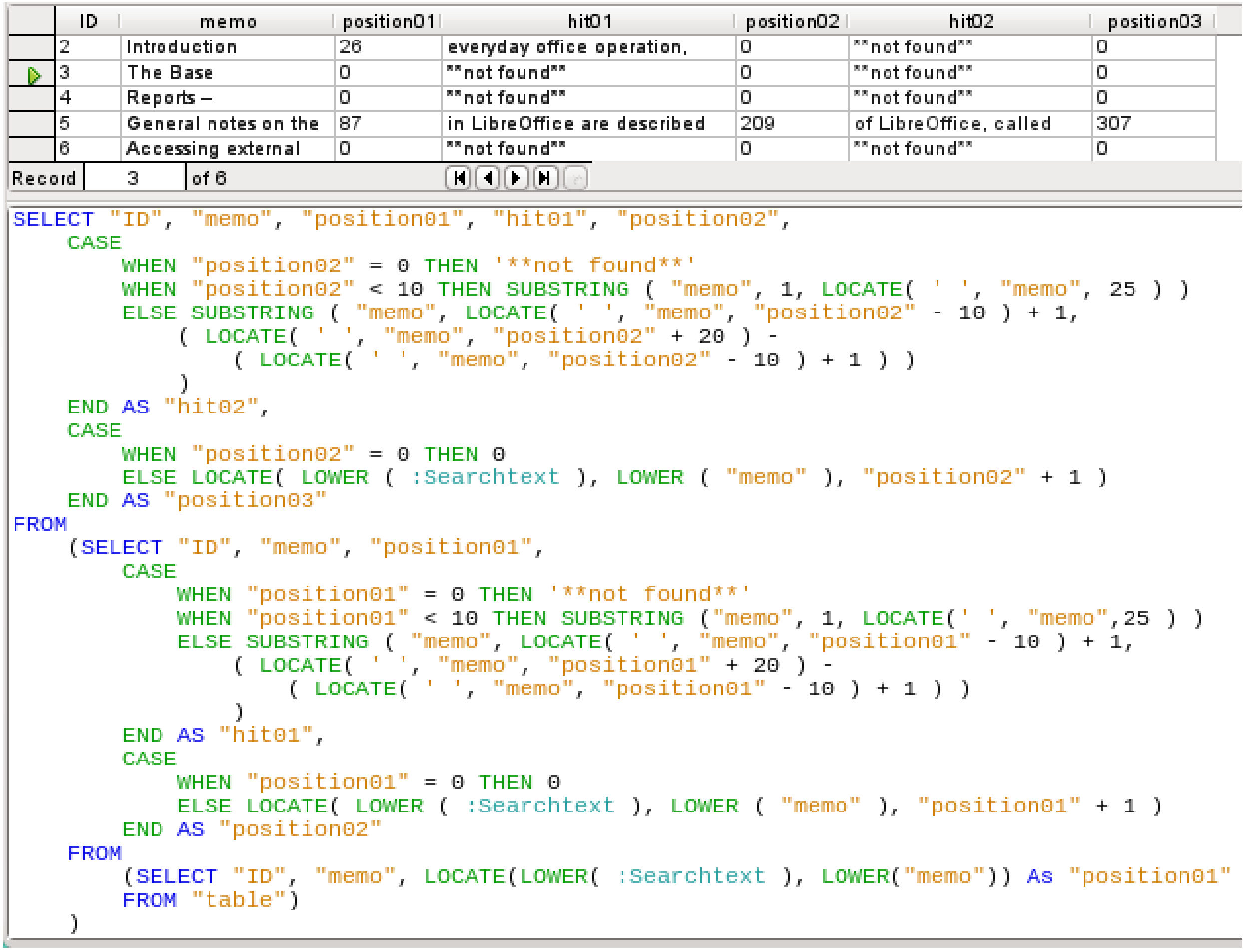
Figure 420: Search results and SQL code using LOCATE (version 3 – displays text around location of match with no partial words in hit column)

Here we search from the 25th character in the “hits” to the next space character. The content to be displayed lies between these two positions.

It is much simpler if the match shows up at the beginning of the field. Here `LOCATE(' ',"Memo",25)` gives the exact position where the text begins. Since we want the text to be displayed from the beginning, this complies exactly with the length of the displayable term.

The search for the space following the search term is no more complicated if the term lies further on in the field. The search simply begins where the match is. Then a further 20 characters are counted, which are to follow under all circumstances. The next space after that is located using `LOCATE(' ',"Memo","Position"+20)`. This gives only the location within the field as a whole, not the length of the string to be displayed. For that, we need to subtract the position at which the display of the matching text should start. This has already been set within the query by `LOCATE(' ',"Memo","Position"-10)+1`. In this way, the correct length of the text can be found.

The same technique can be used to string queries together. The previous query now becomes the data source for the new one. It has been inserted, enclosed in parentheses, under the term FROM. Only the fields are renamed to some extent as there are now multiple positions and matches. In addition, the next position is given a reference using `LOCATE(LOWER(:Searchtext),LOWER("Memo"),"Position01"+1)`. This means that searching starts again at the position after the previous matching text.



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

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

Figure 421: Search results and SQL code using LOCATE (version 4 – displays up to three matches per record)

The outermost query sets the corresponding fields for the other two queries, and also provides “hit02” using the same method as was previously used for “hit01”. In addition, this outermost query determines if there are any further matches. The corresponding position is given as “Position03”. Only record 5 has further matches, and these could be found in a further subquery.

The stacking of queries shown here could be carried further if desired. However, the addition of each new outer query puts an additional load on the system. It would be necessary to carry out some tests to determine how far it was useful and realistic to go. Chapter 9, Macros, shows how macros can be used to find all matching text strings in a field through the use of a form.

Handling images and documents in Base

Base forms use graphical controls to handle images. If you are using an internal HSQLDB database, graphical controls are the only way to read images out of the database without using macros. They can also be used as links to images outside the database file.

Reading images into the database

The database requires a table which fulfills at least the following conditions:

Table 16: Requirements for a table to read images into a database

Field name	***Field type***	***Description***
ID	Integer	ID is the primary key of this table.
Image	Image	Contains the image as binary data.

A primary key must be present, but it does not have to be an integer. Other fields that add information about the image should be added.

The data that will be read into the image field is not visible in a table. Instead you see the word <OBJECT>. In the same way, images cannot be entered directly into a table. You have to use a form that contains a graphical control. The graphical control opens when clicked to show a file picker dialog. Subsequently it shows the image that was read in from the selected file.

Pictures that are to be inserted directly into the database should be as small as possible. As Base provides no way (except by using macros) to export images in their original size, it makes sense to use only the size necessary, for example printing in a report. Original images in the megapixel range are completely unnecessary and bloat the database. After adding only a few images, the internal HSQLDB gives a `Java.NullPointerException` and can no longer store the record. Even if the images are not quite so big, it may happen that the database becomes unusable.

In addition, images should not be integrated into tables that are designed to be searched. If, for example, you have a personnel database, and images for use in passes are to be included, these are best stored in a separate table with a foreign key in the main table. This means that the main table can be searched significantly faster, as the table itself does not require so much memory.

Linking to images and documents

With a carefully designed folder structure, it is more convenient to access external files directly. Files outside the database can be as large as required, without having any effect on the working of the database itself. However, this also means renaming a folder on your own computer or on the Internet can cause you to lose access to the file.

If you do not want to read images directly into the database but only link to them, you need to make a small change to the previous table:

Table 17: Requirements for a table to link to external images

Field name	***Field type***	***Description***
ID	Integer	ID is the primary key of this table.
Image	Text	Contains the path to the image.

If the field type is set to text, the graphical control on the form will transmit the path to the file. The picture can still be accessed by the graphical control exactly like an internal image.

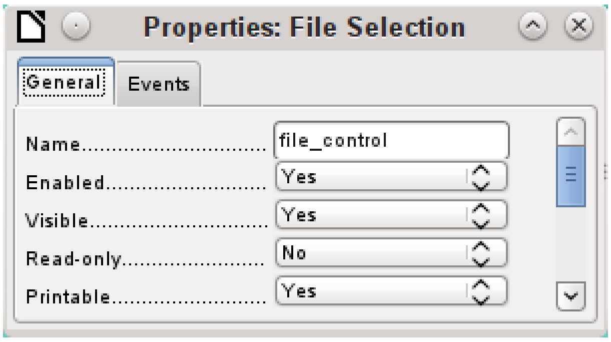
You cannot do the same thing with a document. It is not possible even to read the path in, as graphical controls are designed for graphical images and the filepicker dialog shows only files with a graphical format.

With an image, the content can at least be seen in the graphical control, using the path to the file. With a document, there can be no display even if the path is stored in a table. First we need to enlarge the table somewhat so that at least a small amount of information about the document can be made visible.

Table 18: Requirements for a table to link to external documents

Field name	***Field type***	***Description***
ID	Integer	ID is the primary key of this table.
Description	Text	Description of the document, search terms
File	Text	Contains the path to the document.

To make the path to the document visible, we need to build a file selection field into the form.



Name: file_control
Enabled: Yes
Visible: Yes
Read-only: No
Printable: Yes

Figure 422: Properties dialog (General tab) for a file selection field

A file selection field has no tab for data in its properties dialog. It is therefore not bound to any field in the underlying table.

Linking documents with an absolute path

Using the file selection field, the path can be displayed but not stored. For this a special procedure is necessary which is tied to **Events > Text modified**:

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

The event that triggers the procedure is passed to it and helps to find the form and the field in which the path is to be stored. Using `oEvent AS OBJECT` makes access simpler when another user wants to use a macro with the same name in a subform. It makes the file selection field accessible via `oEvent.Source.Model`. The form is accessed as the Parent of the file selection field. The name of the form is therefore irrelevant. From the form, the field called "graphical_control" can now be accessed. This field is normally used to store the paths to image files. In this case, the URL of the selected file is written into it. To ensure that the URL works with the conventions of the operating system, the text in the file selection field is converted into a generally valid form by using ConvertToUrl.

The database table now contains a path with the absolute format: `file:///....`.

If path entries are read using a graphical control, this will yield a relative path. To make this usable, it must be improved. The procedure for doing this is much lengthier, as it involves a comparison between the input path and the real one.

Linking documents with a relative path

The following macro is bound to the "Text modified" property of the file selection field.

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

First, as in all procedures, the variables are declared. Then the fields that are important for the entry of paths are searched. The whole of the following code is then carried out only if there is actually something in the file selection field, i.e. it has not been emptied by a record change.

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

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

The path to the database file is read. This is carried out, as shown above, first by reading the whole URL, then splitting it into an array so that the first element of the array contains the direct path.

Then all the elements of the path found in the file selector field are read into the array `ar`. The separator is /. This can be done directly in Linux. In Windows, the content of `oField2` must be converted into a URL, which will use a forward slash (not a backslash) as the path delimiter.

The purpose of the split is to get the path to the file by simply cutting off the filename at the end. Therefore, in the next step, the path to the file is put together again and placed in the variable stText. The loop ends not with the last element in the ar array but with the previous element.

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

The final / is removed again, since otherwise an empty array value would appear in the following array, which would interfere with the path comparison. For correct comparison, the text must be converted into a proper URL beginning with `file:///`. Finally the path to the database file is compared with the path which has been created.

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

`ar1ar2` array is compared step by step in a loop.

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

The following code is executed only if the number i is no greater than the number of elements in ar1. If the value in ar2 is the same as the corresponding value in ar1, and no incompatible value has been found up to this point, the common content is stored in a variable that can finally be cut off from the path value.

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

If there is a difference at any point between the two arrays, then for each different value, the sign for going up one directory will be added to the variable stUrl.

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

As soon as the index stored in i is greater than the number of elements in ar1, each further value in ar2 will cause a further ../ to be stored in the variable stUrl.

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

When the loop through ar2 is complete, we have established whether and by how much the file that is to be accessed is higher in the tree than the database file. Now `stUrl_complete` can be created out of the text in the file selector field. This also contains the filename. Finally the value is transferred into the graphical control. The URL value begins with stUrl, which contains the necessary number of dots (../). Then the beginning of `stUrl_complete`, the part that proved to be the same for the database and the external file, is cut off. The way to cut the string is stored in `stUrl_cut`.

Displaying linked images and documents

Linked pictures can be displayed directly in a graphical control. But a larger display would be better at showing details.

Documents are not normally visible in Base.

To make this type of display possible, we again need to use macros. This macro is launched using a button on the form that contains the graphical control.

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

The graphical control in the form is located. As the table does not contain the image itself but only a path to it stored as a text string, this text is retrieved using `getString`.

Then the path to the database file is determined. The odb file, the container for the forms, is accessed using `oDoc.Parent`. The whole URL, including the filename, is read out using `oDoc.Parent.Url`. The filename is also stored in `oDoc.Parent.Title`. The text is separated using the split function with the filename as a separator. This gives the path to the database file as the first and only element of the array.

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

External programs can be launched using the structure com.sun.star.system.SystemShellExecute. The absolute path to the file, put together from the path to the database file and the internally-stored relative path from the database file, is passed to the external program. The operating system's graphical interface determines which program is called on to open the file.

The oShell.execute command takes three arguments. The first is an executable file or the path to a data file that is linked to a program by the system. The second is an argument list for the program. The third is a number that determines how errors are to be reported. The possibilities are 0 (default error message), 1 (no message), and 2 (only allow the opening of absolute URLs).

Reading documents into the database

When reading in the documents, the following conditions should always be observed:

- The larger the documents, the more unwieldy the database becomes. Therefore for large documents, an external database is better than the internal one.
- Like images, documents are not searchable. They are stored as binary data and can therefore be put into an image field.
- Documents read into the internal HSQLDB database can only be read out using macros. You can't do it using SQL queries.

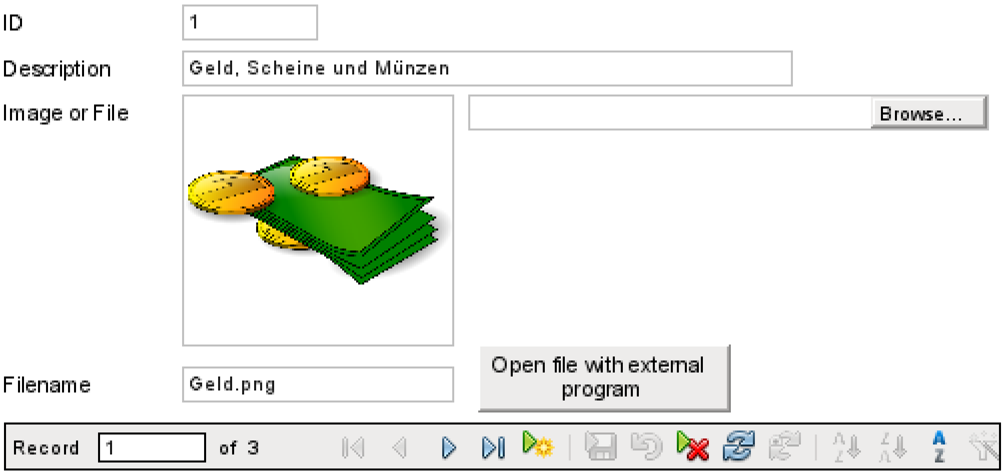
The following macros for reading in and out depend on a table that includes a description of the data and the original filename, as well as a binary version of the file. The filename is not automatically stored along with the file, but it can provide useful information about the type of data stored in a file which is to be read out. Only then can the file safely be read by other programs.

The table contains the following fields:

Table 19: Requirements for a table to read documents in and out

Field name	**Field type**	**Description**
ID	Integer	ID is the primary key of this table.
Description	Text	Document description, search terms, etc.
File	Image	The image or file in binary form.
Filename	Text	The name of the file, including the file suffix. Important for subsequent reading.

The form for reading files in and out looks like this:



ID: 1

Description: Geld, Scheine und Münzen

Image or File

Browse…

Filename: Geld.png

Open file with external program

Record 1 of 3

Figure 423: Form for reading files in and out

If image files are present in the database, they can be viewed in the form’s graphical control. All other types of file are hidden.

The following macro for reading a file in is triggered by Properties: File selection → Events → Text modified.

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

As the triggering event for the macro provides the name of another form field, it is not necessary to check if the fields are in the main form or a subform. All that is necessary is that all the fields must be in the same form.

The field “txt_filename” stores the name of the file to be searched for. In the case of images, this name must be entered by hand without using a macro. Here instead, the filename is determined via a URL and automatically entered when the data is read in.

The field “graphical_control” stores the actual data both for images and for other files.

The complete path, including the filename, is read from the file selector field using `oFeld.Text`. To ensure that the URL is not affected by OS-specific conditions, the text that has been read out is converted into the standard URL format using `ConvertToUrl`. This universally valid URL is split up within an array. The separator is /. The last element of the path is the filename. `Ubound(ar)` gives the index for this last element. The actual filename can then be read out using `ar(Ubound(ar))` and transferred to the field as a string.

To read in the file itself requires `UnoService com.sun.star.ucb.SimpleFileAccess`. This service can read the content of the file as a stream of data. This is stored temporarily in the object oStream and then inserted as a data stream into the field bound to the “File” field in the table. This requires the length of the data stream to be provided as well as the oStream object.

The data are now inside the form field just as with a normal entry. However if the form is simply closed at this point, the data are not stored. Storage requires the Store button in the navigation bar to be pressed; it also happens automatically upon moving to the next record.

Determining the names of image files

In the above method, it was briefly mentioned that the name of the file used for input into a graphical control cannot be directly determined. Here is a macro for determining this filename, which fits the form above. The filename cannot be determined with certainty by an event directly bound to the graphical control. Therefore the macro is launched using **Form Properties > Events > Before record action**.

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

Before the record action, two implementations with different implementation names are carried out. The form is most easily accessible using the implementation ODatabaseForm.

In the graphical control, the URL of the data source can be accessed using `ImageUrl`. This URL is read, the filename is determined using the previous procedure `FileInput_withName`, and is transferred to the field `txt_filename`.

Removing image filenames from memory

If after the above macro is run, you move to the next record, the path to the original image is still available. If a non-image file is now read in using the file selector field, the filename for the image will overwrite the name of that file, unless you use the following macro.

The path cannot be removed by the previous macro, since the image file is only read in when the record is saved. Removing the path at that point would delete the image.

The macro is launched using **Form Properties > Events > After record action**.

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

As in the “ImageRead” procedure, the graphical control is accessed. If there is an entry in ImageUrl, it is removed.

Reading and displaying images and documents

For both non-graphical files and original size images, the **Open file with external program** button must be pressed. Then the files in the temporary folder can be read and displayed using the program linked to the file suffix in the operating system.

The macro is launched using **Properties: Button > Events > Execute action**.

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

The position of the other affected fields in the form is given by the button. If a filename is missing, the file is simply given the name “File”.

The content of the form control “graphical_control” corresponds to that of the File field in the table. It is read out as a data stream. The path to the temporary folder is used as a path for this data; it can be set using **Tools > Options > LibreOffice > Paths**. If the data is subsequently to be used for other purposes, and not just displayed, it can be copied from this path. Within the macro, the file is opened directly after successful reading, using the program that has been bound to the file suffix by the graphical user interface of the operating system.

Code snippets

These code snippets come from queries to mailing lists. Particular problems arise that might perhaps be useful as solutions for your own database experiments.

Getting someone’s current age

A query needs to calculate a person's actual age from a birth date. See also the functions in the appendix to this Base Guide.

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

This query gives the age as a difference in years. But, the age of a child born on 31 December 2011 would be given as 1 year on 1 January 2012, because that is a leap year. So we also need to consider the position of the day within the year. This is accessible using the `DAYOFYEAR()` function. Another function will carry out the comparison.

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

Now we get the correct current age in years.

CASEWHEN can also be used to make the text Birthday today appear in another field, if `DAYOFYEAR("Birthdate") = DAYOFYEAR(CURDATE())`.

A subtle objection might now arise: "What about leap years?". For persons born after 28 February, there will be an error of one day. Not a serious problem in everyday use, but should we not strive for accuracy?

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

The code above achieves this goal. As long as the month of the birth date is greater than the current month, the year difference function will subtract one year. Equally one year will be subtracted when the two months are the same, but the day of the month for the birth date is greater than the day in the current date. However, this formula is not comprehensible to the GUI. Only *Direct SQL-Command* will handle this query successfully and that would prevent our query from being edited. But the query needs to be editable, so here is how to trick the GUI:

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

With this formulation, the GUI no longer reacts with an error message. The age is now given accurately even in leap years and the query still remains editable.

Showing birthdays that will occur in the next few days

Using a small calculation snippet, we can determine from the table who will be celebrating their birthdays within the next eight days.

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

The query shows all records whose date entry lies between the current day of the year and the following 7 days.

To show 8 days even at the end of a year, the day on which the year began must be thoroughly checked. This check occurs only for day numbers that are at most 7 days later than the last day number for the current year (usually 365) plus the day number for the current date. If the current date is more than 7 days from the end of the year, the total is <1. No record in the table has a date like that, so in such cases this partial condition is not fulfilled.

In the above formula, leap years will give a wrong result, as their dates are displaced by the occurrence of 29th February. The code needs to be more extensive to avoid this error:

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

Leap years can be recognized by having 366 as the total number of days rather than 365. This is used for the corresponding determination.

On the one hand, each date value must be tested to see if it lies in a leap year, and also for the correct count for the 60th day (31 days in January and 29 in February). In this case, all following `DAYOFYEAR` values for the date must be increased by 1. Then 1 March in a leap year will correspond exactly to 1 March in a normal year.

On the other hand, the current year `(CURDATE())` must be tested to see if it is in fact a leap year. Here too the number of days must be increased by 1.

Displaying the end value for the next 8 days is not so simple either, since the year is still not included in the query. However this would be an easy condition to add: `YEAR("Date") = YEAR(CURDATE())` for the current year or `YEAR("Date") = YEAR(CURDATE()) + 1` for the next one.

Adding days to the date value

When loaning out media, the library might want to know the exact day on which the medium should be returned. The internal HSQLDB does not provide the `DATEADD()` function which is available in many external databases and also in the internal Firebird. Here follows a roundabout way of achieving this for a limited time span.

First a table is created containing a sequence of dates covering the desired time span. For this purpose, Calc is opened and the name “ID” is placed in field A1 and “Date” in field B1. In Field A2 we enter 1 and in field B2 the starting date, for example 01/15/2015. Select A2 and B2 and drag them down. This will create a sequence of numbers in column A and a sequence of dates in column B.

Then this whole table, including the column headings, is selected and imported into Base: **right click > Paste > Table name > Date**. Under options, Definition and Data, and Use first row as column names are clicked. All the columns are transferred. After that, make sure that the ID field is given the type Integer [INTEGER] and the Date field the type Date [DATE]. A primary key is not necessary as the records are not going to be altered later. Since a primary key has not been defined, the table is write-protected.

Tip

You can also use a query technique to create such a view. If you use a filter table, you can even control the start date and the range of date values.

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

This view accesses a table that contains only the numbers from 1-31 and is write-protected. Another filter table contains the starting year and the year range that the view should cover. The date is put together from these, creating a date expression (year, month, day) in text, that can then be converted into a date. HSQLDB accepts all days up to 31 a month and strings like 02/31/2015. However 02/31/2015 is transmitted as 3/03/2015. Therefore in preparing the view, you must use DISTINCT to exclude duplicate date values.

Here the following view is effective:

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

Using line numbering, the date value is converted into a number. As you cannot delete data in a view, no extra write protection is needed.

Using a query we can now determine a specific date, for example the date in 14 days time:

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

The first column shows the loan date. This column is accessed by a correlating subquery which again is divided into two queries. `SELECT "ID" FROM "Date"` gives the value of the ID field, corresponding to the issue date. 14 days is added to the value. The result is assigned to the ID field by the outer subquery. This new ID then determines which date goes into the date field.

In the display of this query, the date type is not automatically recognized, so that it becomes necessary to use formatting. In a form, the corresponding display can be stored, so that each query will yield a date value.

A direct variant for determining the date value is possible using a shorter way:

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

The numeric value returned can be formatted inside a form as a date, using a formatted field. However it takes a lot of work to make it available for further SQL processing in a query.

Adding a time to a timestamp

MySQL has a function called `TIMESTAMPADD()`. A similar function does not exist in HSQLDB. But the internal numeric value of the timestamp can be used to do the addition or subtraction, using a formatted field in a form.

Unlike the addition of days to a date, times cause a problem which might not be obvious in the beginning.

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

The new calculated time is based on the difference from the system's zero time. As in date calculations, this is the date of 12/30/1899.

Note

The zero date of 12/30/1899 is supposed to have been chosen because the year 1900, unlike most years divisible by 4, was not a leap year. So the tag '1' of the internal calculation was moved back to 12/31/1899 and not 01/01/1900.

The difference is expressed in seconds, but the internal number counts days as numbers before the decimal point, and hours, minutes and seconds as decimal places. Since a day contains 60*60*24 seconds, the second count must be divided by 86400 to be able to calculate the days and fractions of days correctly. If the internal HSQLDB is to give decimal places at all, they must be included in the calculation, so instead of 86400, we must divide by 86400.0000000000. Decimal places in a query must use a decimal point as separator, regardless of locale conventions. The result will have 10 decimal places after the dot.

To this result must be added the total hours as a fractional part of a day. The calculated figure, suitably formatted, can be created in the query. The formatting is not saved but it can be transferred with the correct format using a formatted field in a form or report.

If minutes or seconds are to be added, be careful that they are supplied as fractions of a day.

If the date falls within November, December, January, etc. there are no problems with the calculation. They appear quite accurate: adding 36 hours to a timestamp of 01/20/2015 13:00:00 gives 01/22/2015 00:00:00. But things are different for 04/20/2015 13:00:00. The result is 04/22/2015 00:00:00. The calculation goes wrong because of summer time. The hour “lost” or “gained” by the time change is not taken into account. Within a single time zone, there are various ways of getting a “correct” result. Here is a simple variation:

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

Instead of counting hours, minutes and seconds since the date origin, they are counted from the current date. On 05/20/2015 the time is 13:00 but without summer time, it would be shown as 12:00. The HOUR function takes summer time into account and gives 13 hours as the hourly part of the time. This can then be added correctly to the daily part. Minutes and seconds are dealt with in exactly the same way. Finally the extra hours are added as a fractional part of a day and the whole thing is displayed as a calculated timestamp using cell formatting.

Two things need to be kept in view in this calculation:

- When moving from winter time to summer time, the hourly values do not come out correctly. This can be corrected using an ancillary table, that takes the dates for the beginning and end of summer time and corrects the hourly count. A somewhat complicated business.
- The display of times is possible only with formatted fields. The result is a decimal number, not a timestamp that could be stored directly as such in the database. Either it must be copied within the form or converted from a decimal number to a timestamp by using a complicated query. The breaking point in the conversion is the date value, as leap years or months with different numbers of days may be involved.

Getting a running balance by categories

Instead of using a household book, a database on a PC can simplify the tiresome business of adding up expenses for food, clothing, transport, and so on. We want most of these details to be immediately visible in the database, so our example assumes that income and expenditure will be stored as signed values in one field called Amount. In principle, the whole thing can be expanded to cover separate fields and a relevant summation for each.

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

This query causes for each new record a direct calculation of the current account balance. At the same time the query remains editable because the Balance field is created through a correlating sub-query. The query depends on the automatically created primary key ID to calculate the state of the account. However, balances are usually calculated on a daily basis. So we need a date query.

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

The expenditure now appears sorted and summed by date. There still remains the question of the category, since we want corresponding balances for the individual categories of expenditure.

```
SELECT "ID", "Date", "Amount", "Acct_ID",

( SELECT "Acct" FROM "Acct" WHERE "ID" = "a"."Acct_ID" ) AS "Acct_name",

( SELECT SUM( "Amount" ) FROM "Cash" WHERE "Date" <= "a"."Date" AND
"Acct_ID" = "a"."Acct_ID" ) AS "Balance",

( SELECT SUM( "Amount" ) FROM "Cash" WHERE "Date" <= "a"."Date" ) AS
"Total_balance"

FROM "Cash" AS "a" ORDER BY "Date", "ID" ASC
```

This creates an editable query in which, in addition to the entry fields (`Date, Amount, Acct_ID`), the account name, the relevant balance, and the total balance appear together. As the correlating subqueries are partially based on previous entries (`"Date" <= "a"."Date"`) only new entries will go through smoothly. Alterations to a previous record are initially detectable only in that record. The query must be updated if later calculations dependent on it are to be carried out.

Line numbering

Automatically incrementing fields are fine. However, they do not tell you definitely how many records are present in the database or are actually available to be queried. Records are often deleted and many users try in vain to determine which numbers are no longer present in order to make the running number match up.

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

The ID field is read, and the second field is determined by a correlating sub-query, which seeks to determine how many field values in ID are smaller than or equal to the current field value. From this a running line number is created.

Each record to which you want to apply this query contains fields. To apply this query to the records, you must first add these fields to the query. You can place them in whatever order you desire in the SELECT clause. If you have the records in a form, you need to modify the form so that the data for the form comes from this query.

For example the record contains field1, field2, and field3. The complete query would be:

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

A numbering for a corresponding grouping is also possible:

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

Here one table contains different calculated numbers. ("Calculation"). For each calculated number, "Nr." is separately expressed in ascending order after sorting on the ID field. This produces a numbering from 1 upwards.

If the actual sort order within the query is to agree with the line numbers, an appropriate type of sorting must be mapped out. For this purpose the sort field must have a unique value in all records. Otherwise two place numbers will have the same value. This can actually be useful if, for example, the place order in a competition is to be depicted, since identical results will then lead to a joint position. In order for the place order to be expressed in such a way that, in case of joint positions, the next value is omitted, the query needs to be be constructed somewhat differently:

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

All entries are evaluated for which the Time field has a smaller value. That covers all athletes who reached the winning post before the current athlete. To this value is added the number 1. This determines the place of the current athlete. If the time is identical with that of another athlete, they are placed jointly. This makes possible place orders such as 1st Place, 2nd Place, 2nd Place, 4th Place.

It would be more problematic, if line numbers were required as well as a place order. That might be useful if several records needed to be combined in one line.

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

The second column still gives the place order. The third column checks first if only one person crossed the line with this time. If so, the place order is converted directly into a line number. Otherwise a further value is added to the place order. For the same time ("Time" = "a"."Time") at least 1 is added, if there is a further person with the primary key ID, whose primary key is smaller than the primary key in the current record ("ID" < "a"."ID"). This query therefore yields identical values for the place order so long as no second person with the same time exists. If a second person with the same time does exist, the ID determines which person has the lesser line number.

Incidentally, this sorting by line number can serve whatever purpose the users of the database want. For example, if a series of records are sorted by name, records with the same name are not sorted randomly but according to their primary key, which is of course unique. In this way too, numbering can lead to a sorting of records.

Line numbering is also a good prelude to the combining of individual records into a single record. If a line-numbering query is created as a view, a further query can be applied to it without creating any problem. As a simple example here once more is the first numbering query with one extra field:

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

This query is turned into the view View1. The query can be used, for example, to put the first three names together in one line:

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

In this way several records can be converted into adjacent fields. This numbering simply runs from the first to the last record.

If all these individuals are to be assigned the same surname, this can be carried out as follows:

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

Now that the view has been created, the family can be assembled.

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

In this way, in an address book, all members of one family ("Surname") can be collected together so that each address need be considered only once when sending a letter, but everyone who should receive the letter is listed.

We need to be careful here, as we do not want an endlessly looping function. The query in the above example limits the parallel records that are to be converted into fields to 3. This limit was chosen deliberately. No further names will appear even if the value of "Nr." is greater than 3.

In a few cases such a limit is clearly understandable. For example, if we are creating a calendar, the lines might represent the weeks of the year and the columns the weekdays. As in the original calendar only the date determines the field content, line numbering is used to number the days of each week continuously and then the weeks in the year become the records. This does require that the table contains a date field with continuous dates and a field for the events. Also the earliest date will always create an “Nr.” = 1. So, if you want the calendar to begin on Monday, the earliest date must be on Monday. Column 1 is then Monday, column 2 Tuesday and so on. The subquery then ends at "Nr." = 7. In this way all seven days of the week can be shown alongside each other and a corresponding calendar view created.

Getting a line break through a query

Sometimes it is useful to assemble several fields using a query and separate them by line breaks, for example when reading a complete address into a report.

The line break within the query is represented by `Char(13)`. Example:

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

This yields:

```
Firstname Surname
Road
Town
```

Such a query, with a line numbering up to 3, allows you to print address labels in three columns by creating a report. The numbering is necessary in this connection so that three addresses can be placed next to one another in one record. That is the only way they will remain next to each other when read into the report.

In some operating systems it is necessary to include char(10) alongside char(13) in the code.

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

Grouping and summarizing

For other databases, and for newer versions of HSQLDB, the `Group_Concat()` command is available. It can be used to group individual fields in a record into one field. So, for example, it is possible to store first names and surnames in one table, then to present the data in such a way that one field shows the surnames as family names while a second field contains all the relevant first names sequentially, separated by commas.

This example is similar in many ways to line numbering. The grouping into a common field is a kind of supplement to this.

Table 20: Names of individuals to be grouped

Surname	***Firstname***
Müller	Karin
Schneider	Gerd
Müller	Egon
Schneider	Volker
Müller	Monika
Müller	Rita

is converted by the query to:

Table 21: Names grouped by surname

Surname	***Firstnames***
Müller	Karin, Egon, Monika, Rita
Schneider	Gerd, Volker

This procedure can, within limits, be expressed in HSQLDB. The following example refers to a table called Name with the fields IDs, Firstname and Surname. The following query is first run on the table and saved as a view called View_Group.

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

You can read in the Queries chapter how this query accesses the field content in the same query line. It yields an ascending numbered sequence, grouped by Surname. This numbering is necessary for the following query, so that in the example a maximum of 5 first names is listed.

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

Using sub-queries, the first names of the group members are searched for one after another and combined. From the second sub-query onward you must ensure that 'NULL' values do not set the whole combination to 'NULL'. That is why a result of '' rather than 'NULL' is shown.

Base Guide 7.3

Chapter 9
Macros

General remarks on macros

In principle a database in Base can be managed without macros. At times, however, they may become necessary for:

- More effective prevention of input errors.
- Simplifying certain processing tasks (changing from one form to another, updating data after input into a form, and so on).
- Allowing certain SQL commands to be called up more easily than with the separate SQL editor.

You must decide for yourself how intensively you wish to use macros in Base. Macros can improve usability but are always associated with small reductions in the speed of the program, and sometimes with larger ones (when coded poorly). It may also pose issues of code maintenance in the long term. It is always better to start off by fully utilizing the possibilities of the database and the provisions for configuring forms before trying to provide additional functionality with macros. Macros should always be tested on larger databases to determine their effect on performance.

Macros are created using **Tools > Macros > Organize Macros > Basic**. A window appears which provides access to all macros. For Base, the important area corresponds to the filename of the Base file.

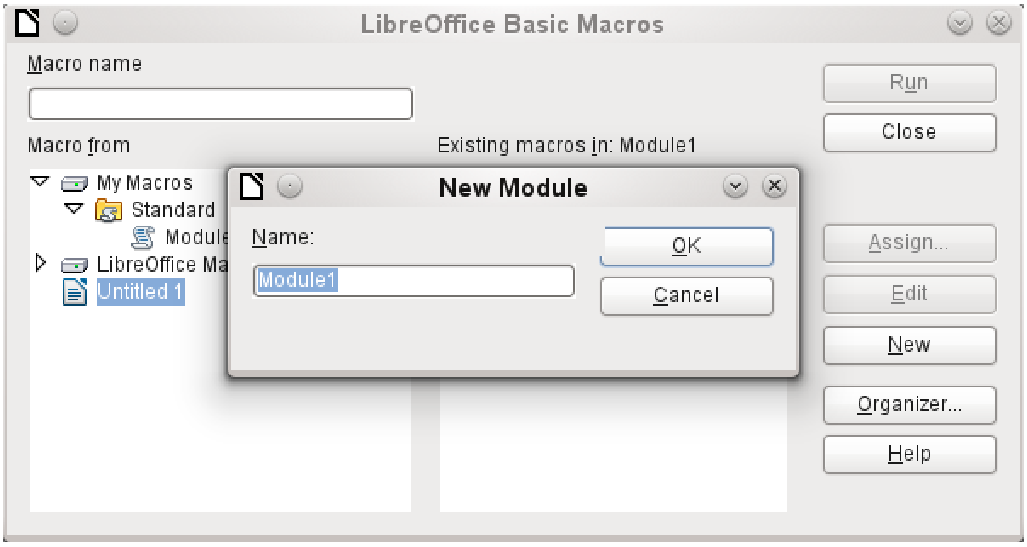


Figure 424: New Module dialog

The **New** button in the LibreOffice Basic Macros dialog opens the New Module dialog, which asks for the module name (the folder in which the macro will be filed). The name can be altered later if desired.

As soon as this is given, the macro editor appears. Its input area already contains the Start and the End for a subroutine:

```
REM  *****  BASIC  *****

Sub Main

End Sub
```

If macros are to be used, the following steps are necessary:

- Under **Tools > Options > Security > Macro Security** the security level should be reduced to Medium. If necessary, you can additionally use the Trusted sources tab to set the path to your own macro files to prevent later queries about the activation of macros.
- The database file must be closed and then reopened after the creation of the first macro module.

Some basic principles for the use of Basic code in LibreOffice:

- Lines have no line numbers, by default (though there is an option to enable them) and must end with a hard return.
- Functions, reserved expressions, and similar elements are not case-sensitive. So "String" is the same as "STRING" or "string" or any other combination of upper and lower case. Case should be used only to improve legibility. Names for constants and enumerations, however, are case sensitive the first time that they are seen by the macro compiler, so it is best to always write those using the proper case.
- One can distinguish between procedures (beginning with **`Sub`**) and functions (beginning with **`Function`**). Procedures were originally program segments without a return value, while functions return values that can be further processed. But this distinction is increasingly becoming irrelevant. People nowadays use terms such as “method” or “routine” whether there is a return value or not. A procedure can also have a return value (apart from “Variant”).

```
   Sub myProcedure As Integer 'This syntax for Sub is not officially
   supported.
End Sub
```

For further details, see Chapter 13, Getting Started with Macros, in the *Getting Started Guide*.

Note

Macros in the PDF and ODT versions of this chapter are colored according to the rules of the LibreOffice macro editor:

```
Macro designation
Macro comment
Macro operator
Macro reserved expression
Macro number
Macro character string
```

Note

There are several references to the LibreOffice *API Reference* throughout this chapter. In electronic versions of the chapter, such references will be hyperlinks. For those reading a hard copy, the API Reference can be found at https://api.libreoffice.org/docs/idl/ref/.

Macros in Base

Using macros

The “direct way”, using **Tools > Macros > Run Macro** is possible, but not usual for Base macros. A macro is normally assigned to an event and launched when that event occurs. Macros are used for:

- Handling events in forms
- Editing a data source inside a form
- Switching between form controls
- Reacting to what the user does inside a control

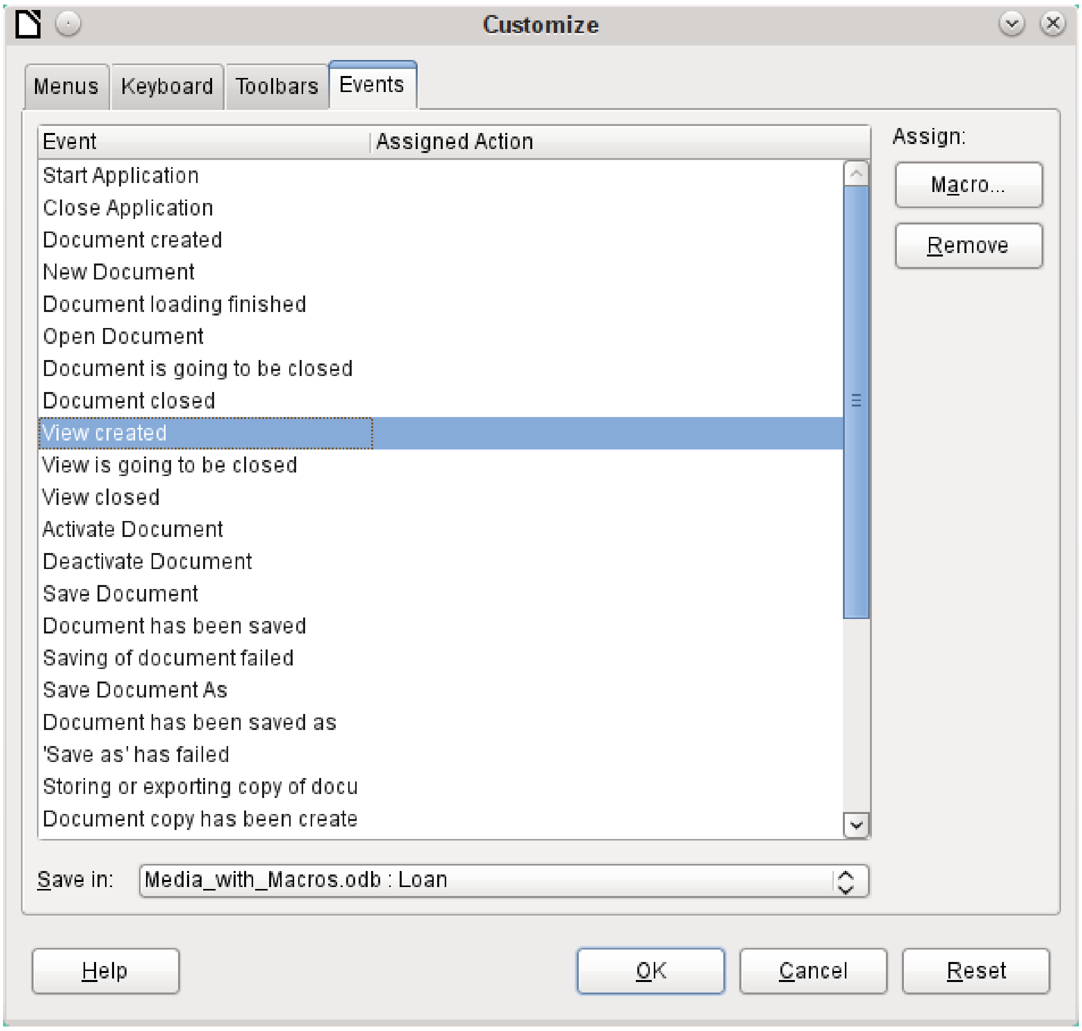
The “direct way” is not possible – not even for testing – when one of the objects **`thisComponent`** (see “Accessing forms” on page 429) or **`oEvent`** (see “Accessing form elements” on page 430) is to be used.

Assigning macros

If a macro is to be launched by an event, it must first be defined. Then it can be assigned to an event. Such events can be accessed through two locations.

Events that occur in a form when the window is opened or closed

Actions that take place when a form is opened or closed are registered as follows:



Event	Assigned Action
Start Application	
Close Application	
Document created	
New Document	
Document loading finished	
Open Document	
Document is going to be closed	
Document closed	
View created	
View is going to be closed	
View closed	
Activate Document	
Deactivate Document	
Save Document	
Document has been saved	
Saving of document failed	
Save Document As	
Document has been saved as	
'Save as' has failed	
Storing or exporting copy of docu	
Document copy has been create	

Save in: Media_with_Macros.odb : Loan

Figure 425: Customize dialog, Events tab

1) While designing the form, open the Events tab in **Tools > Customize**.
2) Choose the appropriate event. Some macros can only be launched when the View created event is chosen. Other macros, for example to create a full-screen form, should be launched by Open document.
3) Use the **Macro** button to find the macro you want and confirm your choice.

4) Under Save in, give the name of the form.
5) Confirm with **OK**.

Events in a form in an open window

Once the window is opened to show the overall content of the form, individual elements of the form can be accessed. This includes the elements you have assigned to the form.

The form elements can be accessed using the Form Navigator, as shown in the illustration below. They can equally well be accessed by using the contextual menus of individual controls within the form interface.

The events listed under **Form Properties > Events** all take place while the form window is open. The can be set separately for each form or subform in the form window.

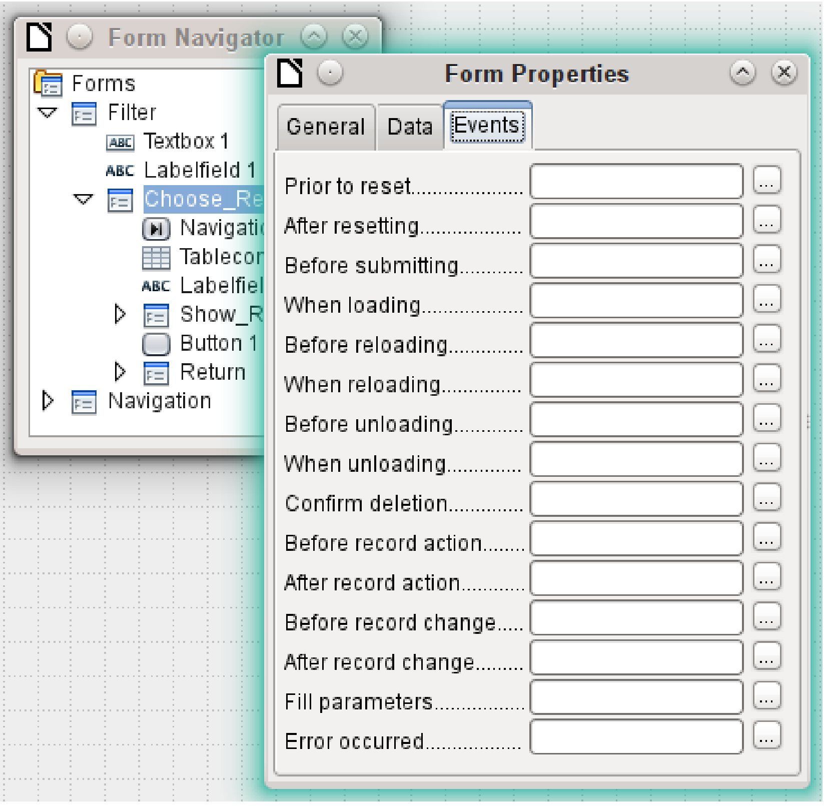


Figure 426: Form Properties dialog, Events tab

Note

Base uses the word “form” both for a window that is opened for the input of data, and for elements within this window that are bound to a specific data source (table or query).

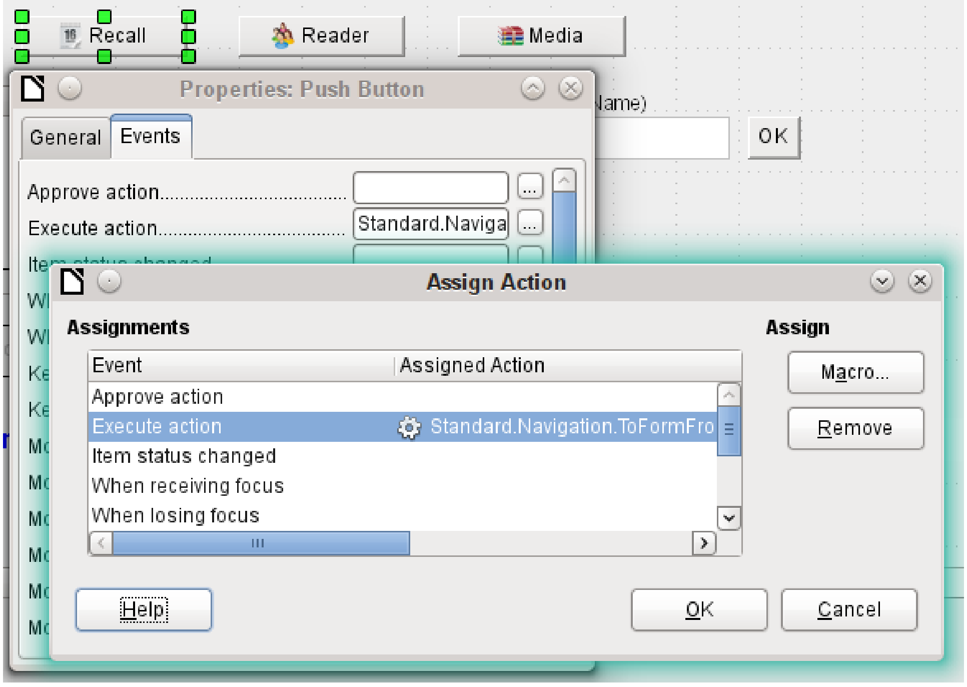
A single form window might well contain several forms with different data sources. In the Form Navigator, you always see first the term Forms, which in the case of a simple form contains only one subordinate entry.

Events within a form

All other macros are registered using the properties of subforms and controls through the Events tab.

1) Open the window for the properties of the control (if you have not already done so).
2) Choose a suitable event in the Events tab.

3) To edit the data source, use events that refer to *Record* or *Update* or *Reset*.
   - For buttons, or the choices within list or option fields the event *Execute action* would be the first port of call.
   - All other events depend on the type of control and the desired action.
4) Click the ... button to the right to open the Assign action dialog.
5) Click the **Macro** button to choose the macro defined for the action.
6) Click **OK** to confirm the assignment.



Approve action.......................................
Execute action...................................... Standard.Naviga

Event	Assigned Action
Approve action	
Execute action	Standard.Navigation.ToFormFro
Item status changed	
When receiving focus	
When losing focus	

Figure 427: Assign Action dialog

Components of macros

This section explains some of the macro language that is commonly used in Base, especially within forms. As far as is possible (and reasonable), examples are given in all the following sections.

The “Framework” of a macro

The definition of a macro begins with its type – **`Sub`** or **`Function`** – and ends with **`End Sub`** or **`End Function`**. A macro that is assigned to an event can receive arguments (values); the only useful one is the **`oEvent`** argument. All other routines that might be called by such a macro can be defined with or without a return value, depending on their purpose, and provided with arguments if necessary.

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

It is helpful to write out this framework immediately and put in the content afterwards. Do not forget to add comments to explain the macro, remembering the rule “As many as necessary, as few as possible”. In addition, Basic does not distinguish between upper and lower case. Usually fixed terms like **SUB** are written preferably in upper case, other concepts in mixed case.

Defining variables

In the next step, at the beginning of the routine, the **Dim** command is used to define the variables that will occur within the routine, each with its appropriate data type. Basic itself does not require this; it accepts any new variables that occur within the program. However the program code is “safer” if the variables, especially their data types, are declared. Many programmers make this a requirement, using Basic’s Explicit option when they begin to write a module. This means “Do not recognize any old variable, but only those I have declared beforehand”.

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

Only alphabetic characters (A-Z or a-z), numbers and the underline character ‘_’ may be used in variable names. No special characters are allowed. Spaces are allowed under some conditions, but are best avoided. The first character must be alphabetic.

It is common practice to specify the data type in the first character. Then it can be recognized wherever the variable occurs in the code. Also recommended are “expressive names”, so that the meaning of the variable is obvious from its name.

Note

You should make this more specific where necessary, as only one letter does not allow you to distinguish between the data types “Double” and “Date” or “Single” and “String”.

A list of possible data types in Star Basic can be found in Appendix A in this book. They differ in various places from the types in the database and in the LibreOffice API (Application Programming Interface). Such changes are made clear in the examples.

Defining arrays

For databases in particular, the assembly of several variables into a record is important. If several variables are stored together in a single common location, this is called an array. An array must be defined before data can be written into it.

```
Dim arData()
```

creates an empty array.

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

creates an array of a specific size (2 elements) and provides it with values.

Using

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

causes the two defined elements to be displayed onscreen. The element count begins with 0.

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

This creates an array in which three elements of any type can be stored, for example a record for "Lisa""Schmidt""Cologne". You cannot put more than three elements into this array. If you want to store more elements, you must make the array larger. However if the size of an array is redefined while a macro is running, the array is initially empty, just like a new array.

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

Adding **`Preserve`** keeps the preceding data so that the array is truly extended by the entry of the date (here in the form of text).

The array shown above can store only one record. If you want to store several records, as a table in a database does, you need to define a two-dimensional array.

```
Dim arData(2,1)
arData(0,0) = "Lisa"
arData(1,0) = "Schmidt"
arData(2,0) = "Cologne"
arData(0,1) = "Egon"
arData(1,1) = "Müller"
arData(2,1) = "Hamburg"
```

Here too it is possible to extend the previously defined array and preserve the existing contents by using **`Preserve`**.

Accessing forms

The form lies in the currently active document. The region which is represented here is called **`drawpage`**. The container in which all forms are kept is called **`forms`**; in the Form Navigator this shows up as the primary heading with all the individual forms attached. The variables named above receive their values like this:

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

The form to be accessed is called Filter. This is the name that is visible in the top level of the Form Navigator (by default the first form is called MainForm). Subforms lie in hierarchical order within the main form and can be reached step by step:

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

Instead of using intermediate varables, you can go straight to a particular form. An intermediate object, which can be used more than once, needs to be declared and assigned a separate value. In the following example, **`oSubForm`** is no longer used.

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

Note

If a name consists solely of ascii letters and numbers with no spaces or special characters, the name can be used directly in an assignment statement.

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

Contrary to normal Basic usage, such names must be written with the correct case.

A different mode of access to the form is provided by the event that triggers the macro.

If a macro is launched from a form event such as **Form Properties > Before record action**, the form itself can be reached as follows:

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

If the macro is launched from an event in a form control, such as **Text box > When losing focus**, both the form and the field become accessible:

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

Access to events has the advantage that you need not bother about whether you are dealing with a main form or a subform. Also the name of the form is of no importance to the functioning of the macro.

Accessing form elements

Elements within forms are accessed in a similar way: declare a suitable variable as **`object`** and search for the appropriate control within the form:

```
   Dim btnOK As Object  ' Button »OK»
   btnOK = oSubSubForm.getByName("button 1")   ' from the form 
readerdisplay
```

This method always works when you know which element the macro is supposed to work with. However when the first step is to determine which event launched the macro, the **oEvent** method shown above becomes useful. The variable is declared within the macro “framework” and gets assigned a value when the macro is launched. The **Source** property always yields the element that launched the macro, while the **Model** property describes the control in detail:

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

If you want, further actions can be carried out with the object obtained by this method.

Please note that subforms count as components of a form.

Access to the database

Normally access to the database is controlled by forms, queries, reports or the mailmerge function, as described in previous chapters. If these possibilities prove insufficient, a macro can specifically access the database in several ways.

Connecting to the database

The simplest method uses the same connection as the form. **`oForm`** is determined as shown above.

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

Or you can fetch the data source (i.e. the database) through the document and use its existing connection for the macro:

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

A further way allows the connection to the database to be created on the fly:

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

The **`If`** condition controls only one line so **`End If`** is not required.

If the macro is to be launched through the user interface and not from an event in a form, the following variant is suitable:

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

Access to databases outside the current database is possible as follows:

```
Dim oDatabaseContext As Object
Dim oDatasource As Object
Dim oConnection As Object
oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
oDatasource = oDatabaseContext.getByName("registered name of Database in
LO")
oConnection = oDatasource.GetConnection("","")
```

Connections to databases not registered with LibreOffice are also possible. In such cases, instead of the registered name, the path to the database must be given as `file:///……/database.odb`.

Expanded instructions on database connections are given in “Making a connection to a database” (page 492).

SQL commands

You work with the database using SQL commands. These need to be created and sent to the database; the result is determined according to the type of command and the results can be further processed. The **`createStatement`** directive creates a suitable object for this purpose.

```
Dim oSQL_Statement As Object ' the object that will carry out the SQL-
command
Dim stSql As String         ' Text of the actual SQL-command
Dim oResult As Object        ' result of executeQuery
Dim iResult As Integer        ' result of executeUpdate
oSQL_Statement = oConnection.createStatement()
```

To *query data*, you call the **`executeQuery`** method; the result is then evaluated. Table and field names are usually double-quoted. The macro must mask these with additional double quotes to ensure that they appear in the command.

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

To *modify data* – that is to **INSERT**, **UPDATE** or **DELETE** – or to influence the *database structure*, you call the **executeUpdate** method. Depending on the command and the database, this yields either nothing useful (a zero) or the number of records modified.

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

For the sake of completeness, there is one more special case to be mentioned: if the **`oSQL_Statement`** is to be used in different ways for **`SELECT`** or for other purposes, there is another method available, namely **`execute`**. We will not be using it here. For further information, see the API Reference.

Pre-prepared SQL commands with parameters

In all cases where manual entry by a user needs to be transferred into a SQL statement, it is easier and safer not to create the command as a long character string but to prepare it in advance and use it with parameters. This makes the formatting of numbers, dates, and strings easier (the constant double quotes disappear) and prevents malicious input from causing data loss.

To use this method, the object for a particular SQL command is created and prepared:

```
Dim oSQL_Statement As Object   ' the object, that executes the SQL-command
Dim stSql As String            ' Text of the actual SQL-command
stSql = "UPDATE author " _
     & "SET lastname = ?, firstname = ?" _
     & "WHERE ID = ?"
oSQL_Statement = oConnection.prepareStatement(stSql)
```

The object is created with **prepareStatement** so that the SQL command is known in advance. Each question mark indicates a position which later – before the command is executed – will receive an actual value. Because the command is prepared in advance, the database knows what type of entry – in this case two strings and a number – is expected. The various positions are distinguished by number (counting from 1).

Then the values are transferred with suitable statements and the SQL command is carried out. Here the values are taken from form controls, but they could also originate from other macros or be given as plain text:

```
oSQL_Statement.setString(1, oTextfeld1.Text)   ' Text for the surname
oSQL_Statement.setString(2, oTextfeld2.Text)   ' Text for the first name
oSQL_Statement.setLong(3, oZahlenfeld1.Value)  ' value for the appropriate
ID
iResult = oSQL_Statement.executeUpdate
```

The complete list of assignments is in “Parameters for prepared SQL commands" (page 449).

For further information on the advantages of this method, see below (external links):

- SQL injection (https://en.wikipedia.org/wiki/SQL_injection)
- Why use PreparedStatement in Java JDBC (https://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html)
- Structured Query Language (https://en.wikibooks.org/wiki/Structured_Query_Language)

Reading and using records

There are several ways, depending on requirements, to transfer information out of a database into a macro so that it can be processed further.

Please note: references to a form include subforms. What is intended is that form or part of a form that is bound to a particular data source.

Using forms

The current record and its data are always available through the form that shows the relevant data (table, query, SELECT). There are several **get**data_ type methods, such as:

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

All these methods require a column number from the data source; the count starts at 1.

Note

For all methods that work with databases, counting starts at 1. This applies to both columns and rows.

If you prefer to use column names instead of column numbers to work with the underlying data source (table, query, view), the column number can be determined using **findColumn**. Here is an example for finding the column called Name.

```
Dim sName As String
nName = oForm.findColumn("Name")
sName = oForm.getString(nName)
```

The type of value returned always matches the method type, but the following special cases should be noted:

- There are no methods for data of the types **Decimal**, **Currency** etc. which are used for commercially exact calculations. As Basic automatically carries out the appropriate conversion, you can use **getDouble**.
- When using **getBoolean**, you must take account of the way TRUE and FALSE are defined in the database. The usual definitions (logical values, 1 as TRUE) are processed correctly.
- Date values can be defined not only with the data type **Date**, but also (as above) as **util.Date**. This makes it easier to read and modify year, month and day.
- With whole numbers, beware of different data types. The above example uses **getLong**; the Basic variable ID must also have the data type **Long**, as this matches the **Integer** type in the database.

The complete list of these methods is to be found in “Editing rows of data" (page 446).

Tip

If values from a form are to be used directly for further processing in SQL (for example for input into another table), it is much simpler not to have to query the field type.

The following macro, which is bound to **Properties: Button > Events > Execute action** reads the first field in the form independently of the type necessary for future processing in Basic.

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

If fields are read using **getString()**, all formatting necessary for further SQL processing is preserved. A date that is displayed as 08.03.19 is read out in the format 2019-03-08 and can be used directly in SQL.

Reading out in a format corresponding to the type is only mandatory if the value is to be further processed within the macro, for example in a calculation.

Result of a query

The set of results from a query can be used in the same way. In the *SQL commands* section, you will find the variable **oResult** for this result set, which is usually read out something like this:

```
While oResult.next          ' one record after another
   REM transfer the result into variables
   stVar = oResult.getString(1)
   inVar = oResult.getLong(2)
   boVar = oResult.getBoolean(3)
   REM do something with these values
Wend
```

According to the type of SQL command, the expected result and its purpose, the **WHILE** loop can be shortened or dropped altogether. But basically a result set can always evaluated in this way.

If only the first record is to be evaluated

```
oResult.next
```

accesses the row for this record and with

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

reads the content of the first field. The loop ends here.

The query for the above example has text in the first column, an integer number in the second (**Integer** in the database corresponds to **Long** in Basic), and a Yes/No field in the third. The fields are accessed through a field index which, unlike an array index, starts from 1.

Navigation through such a result is not possible. Only single steps to the next record are allowed. To navigate within the record, the **ResultSetType** must be known when the query is created. This is accessed using

```
oSQL_Result.ResultSetType = 1004
```

or

```
oSQL_Result.ResultSetType = 1005
```

Type **1004 – SCROLL_INTENSIVE** allows you to navigate freely but does not pick up changes in the original data. Type **1005 – SCROLL_SENSITIVE** recognizes changes in the original data which might affect the query result.

The total number of rows in the result set can be determined only after a numeric type for the result has been specified. It is carried out as follows:

```
Dim iResult As Long
If oResult.last               ' go to the last record if possible
   iResult = oResult.getRow    ' the running number is the sum
Else
  iResult = 0
End If
```

Using a control

If a control is bound to a data source, the value can be read out directly, as described in the next section. However this can lead to problems. It is safer to use the procedure described in “Using forms” (page 433) or else the following method, which is shown for several different types of control:

```
sValue = oTextField.BoundField.Text          ' example for a Text field
nValue = oNumericField.BoundField.Value      ' example for a numeric field
dValue = oDateField.BoundField.Date          ' example for a date field
```

BoundField represents the link between the visible control and the actual content of the data set.

Navigating in a data set

In the last but one example the **`Next`** method was used to move from one row of the result set to the next. There are further similar methods and tests that can be used both for the data in a form – represented by the variable **`oForm`** – and for a result set. For example, using the method described in “Automatic updating of forms” (page 450), the previous record can be selected again:

```
Dim loRow As Long
loRow = oForm.getRow()      ' save the current row number
oForm.reload()              ' reload the record set
oForm.absolute(loRow)       ' go back to the same rowthe
```

The section “Automatic updating of forms” shows all the methods that are suitable for this.

Note

There is a bug which affects forms. It sets the current row number when data is altered within a form to ‘0’. See https://bugs.documentfoundation.org/show_bug.cgi?id=82591. To get the correct current row number, bind the following macro to the event **Form > Properties > Events > After record change**.

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

The new row number is read out and assigned to the global variable **`loRow`**. This variable is to be placed at the start of all modules and will retain its content until you exit Base or change the value by calling **`RowCounter`** again.

Editing records – adding, modifying, deleting

In order to edit records, several things have to work together:

- Information must be entered by the user into a control, using the keyboard.
- The data set behind the form must be informed about the change. This happens when you move out of the field into a new one.
- The database itself must be modified. This happens when you move from one record to another.

When you are doing this through a macro, these partial steps must all be considered. If any one of them is lacking or is carried out wrongly, changes will be lost and will not end up in the database. First of all the change must not be in the control’s displayed value but in the data set itself. This makes it pointless to change the **`Text`** property of a control.

Please note that tables are the only data sets that can be altered without causing problems. For other data sets, editing is possible only under special circumstances.

Changing the content of a control

If you wish to change only a single value, the **`BoundField`** property of the control can be used with an appropriate method. Then the change must be transmitted to the database. Here is an example for a date field into which the actual date is to be entered:

```
Dim unoDate As New com.sun.star.util.Date
unoDate.Year = Year(Date)
unoDate.Month = Month(Date)
unoDate.Day = Day(Date)
oDateField.BoundField.updateDate( unoDate )
oForm.updateRow()       ' the change is transmitted to the databset
```

For **BoundField** you use the **updateXxx** method that matches the data type of the field. In this example the field is a **Date** field. The new value is passed as the argument – in this case the current date, converted into the format which the macro requires.

Altering rows in a data set

The previous method is unsuitable when several values in a row need to be changed, For one thing, a control would have to exist on the form for every field, which is often not desired and not useful. Also, an object must be fetched for each field. The simple and direct way uses the form like this:

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

For each column in the data set, the **updateXxx** method appropriate to its type is called. The arguments are the column number (counting from 1) and the desired value. Then the alterations are passed on to the database.

Creating, modifying, and deleting rows

The named changes refer to the current row of the data set underlying the form. Under some circumstances it is necessary to call a method from “Navigating in a data set” (page 446). The following steps are necessary:

1) Choose the current record.
2) Change the values as described in the previous section.
3) Confirm the change with the following command:
   `oForm.updateRow()`
4) In special cases it is possible to cancel and return to the previous state:
   `oForm.cancelRowUpdates()`

For a new record there is a special method, comparable with changing to a new row in a table control. This is done as follows:

1) Prepare for a new record:
   `oForm.moveToInsertRow()`
2) Enter all wanted/required values. This is done using the **updateXxx** methods as shown in the previous section.
3) Confirm the new data with the following command:
   `oForm.insertRow()`
4) The new entry cannot be easily reversed. Instead you will have to delete the new record.

There is a simple command to delete a record; proceed as follows:

1) Choose the desired record and make it current, as for a modification.

2) Use the following command to delete it:
```
oForm.deleteRow()
```

Tip

To ensure that changes are carried over into the database, they must be confirmed explicitly with **`updateRow`** or **`insertRow`** as appropriate. While pressing the Save button will automatically use the appropriate function, with a macro you must determine before saving if the record is new (**`Insert`**) or a modification of an existing one (**`Update`**).

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

Testing and changing controls

Apart from the content of the data set, a lot more information can be read out of a control, edited and modified. This is particularly true of properties, as described in Chapter 4, Forms.

Several examples in “Improving usability” (page 450) use the additional information in the field:

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

As mentioned in the previous section, the **`Text`** property can only be modified usefully if the control is not bound to a data set. However there are other properties which are determined as part of the form definition but can be adapted at run time. For example, a label could be given a different text color if it represented a warning rather than information:

```
Sub showWarning(oField As Object, iType As Integer)
   Select Case iType
      Case 1
         oField.TextColor = RGB(0,0,255)   ' 1 = blue
      Case 2
         oField.TextColor = RGB(255,0,0)   ' 2 = red
      Case Else
         oField.TextColor = RGB(0,255,0)   ' 0 = green (neither 1 nor 2)
   End Select
End Sub
```

English names in macros

Whereas the designer of a form can use native language designations for properties and data access, only English names can be used in Basic. These are set out in the following synopsis.

Properties that are normally only set in the form definition are not included here. Nor are methods (functions and/or procedures) which are only rarely used or only required for more complex declarations.

The synopsis includes the following:

- Name – Name to be used for the property in macro code
- Data type – A Basic data type. For functions, the return type. Not included for procedures.
- R/W – Indicates how you can use the value:
  - R – read only
  - W – write (modify) only
  - (R) – Reading possible, not suitable for editing
  - (W) – Writing possible but not useful
  - R+W – suitable for reading and writing

Further information can be found in the API Reference by searching for the English name of the control. There is a useful tool called Xray for finding out which properties and methods are available for an element.

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

This launches the Xray extension for the argument.

Properties of forms and controls

The model of a control describes its properties. According to the situation, the value of a property can be accessed read-only or write-only. The order follows that in the lists of "Properties of Control Fields" in Chapter 4, Forms.

Font

In every control that shows text, the font properties can be customized.

Table 22: Font properties

Name	Data type	R/W	Property
FontName	string	R+W	Name of the font
FontHeight	single	R+W	Size of the font
FontWeight	single	R+W	Whether bold or normal
FontSlant	integer	R+W	Whether italic or roman
FontUnderline	integer	R+W	Whether underlined
FontStrikeout	integer	R+W	Whether struck through

Formula

English term: Form

Table 23: Formula properties

Name	Data type	R/W	Property
ApplyFilter	boolean	R+W	Filter applied
Filter	string	R+W	Current filter for the record
FetchSize	long	R+W	Number of records loaded at once
Row	long	R	Current row number
RowCount	long	R	Number of records

These properties apply to all controls

Control – see also FormComponent

Table 24: Properties applicable to all controls

Name	***Data type***	***R/W***	***Property***
Name	string	R+(W)	Name of the field
Enabled	boolean	R+W	Active: Field can be selected
EnableVisible	boolean	R+W	Field is displayed
ReadOnly	boolean	R+W	Field content cannot be changed
TabStop	boolean	R+W	Field can be reached through the Tab key
Align	integer	R+W	Horizontal alignment: 0 = left, 1 = centered, 2 = right
BackgroundColor	long	R+W	Background color
Tag	string	R+W	Additional information
HelpText	string	R+W	Help text as a Tooltip

These apply to many types of control

Table 25: Properties applicable to many controls

Name	***Data type***	***R/W***	***Property***
Text	string	(R+W)	Displayed content of the field. In text fields, this can be read and further processed, but that does not usually work for other types.
Spin	boolean	R+W	Spinbox incorporated in a formatted field.
TextColor	long	R+W	Text (foreground) color.
DataField	string	R	Name of the field in the Data set.
BoundField	object	R	Object representing the connection to the data set and providing access to the field content.

Text field – further properties (TextField)

Table 26: Further text field properties

Name	***Data type***	***R/W***	***Property***
String	string	R+W	Displayed field content.
MaxTextLen	integer	R+W	Maximum text length.
DefaultText	string	R+W	Default text.
MultiLine	boolean	R+W	Indicates if there is more than one line.
EchoChar	(integer)	R+W	Character displayed during password entry.

Numeric Field (NumericField)

Table 27: Numeric field properties

Name	***Data type***	***R/W***	***Property***
ValueMin	double	R+W	Minimum acceptable input value
ValueMax	double	R+W	Maximum acceptable input value
Value	double	R+(W)	Current value (Do not use for values from the data set).

Name	*Data type*	*R/W*	*Property*
ValueStep	double	R+W	Interval corresponding to one click for the mouse wheel or spinbox.
DefaultValue	double	R+W	Default value.
DecimalAccuracy	integer	R+W	Number of decimal places.
ShowThousandsSeparator	boolean	R+W	Show the locale separator for thousands.

Date field (DateField)

Date values are defined by the data type **`long`** and are displayed in ISO format: YYYYMMDD, for example 20190304 for 04 March 2019. To use this type with **`getDate`** and **`updateDate,`** and with the type **`com.sun.star.util.Date`**, see the examples.

Table 28: Date field properties

Name	*Data type*	*Datatype since LO 4.1.1*	*R/W*	*Property*
DateMin	long	com.sun.star .util.Date	R+W	Minimum acceptable entry date.
DateMax	long	com.sun.star .util.Date	R+W	Maximum acceptable entry date.
Date	long	com.sun.star .util.Date	R+(W)	Current value (Do not use for values from the data set).
DateFormat	integer		R+W	OS-specific date format: 0 = short Date (simple) 1 = short Date `dd.mm.yy` (2-digit year) 2 = short Date `dd.mm.yyyy` (4-digit year) 3 = long Date (includes day of the week and month name) Further possibilities can be found in the form definition or in the API Reference.
DefaultDate	long	com.sun.star .util.Date	R+W	Default value.
DropDown	boolean		R+W	Show a drop-down monthly calendar.

Time field (TimeField)

Time values are also of the type **`long`**.

Table 29: Time field properties

Name	*Data type*	*Data type from LO 4.1.1*	*R/W*	*Property*
TimeMin	long	com.sun.star .util.Time	R+W	Minimum acceptable entry value.

Name	Data type	Data type from LO 4.1.1	R/W	Property
TimeMax	long	com.sun.star .util.Time	R+W	Maximum acceptable entry value.
Time	long	com.sun.star .util.Time	R+(W)	Current value (Do not use for values from the data set).
TimeFormat	integer		R+W	Time format: 0 = short as `hh:mm` (hours, minutes, 24 hour clock) 1 = long as `hh:mm:ss` (same thing with Seconds, 24 hour clock) 2 = short as `hh:mm` (12 hour clock with AM/PM) 3 = long as `hh:mm:ss` (12 hour clock with AM/PM) 4 = short entry for a time duration 5 = long entry for a time duration
DefaultTime	long	com.sun.star .util.Time	R+W	Default value.

Currency field (CurrencyField)

A currency field is a numeric field with the following additional possibilities.

Table 30: Currency field properties

Name	Datatype	R/W	Property
CurrencySymbol	string	R+W	Currency symbol for display only.
PrependCurrencySymbol	boolean	R+W	Symbol is displayed before the number.

Formated field (FormattedControl)

A formatted control can be used as desired for numbers, currency or date/time. Very many of the properties already described apply here to but with different names.

Table 31: Formatted field properties

Name	Data type	L/S	Property
CurrentValue	variant	R	Current value of the contents. The actual data type depends on the contents and format.
EffectiveValue		R+(W)	
EffectiveMin	double	R+W	Minimum acceptable entry value.
EffectiveMax	double	R+W	Maximum acceptable entry value.
EffectiveDefault	variant	R+W	Default value.
FormatKey	long	R+(W)	Format for display and entry. There is no easy way to alter this using a macro.
EnforceFormat	boolean	R+W	Format is tested during entry. Only certain characters and combinations are allowed.

Listbox (ListBox)

Read and write access to the value lying behind the selected line is somewhat complicated but possible.

Table 32: List box properties

Name	***Data type***	***R/W***	***Property***
ListSource	array of string	R+W	Data source: Source of the list contents or name of the data set that provides the visible entry.
ListSourceType	integer	R+W	Type of data source: 0 = Value list 1 = Table 2 = Query 3 = Result set from a SQL command 4 = Result of a database command 5 = Field names from a database-table
StringItemList	array of string	R	List entries available for selection.
ItemCount	integer	R	Number of available list entries
ValueItemList	array of string	R	List of values to be passed from the form to the table.
DropDown	boolean	R+W	Drop-down list.
LineCount	integer	R+W	Total displayed lines when fully dropped down.
MultiSelection	boolean	R+W	Multiple selection intended.
SelectedItems	array of integer	R+W	List of selected entries as a list of positions in the overall entry list.

The first selected element from the list field is obtained like this:

```
oControl = oForm.getByName("Name of the Listbox")
sEintrag = oControl.ValueItemList( oControl.SelectedItems(0) )
```

Note

Since LibreOffice 4.1, the value passed to the database can be determined directly.

```
oControl = oForm.getByName("Name of the Listbox")
iD = oControl.getCurrentValue()
```

`getCurrentValue()` returns the value that will be stored in the database table. In listboxes this depends on the field to which they are bound (`BoundField`).

Up to and including LibreOffice 4.0, this function returned the displayed content, not the underlying value in the table.

Please note that the entry is an “array of string”, should the query for a list field be exchanged to restrict a selection option:

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

Combo boxes (ComboBox)

In spite of having similar functionality as listboxes, the properties of comboboxes are somewhat different. See the example “Comboboxes as listboxes with an entry option” on page 469.

Table 33: Combo box properties

Name	**Data type**	**R/W**	**Property**
Autocomplete	boolean	R+W	Fill automatically.
StringItemList	array of string	R+W	List entries available for use.
ItemCount	integer	R	Number of available list entries.
DropDown	boolean	R+W	Drop-down list.
LineCount	integer	R+W	Number of rows shown when dropped down.
Text	string	R+W	Currently displayed text.
DefaultText	string	R+W	Default entry.
ListSource	string	R+W	Name of the data source that provides the list entries.
ListSourceType	integer	R+W	Type of data source. Same possibilities as for listboxes (only the choice of Value list is ignored).

Checkboxes (CheckBox) and radio buttons (RadioButton)

Option Buttons can also be used.

Table 34: Check box and radio button properties

Name	**Data type**	**R/W**	**Property**
Label	string	R+W	Title (label)
State	short	R+W	Status 0 = not selected 1 = selected 2 = undefined
MultiLine	boolean	R+W	Line breaks for long text.

Pattern Field (PatternField)

In addition to the properties for simple text, the following are of interest:

Table 35: Pattern field properties

Name	***Data type***	***R/W***	***Property***
EditMask	string	R+W	Input mask.
LiteralMask	string	R+W	Character mask.
StrictFormat	boolean	R+W	Format testing during input.

Table control (GridControl)

Table 36: Table control properties

Name	***Data type***	***R/W***	***Property***
Count	long	R	Number of columns.
ElementNames	array of string	R	List of column names.
HasNavigationBar	boolean	R+W	Navigation bar available.
RowHeight	long	R+W	Row height.

FixedText – also called Label

Table 37: Fixed text / label properties

Name	***Data type***	***R/W***	***Property***
Label	string	R+W	Text displayed.
MultiLine	boolean	R+W	Line breaks for long text.

Group Boxes (GroupBox)

There are no properties for group boxes that are normally processed using macros. It is the status of the individual option fields that matters.

Buttons

CommandButton or ImageButton

Table 38: Command and image button properties

Name	***Data type***	***R/W***	***Property***
Label	string	R+W	Title – Label text.
State	short	R+W	Default state selected for toggling.
MultiLine	boolean	R+W	Line breaks for long text.
DefaultButton	boolean	R+W	Whether this is a default button.

Navigation bar (NavigationBar)

Further properties and methods associated with navigation – for example filters and changing the record pointer – are controlled using the form.

Table 39: Navigation bar properties

Name	***Data type***	***R/W***	***Property***
IconSize	short	R+W	Size of icons.
ShowPosition	boolean	R+W	Position can be entered and is displayed.
ShowNavigation	boolean	R+W	Allows navigation.
ShowRecordActions	boolean	R+W	Allows record actions.
ShowFilterSort	boolean	R+W	Allows filter sorting.

Methods for forms and controls

The data type of the parameter is indicated by an abbreviation:

- column number for the desired field in the data set, counting from 1
- numerical value – could be either an integer or a decimal number
- s – String; maximum length depends on the table definition.
- b – Boolean (logical) – true or false
- d – Date value

Navigating in a data set

These methods work both in forms and in the results set from a query.

“Cursor” in the description means the record pointer.

Table 40: Navigation methods: Testing for the position of the cursor

Name	***Data type***	***Description***
isBeforeFirst	boolean	The cursor is before the first record. This is the case if it has not yet been reset after entry.
isFirst	boolean	Shows if the cursor is on the first entry.
isLast	boolean	Shows if the cursor is on the last entry.
isAfterLast	boolean	The cursor is after the last row when it is moved on with next.
getRow	long	Current row number.

Table 41: Navigation methods: Setting the cursor
For boolean data types, True means that the navigation was successful.

Name	***Data type***	***Description***
beforeFirst	–	Moves before the first row.
first	boolean	Moves to the first row.
previous	boolean	Goes back one row.
next	boolean	Goes forward one row.
last	boolean	Goes to the last record.
afterLast	–	Goes after the last record.
absolute(n)	boolean	Goes to the row with the given row number.
relative(n)	boolean	Goes backwards or forwards by the given amount: forwards for positive and backwards for negative arguments.

Table 42: Navigation methods: Methods affecting the current record status

Name	***Data type***	***Description***
refreshRow	–	Reads the original values for the row back in.
rowInserted	boolean	Indicates if this is a new row.
rowUpdated	boolean	Indicates if the current row has been altered.
rowDeleted	boolean	Indicates if the current row has been deleted.

Editing rows of data

The methods used for reading are available for any form or data set. Methods for alteration and storage can be used only for editable data sets (usually tables, not queries).

Table 43: Methods for editing rows of data: Methods for the whole row

Name	**Data type**	**Description**
insertRow	–	Saves a new row.
updateRow	–	Confirms alteration of the current row.
deleteRow	–	Deletes the current row.
cancelRowUpdates	–	Reverses changes in the current row.
moveToInsertRow	–	Moves the cursor into a row corresponding to a new record.
moveToCurrentRow	–	After the entry of a new record, returns the cursor to its previous position.

Table 44: Methods for editing rows of data: Reading values

Name	**Data type**	**Description**
getString(c)	string	Gives the content of the column as a character string.
getBoolean(c)	boolean	Gives the content of the column as a boolean value.
getByte(c)	byte	Gives the content of the column as a single byte.
getShort(c)	short	Gives the content of the column as an integer.
getInt(c)	integer	Gives the content of the column as an integer.
getLong(c)	long	Gives the content of the column as an integer.
getFloat(c)	float	Gives the content of the column as a single precision decimal number.
getDouble(c)	double	Gives the content of the column as a double precision decimal number. The automatic conversions carried out by Basic makes this a suitable type for decimal and currency fields.
getBytes(c)	array of bytes	Gives the content of the column as an array of single bytes.
getDate(c)	Date	Gives the content of the column as a date.
getTime(c)	Time	Gives the content of the column as a time value.
getTimestamp(c)	DateTime	Gives the content of the column as a timestamp (date and time).
wasNull	boolean	Indicates if the value of the most recently read column was NULL.

Note

In Basic itself, date and time values are both given the type DATE. To access dates in data sets, various types are available: **com.sun.star.util.Date** for a date, **com.sun.star.util.Time** for a time, and **com.sun.star.util.DateTime** for a timestamp.

Table 45: Methods for saving values

Name	**Data type**	**Description**
updateNull(c)	–	Sets the column content to NULL.

updateBoolean(c,b)	–	Changes the content of column c to the logical value b.
updateByte(c,x)	–	Stores byte x in column c.
updateShort(c,n)	–	Stores the integer n in column c.
updateInt(c,n)	–	Stores the integer n in column c.
updateLong(c,n)	–	Stores the integer n in column c.
updateFloat(c,n)	–	Stores the decimal number n in column c.
updateDouble(c,n)	–	Stores the decimal number n in column c.
updateString(c,s)	–	Stores the string s in column c.
updateBytes(c,x)	–	Stores the byte array x in column c.
updateDate(c,d)	–	Stores the date d in column c.
updateTime(c,d)	–	Stores the time d in column c.
updateTimestamp(c,d)	–	Stores the timestamp d in column c.

Editing individual values

This method uses the **`BoundField`** property of a control to read or modify the content of the corresponding column. It corresponds almost exactly to the method described in the previous section, except that the column number is not given.

Table 46: Methods for editing individual values: reading values

Name	***Data type***	***Description***
getString	string	Gives the content of the field as a character string.
getBoolean	boolean	Gives the content of the field as a logical value.
getByte	byte	Gives the content of the field as a single byte.
getShort	short	Gives the content of the field as an integer.
getInt	integer	Gives the content of the field as an integer.
getLong	long	Gives the content of the field as an integer.
getFloat	float	Gives the content of the field as a single-precision decimal value.
getDouble	double	Gives the content of the field as a double-precision decimal number. The automatic conversions carried out by Basic makes this a suitable type for decimal and currency fields.
getBytes	array of bytes	Gives the content of the field as an array of bytes.
getDate	Date	Gives the content of the field as a date.
getTime	Time	Gives the content of the field as a time.
getTimestamp	DateTime	Gives the content of the field as a timestamp.
wasNull	boolean	Indicates if the value of the most recently read column was NULL.

Note

In Basic itself, date and time values are both given the type DATE. To access dates in data sets, various types are available: **com.sun.star.util.Date** for a date, **com.sun.star.util.Time** for a time, and **com.sun.star.util.DateTime** for a timestamp.

Table 47: Methods for editing individual values: storing values

Name	***Data type***	***Description***
updateNull	–	Sets the content of the column to NULL.
updateBoolean(b)	–	Sets the content of the column to the logical value b.
updateByte(x)	–	Stores the byte x in the column.
updateShort(n)	–	Stores the integer n in the column.
updateInt(n)	–	Stores the integer n in the column.
updateLong(n)	–	Stores the integer n in the column.
updateFloat(n)	–	Stores the decimal number n in the column.
updateDouble(n)	–	Stores the decimal number n in the column.
updateString(s)	–	Stores the character string s in the column.
updateBytes(x)	–	Stores the byte array x in the column.
updateDate(d)	–	Stores the date d in the column.
updateTime(d)	–	Stores the time d in the column.
updateTimestamp(d)	–	Stores the timestamp d in the column.

Parameters for prepared SQL commands

The methods which transfer the value of a pre-prepared SQL command (see “Pre-prepared SQL commands with parameters” on page 432) are similar to those in the previous section. The first parameter (denoted by I) is a numbered position within the SQL command.

Table 48: Methods for transferring the value of a pre-prepared SQL command

Name	***Data type***	***Description***
setNull(i, n)	–	Sets the content of the column to NULL. N is the SQL data type as given in the API Reference.
setBoolean(i, b)	–	Puts the given logical value b into the SQL command.
setByte(i, x)	–	Puts the given byte x into the SQL command.
setShort(i, n)	–	Puts the given integer n into the SQL command.
setInt(i, n)	–	Puts the given integer n into the SQL command.
setLong(i, n)	–	Puts the given integer n into the SQL command.
setFloat(i, n)	–	Puts the given decimal number into the SQL command.
setDouble(i, n)		
setString(i, s)	–	Puts the given character string into the SQL command.
setBytes(i, x)	–	Puts the given byte array x into the SQL command.

Name	***Data type***	***Description***
setDate(i, d)	–	Puts the given date d into the SQL command.
setTime(i, d)	–	Puts the given time d into the SQL command.
setTimestamp(i, d)	–	Puts the given timestamp d into the SQL command.
clearParameters	–	Removes the previous values of all parameters from a SQL command.

Improving usability

For this first category of macro use, we show various possibilities for improving the usability of Base forms.

Automatic updating of forms

Often something is altered in a form and this alteration is required to appear in a second form on the same page. The following code snippet calls the reload method on the second form, causing it to refresh.

```
Sub Update
```

First the macro is named. The default designation for a macro is **Sub**. This may be written in upper or lower case. **Sub** allows a subroutine to run without returning a value. Further down by contrast a function is described, which does return a value.

The macro has the name Update. You do not need to declare variables because LibreOffice Basic automatically creates variables when they are used. If you misspell a variable, LibreOffice Basic silently creates a new variable without complaint. Use **Option Explicit** To prevent LibreOffice Basic from automatically creating variables; this is recommended by most programmers.

Therefore we usually start by declaring variables. All the variables declared here are objects (not numbers or text), so we add **As Object** to the end of the declaration. To remind us later of the type of the variables, we preface their names with an "o". In principle, though, you can choose almost any variable names you like.

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

The form lies in the currently active document. The container, in which all forms are stored, is named **drawpage**. In the form navigator this is the top-level concept, to which all the forms are subsidiary.

In this example, the form to be accessed is named Display. Display is the name visible in the form navigator. So, for example, the first form by default is called Form1.

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

Since the form has now been made accessible and the point at which it can be accessed is saved in the variable **oForm**, it is now reloaded (refreshed) with the **reload()** command.

```
   oForm.reload()
```

```
End Sub
```

The subroutine begins with **SUB** so it must end with **End Sub**.

This macro can now be selected to run when another form is saved. For example, on a cash register (till), if the total number of items sold and their stock numbers (read by a barcode scanner) are entered into one form, another form in the same open window can show the names of all the items, and the total cost, as soon as the form is saved.

Filtering records

The filter itself can function perfectly well in the form described in Chapter 8, Database Tasks. The variant shown below replaces the Save button and reads the listboxes again, so that a chosen filter from one listbox can restrict the choices available in the other listbox.

✓ **Note**

See also the database Example_Search_and_Filter.odb associated with this book.

```
Sub Filter
   Dim oDoc As Object
   Dim oDrawpage As Object
   Dim oForm1 As Object
   Dim oForm2 As Object
   Dim oFieldList1 As Object
   Dim oFieldList2 As Object
   oDoc = thisComponent
   oDrawpage = oDoc.drawpage
```

First the variables are defined and set to access the set of forms. This set comprises the two forms "Filter" and "Display". The listboxes are in the "Filter" form and have the names "List_1" and "List_2".

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

First the contents of the listboxes are transferred to the underlying form using **commit()**. The transfer is necessary, because otherwise the change in a listbox will not be recognized when saving. The **commit() instruction** need only be applied to the listbox that has just been accessed. After that the record is saved using **updateRow()**. In principle, our filter table contains only one record, which is written once at the beginning. This record is therefore overwritten continuously using an update command.

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

The listboxes are meant to influence each other. For example, if one listbox is used to restrict displayed media to CDs, the other listbox should not include all the writers of books in its list of authors. A selection in the second listbox would then all too often result in an empty filter. That is why the listboxes must be read again. Strictly speaking, the **refresh()** command only needs to be carried out on the listbox that has not been accessed.

After this, form2, which should display the filtered content, is read in again.

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

Listboxes that are to be influenced using this method can be supplied with content using various queries.

The simplest variant is to have the listbox take its content from the filter results. Then a single filter determines which data content will be further filtered.

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

The field content and the number of hits is displayed. To get the number of hits, a sub-query is used. This is necessary as otherwise only the number of hits, without further information from the field, will be shown in the listbox.

The macro creates listboxes quite quickly by this action; they are filled with only one value. If a listbox is not NULL, it is taken into account by the filtering. After activation of the second listbox, only the empty fields and one displayed value are available to both listboxes. That may seem practical for a limited search. But what if a library catalog shows clearly the classification for an item, but does not show uniquely if this is a book, a CD or a DVD? If the classification is chosen first and the second listbox is then set to "CD", it must be reset to NULL in order to carry out a subsequent search that includes books. It would be more practical if the second listbox showed directly the various media types available, with the corresponding hit counts.

To achieve this aim, the following query is constructed, which is no longer fed directly from the filter results. The number of hits must be obtained in a different way.

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

This very complex query can be broken down. In practice it is common to use a **VIEW** for the sub-query. The listbox receives its content from a query relating to this **VIEW.**

The query in detail: The query presents two columns. The first column contains the view seen by a person who has the form open. This view shows the content of the field and, separated by a hyphen, the hits for this field content. The second column transfers its content to the underlying table of the form. Here we have only the content of the field. The listboxes thus draw their content from the query, which is presented as the filter result in the form. Only these fields are available for further filtering.

The table from which this information is drawn is actually a query. In this query the primary key fields are counted (**SELECT COUNT("ID") AS "Count"**). This is then grouped by the search term in the field (**GROUP BY "Field_1"**). This query presents the term in the field itself as the second column. This query in turn is based on a further sub-query:

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

This sub-query deals with the other field to be filtered. In principle, this other field must also match the primary key. If there are further filters, this query can be extended:

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

This allows any further fields that are to be filtered to control what finally appears in the listbox of the first field, "Field_1".

Finally the whole query is sorted by the underlying field.

What the final query underlying the displayed form, actually looks like, can be seen from Chapter 8, Database Tasks.

The following macro can control through a listbox which listboxes must be saved and which must be read in again.

The following subroutine assumes that the Additional Information property for each listbox contains a comma-separated list of all listbox names with no spaces. The first name in the list must be the name of that listbox.

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

An array (a collection of data accessible via an index number) is established and filled with the field names of the listboxes. The first name in the list is the name of the listbox linked to the event.

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

The array is run through from its lower bound (**'Lbound()'**) to its upper bound (**'Ubound()'**) in a single loop. All values which were separated by commas in the additional information, are now transferred successively.

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

The listbox that triggered the macro must be saved. It is found in the variable **aList(0)**. First the information for the listbox is carried across to the underlying table, and then the record is saved.

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

The other listboxes must be refreshed, as they now contain different values depending on the first listbox.

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

The queries for this more usable macro are naturally the same as those already presented in the previous section.

Preparing data from text fields to fit SQL conventions

When data is stored in a SQL command, apostrophes in names such as “O’Connor” can cause problems. This is because single quotes (**'') are used to enclose** text that is to be entered into records. In such cases, we need an intermediate function to prepare the data appropriately.

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

Note that this is a function, not a sub. A function takes a value as argument and then returns a value.

The text to be transferred is first searched to see if it contains an apostrophe. If this is the case, the text is split at this point – the apostrophe itself is the delimiter for the split – and joined together again with two apostrophes. This masks the SQL code. The function yields its result through the following call:

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

This simply means that the variable stTextold is reworked and the result stored in stTextnew. The two variables do not actually need to have different names. The call can be done with:

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

This function is used repeatedly in the following macros so that apostrophes can also be stored using SQL commands.

Calculating values in a form in advance

Values which can be calculated using database functions are not stored separately in the database. The calculation takes place not during the entry into the form but after the record has been saved. If the form consists only of a single table control, this makes little difference. The calculated value can be read out immediately after data entry. But when forms have a set of different individual fields, the previous record may not be visible. In such cases it makes sense for the values that are otherwise calculated inside the database to be shown in the appropriate fields.

Note

See the database Example_direct_Calculation_Form.odb associated with this book.

The following three macros show how such a thing can be done in principle. Both macros are linked to the exit from the particular field. This also allows for the fact that the value in an existing field might subsequently be changed.

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

If a value is entered into the price field, the macro is launched on leaving that field. In the same form as the price field is a field called price_without_tax. For this field **`BoundField.UpdateDouble`** is used to calculate the price without VAT. The data field is derived from a query which in principle carries out the same calculation but using saved data. In this way the calculated value is visible during data entry and also later during navigation through the record without being stored.

If the quantity field contains a value, a further calculation is carried out for the fields bound to it.

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

This short procedure serves only to transmit the solution of the following procedure when leaving the quantity field on the form.

```
Sub Calculation_Total2(oFeld As Object)
```

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

This procedure is merely a way of affecting several fields at once. The procedure is launched from one field quantity, which contains the number of items bought. Using this field and the price field, the total and tax_total are calculated and transferred to the appropriate fields.

These procedures and queries have one shortcoming: the rate of VAT is effectively hard-coded into the program. It would be better to use an argument for this, related to the price, since VAT might vary and not be the same for all products. In such cases the appropriate value for VAT would need to read out of a form field.

Providing the current LibreOffice version

LibreOffice version 4.1 brought some changes to listfields and date values that make it necessary to determine the current version when executing macros in these areas. The following code serves this purpose:

```
Function OfficeVersion()
   Dim aSettings, aConfigProvider
   Dim aParams2(0) As New com.sun.star.beans.PropertyValue
   Dim sProvider$, sAccess$
   sProvider = "com.sun.star.configuration.ConfigurationProvider"
   sAccess = "com.sun.star.configuration.ConfigurationAccess"
   aConfigProvider = createUnoService(sProvider)
   aParams2(0).Name = "nodepath"
   aParams2(0).Value = "/org.openoffice.Setup/Product"
   aSettings = aConfigProvider.createInstanceWithArguments(sAccess, 
aParams2())
   OfficeVersion() = 
Array(aSettings.ooName,aSettings.ooSetupVersionAboutBox)
End Function
```

This function returns an array in which the first element is LibreOffice and the second is the full version number, for example 4.1.5.2.

Returning the value of listfields

Since LibreOffice 4.1, the value returned by a listbox to the database is stored in CurrentValue. This was not the case in previous versions, nor in OpenOffice or Apache OpenOffice. The following function will do the calculation. The LibreOffice version must be checked to see if it is later than LibreOffice 4.0.

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

Before LibreOffice 4.1, the value that was passed on was read out of the listbox’s value list. The visibly chosen record is SelectedItems(0). '0' because several additional values could be selected in a listbox.

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

-1 is a value that is not used as an AutoValue and therefore will not exist in most tables as a foreign key.

```
      ID_Determination = -1
   Else
      ID_Determination = Cint(stContent)
```

Convert to integer

```
   End If
End Function
```

The function transmits the value as an integer. Most primary keys are automatically incrementing integers. When a foreign key does not satisfy this criterion, the return value must be adjusted to the appropriate type.

The displayed value of a listfield can be further determined using the field’s view property.

```
Sub Listfielddisplay
   Dim oDoc As Object
   Dim oForm As Object
   Dim oListbox As Object
   Dim oController As Object
   Dim oView As Object
   oDoc = thisComponent
   oForm = oDoc.Drawpage.Forms(0)
   oListbox = oForm.getByName("Listbox")
   oController = oDoc.getCurrentController()
   oView = oController.getControl(oListbox)
   print "Displayed content: " & oView.SelectedItem
```

```
End Sub
```

The controller is used to access the view of the form. This determines what appears in the visual interface. The selected value is **`SelectedItem`**.

Limiting listboxes by entering initial letters

It can sometimes happen that the content of listboxes grows too big to handle. To make searching faster in such cases, it is useful to limit the content of the listbox to the values indicated by entering one or more initial characters. The listbox itself is provided with a SQL command that serves as a placeholder. This could be:

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

This prevents Base from having to read a huge list of values when the form is opened.

The following macro is linked to **Properties: Listbox > Events > Key released**.

```
Global stListStart As String
Global lTime As Long
```

First, global variables are created. These variables are necessary to enable searching not only for a single letter but also, after further keys have been pressed, for combinations of letters.

The letters entered are stored sequentially in the global variable **`stListStart`**.

The global variable **`lTime`** is used to store the current time in seconds. If there is a long pause between keystrokes, the **`stListStart`** variable should be reset. For this reason, the time difference between successive entries is queried.

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

The macro is launched by a keystroke. Within the API, each key has a numeric code which can be looked up in the *com::sun::star::awt::Key* constant group reference section of the API Reference. Special characters like ä, ö, and ü have the KeyCode 0. All other letters and numbers have a KeyCode less than 538.

It is important to check the **`KeyCode`** because hitting the Tab key to move to another field will also launch the macro. The **`KeyCode`** for the Tab key is 1282, so any further code in the macro will not be executed.

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

The SQL code for the listbox is stored in an array. However, SQL commands count as single data elements, so the array is dimensioned as **`stSql(0)`**.

When reading SQL code out of the listbox, please note that the SQL code is not directly accessible as text. Instead the code is available as a single array element: **`oField.ListSource(0)`**.

After declaring variables for future use, the SQL command is split up. To get the field which is to be filtered, we split the code at the first comma. The field must therefore be placed first in the command. Then this code is split again at the first double quote character, which introduces the fieldname. Here this is done using simple arrays. The **`stField`** variable needs to have the quotation marks put back at the beginning. In addition **`Rtrim`** is used to prevent any space from occurring at the end of the expression.

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

A sort instruction is expected next in the SQL code. However commands in SQL can be in upper, lower or mixed case, so the **`inStr`** function is used instead of **`Split`** to find the ORDER character string. The last parameter for this function is 1, indicating that the search should be case-insensitive. Everything to the left of the ORDER string is to be used for constructing the new SQL code. This ensures that the code can also serve listfields which come from different tables or have been defined in SQL code using further conditions.

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

If the query contains the term LOWER, it means that it was created using this **`ListFilter`** procedure. Therefore in constructing the new query, we need go only as far as this position.

If this is not the case, and the query already contains the term WHERE (in upper or lower case), any further conditions to the query need to be prepended with **`AND`**.

If neither condition is fulfilled, a **`WHERE`** is attached to the existing code.

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

If a time value has been stored in the global variable, and the difference between this and the current time is less than 5 seconds, the entered letter is joined onto the previous one. Otherwise the letter is treated as a new single-letter entry. The listfield will then be re-filtered according to this entry. After this, the current time is stored in **`lTime`**.

```
      stText = LCase( stListStart & "%")
      stSql(0) = stQuery + "LOWER("+stField+") LIKE '"+stText+"' ORDER BY
"+stField+""
      oFeld.ListSource = stSql
      oField.refresh
   End If
End Sub
```

The SQL code is finally put together. The lower-case version of the field content is compared with the lower-case version of the entered letter(s). The code is inserted into the listbox and the field updated so that only the filtered content can be looked up.

Converting dates from a form into a date variable

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

Here all LibreOffice versions from 4.1 onward are intercepted. For this purpose, the version number is split into its individual elements, and the major and minor release numbers are checked. This will work up to LibreOffice 9.

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

Since LibreOffice 4.1.2, dates have been stored as arrays within form controls. This means that the current value of the control cannot be used to access the date itself. The date needs to be recreated from the day, month and year if it is to be used further in macros.

Searching data records

You can search database records without using a macro. However, the corresponding query that must be set up can be very complicated. A macro can solve this problem with a loop.

The following subroutine reads the fields in a table, creates a query internally, and finally writes a list of primary key numbers of records in the table that are retrieved by this search term. In the following description, there is a table called Searchtmp, which consists of an auto-incrementing primary key field (ID) and a field called Nr. that contains all the primary keys retrieved from the table being searched. The table name is supplied initially to the subroutine as a variable.

To get a correct result, the table must contain the content you are searching for as text and not as foreign keys. If necessary, you can create a VIEW for the macro to use.

Note

See the database Example_Search_and_Filter.odb associated with this book.

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

The content of the search text field is initially converted into lower case so that the subsequent search function need only compare lower case spellings.

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

First it must be determined if a search term has actually been entered. If the field is empty, it will be assumed that no search is required. All records will be displayed without further searching.

If a search term has been entered, the column names are read from the table being searched, so that the query can access the fields.

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

Note

SQL formulas in macros must first be placed in double quotes like normal character strings. Field names and table names are already in double quotes inside the SQL formula. To create final code that transmits the double quotes properly, field names and table names must be given two sets of these quotes.

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

becomes, when displayed with the command `MsgBox stSql`,
`SELECT "Name" FROM "Table";`

The index of the array, in which the field names are written is initially set to 0. Then the query begins to be read out. As the size of the array is unknown, it must be adjusted continuously. That is why the loop begins with **'ReDim Preserve arContent(inI)'** to set the size of the array and at the same time to preserve its existing contents. Next the fields are read and the array index incremented by 1. Then the array is dimensioned again and a further value can be stored.

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

Now the query is put together within a loop and subsequently applied to the table defined at the beginning. All case combinations are allowed for, since the content of the field in the query is converted to lower case.

The query is constructed such that the results end up in the "searchtmp" table. It is assumed that the primary key is the first field in the table (**arContent(0)**).

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

The display form must be reloaded. Its data source is a query, in this example Searchquery.

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

This creates a table that is to be evaluated by the query. As far as possible, the query should be constructed so that it can subsequently be edited. A sample query is shown:

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

All elements of the **searchtable** are included, including the primary key. No other table appears in the direct query; therefore no primary key from another table is needed and the query result remains editable.

The primary key is saved in this example under the name **Nr.** The macro reads precisely this field. There is an initial check to see if the content of the **Nr.** field appears in the **searchtmp table**. The **IN** operator is compatible with multiple values. The sub-query can also yield several records.

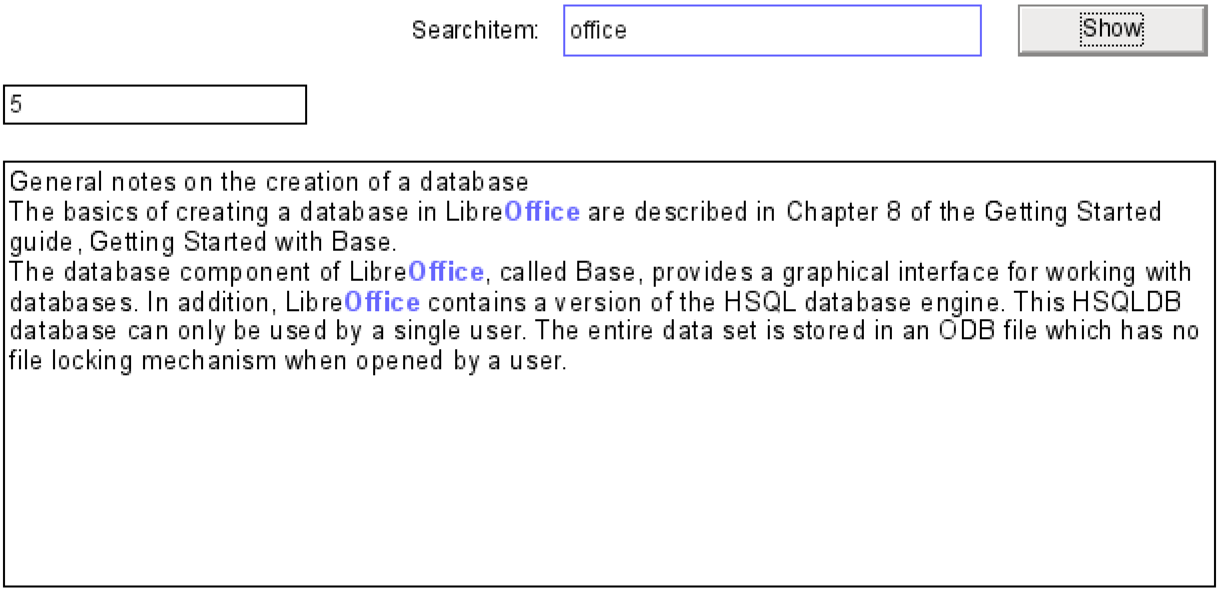
For larger amounts of data, value matching by using the **IN** operator quickly slows down. Therefore it is not a good idea to use an empty search field simply to transfer all primary key fields from **searchtable** into the **searchtmp** table and then view the data in the same way. Instead an empty search field creates an empty **searchtmp** table, so that no records are available. This is the purpose of the second half of the condition:

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

If a record is found in the Searchtmp table, it means that the result of the first query is greater than 0. In this case: **"Nr." = '-1'** (here we need a number which cannot occur as a primary key, so **'-1'is a good value**). If the query yields precisely 0 (which will be the case if no records are present), then **"Nr." = "Nr."**. This will list every record which has a **Nr.** As **Nr.** is the primary key, this means all records.

Highlighting search terms in forms and results

With a large text field, it is often unclear where matches to a search term occur. It would be nice if the form could highlight the matches. It should look something like that shown in Figure 428.



General notes on the creation of a database
The basics of creating a database in LibreOffice are described in Chapter 8 of the Getting Started guide, Getting Started with Base.
The database component of LibreOffice, called Base, provides a graphical interface for working with databases. In addition, LibreOffice contains a version of the HSQL database engine. This HSQLDB database can only be used by a single user. The entire data set is stored in an ODB file which has no file locking mechanism when opened by a user.

Figure 428: Form to highlight search matches

To get a form to work like this, we need a couple of extra items in our box of tricks.

Note

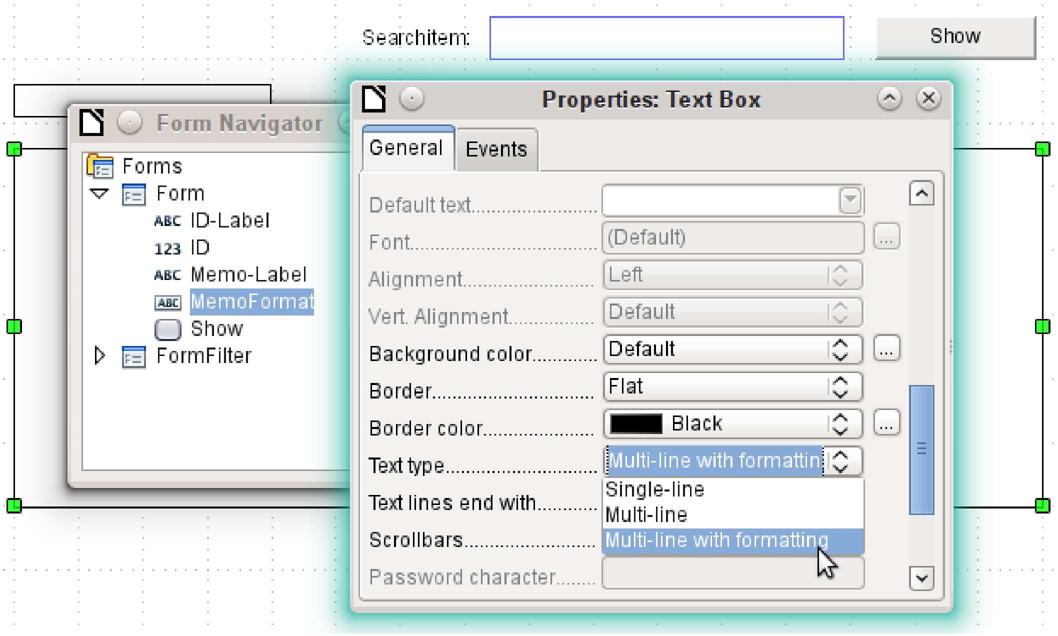
See the database Example_Autotext_Searchmarkin_Spelling.odb associated with this book.

The operation of a search field like this has already been explained. A filter table is created and a form is used to write the current values of a single record into this table. The main form is provided with its content using a query which looks like this:

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

When search text is entered, all records in the table “Table” that have the search text in the “memo” field are displayed. The search is not case-sensitive.

If no search text is entered, all the records in the table are displayed. As the primary key of this table is included in the query, the latter can be edited.



Default text
Font (Default)
Alignment Left
Vert. Alignment Default
Background color Default
Border Flat
Border color Black
Text type Multi-line with formattin
Text lines end with
Scrollbars
Password character

Single-line
Multi-line
Multi-line with formatting

Figure 429: Setting the text type to multi-line with formatting

In the form, in addition to the ID field for the primary key, there is a field called MemoFormat which has been configured (using **Properties > General > Text type > Multi-line with formatting**) to show colored as well as black text. Careful consideration of the properties of the text field reveals that the Data tab has now disappeared. This is because data cannot be entered into a field that has additional formatting which the database itself cannot store. Nevertheless, it is still possible to get text into this field, to mark it up, and to transfer it out after an update by using a macro.

The ContentRead procedure serves to transfer the content of the database field “memo” into the formatted text field MemoFormat, and to format it so that any text corresponding to that in the search field will be highlighted.

The procedure is bound to **Form > Events > After record change**.

```
Sub ContentRead(oEvent As Object)
   Dim inMemo As Integer
   Dim oField As Object
   Dim stSearchtext As String
   Dim oCursor As Object
   Dim inSearch As Integer
   Dim inSearchOld As Integer
   Dim inLen As Integer
   oForm = oEvent.Source
   inMemo = oForm.findColumn("memo")
   oField = oForm.getByName("MemoFormat")
   oField.Text = oForm.getString(inMemo)
```

First the variables are defined. Then the table field “memo” is searched from the form and the **getString**() function is used to read the text from the numbered column. This is transferred into the field which can be formatted but which has no link to the database: MemoFormat.

Initial tests showed that the form opened but the form toolbar at the bottom was no longer created. Therefore a very short wait of 5/1000 seconds was built in. After this the displayed content is read out of the FormFilter (which is parallel to the Form in the forms hierarchy).

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

To be able to format text, an (invisible) **TextCursor** must be created in the field that contains the text. The default display of the text uses a 12-point serif font which may not occur in other parts of the form and cannot be directly customized using the form control properties. In this procedure, the text is set to the desired appearance right at the beginning. If this is not done, differences in formatting can cause the upper boundary of the text in the field to be cut off. In early tests, only 2/3 of the first line was legible.

In order for the invisible cursor to mark the text, It is set initially to the beginning of the field and then to the end. The argument in both cases is **true**. Next come the specifications for font size, font face, color, and weight. Then the cursor is set back to the beginning again.

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

If there is text in the field and an entry has been made requesting a search, this text is now searched to find the search string. The outer loop asks first if these conditions are met; the inner one establishes if the search string is really in the text in the MemoFormat field. These settings could actually be omitted, since the query on which the form is based only displays text that fulfills these conditions.

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

The text is searched for the search string. This takes place in a loop which ends when no further matches are displayed. **InStr**() returns the location of the first character of the search string in the specified display format, independent of case. The loop is controlled by the requirement that at the end of each cycle, the start of **inSearch** has been incremented by 1 (-1 in the first line of the loop and +2 in the last line). For each cycle, the cursor is moved to the initial position without marking using **oCursor.goRight(Position, false)**, and then to the right with marking by the length of the search string. Then the desired formatting (blue and somewhat bolder) is applied and the cursor moved back to its next starting point for the next run.

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

The ContentWrite procedure serves to transfer the content of the formattable text field MemoFormat into the database. This proceeds independently of whether any alteration takes place.

The procedure is bound to **Form > Events > Before record change**.

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

The trigger event is implemented twice. Only the implementation name which ends with OdatabaseForm gives the correct access to the record (implementations are explained on page 488).

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

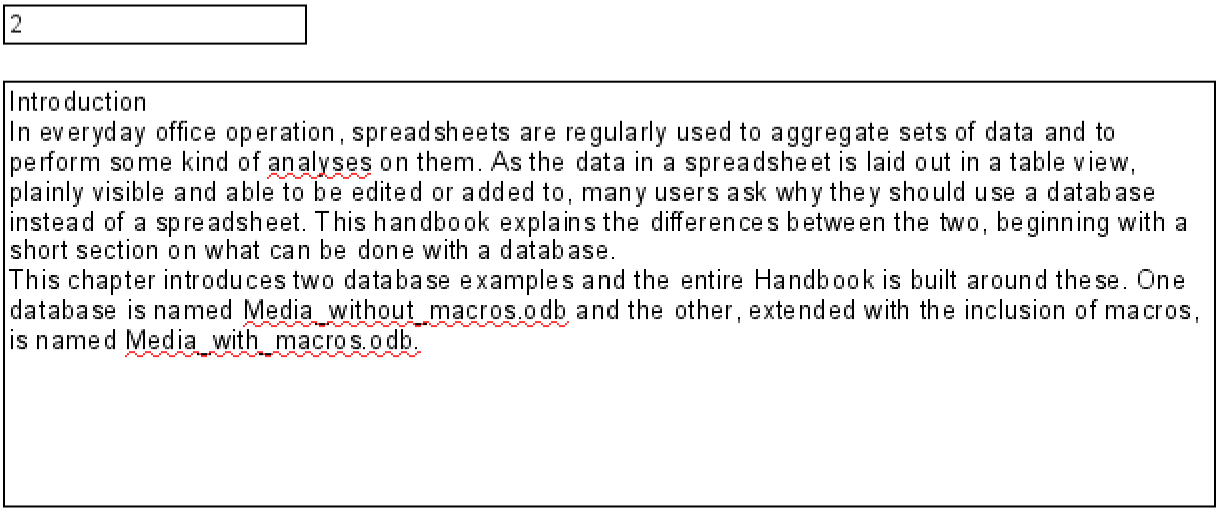
When the form is read or reloaded, the cursor stands before the current record. Then if an attempt is made, you get the message “Invalid cursor status”.

```
         inMemo = oForm.findColumn("memo")
         loID = oForm.findColumn("ID")
         oField = oForm.getByName("MemoFormat")
         stMemo = oField.Text
         If stMemo <> "" Then
            oForm.updateString(inMemo,stMemo)
         End If
         If stMemo <> "" And oForm.getString(loID) <> "" Then
            oForm.UpdateRow()
         End If
      End If
   End If
End Sub
```

The “memo” table field is located from the data source of the form, along with that for “ID”. If the field MemoFormat contains text, it is transferred into the Memo field of the data source using **`oForm.updateString`**(). Only if there is an entry in the ID field (in other words a primary key has been set) does an update follow. Otherwise a new record is inserted through the normal working of the form; the form recognizes the change and stores it independently.

Checking spelling during data entry

This macro can be used for multi-line formatted text fields. As in the previous chapter, the content of each record must first be written and then the new record can be loaded into the form control. The procedures TransferContent and WriteContent differ only in the point at which the search function can be bracketed out.



2

Introduction
In everyday office operation, spreadsheets are regularly used to aggregate sets of data and to perform some kind of analyses on them. As the data in a spreadsheet is laid out in a table view, plainly visible and able to be edited or added to, many users ask why they should use a database instead of a spreadsheet. This handbook explains the differences between the two, beginning with a short section on what can be done with a database.
This chapter introduces two database examples and the entire Handbook is built around these. One database is named Media_without_macros.odb and the other, extended with the inclusion of macros, is named Media_with_macros.odb.

Figure 430: Form to check spelling

The spelling checker is launched in the above form whenever a space or a return is hit within the form control. In other words, it runs at the end of each word. It could also be linked to the control losing focus to ensure that the last word is checked.

The procedure is bound to **Form > Events > Key released**.

```
SUB MarkWrongWordsDirect(oEvent As Object)
   GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

The **`RTrimStr`** function is used to remove any punctuation mark at the end of the string. Otherwise all words which ended with a comma, full stop or other punctuation mark would show up as spelling mistakes. In addition, **LTrimChar** is used to remove brackets at the beginning of words.

```
   Dim aProp() As New com.sun.star.beans.PropertyValue
   Dim oLinuSvcMgr As Object
   Dim oSpellChk As Object
   Dim oField As Object
   Dim arText()
   Dim stWord As String
   Dim inlenWord As Integer
   Dim ink As Integer
   Dim i As Integer
   Dim oCursor As Object
   Dim stText As Object
   oLinguSvcMgr = 
createUnoService("com.sun.star.linguistic2.LinguServiceManager")
   If Not IsNull(oLinguSvcMgr) Then
      oSpellChk = oLinguSvcMgr.getSpellChecker()
   End If
```

First all variables are declared. Then the Basic spell-checking module **SpellChecker** is accessed. It will be this module that will actually check individual words for correctness.

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

The event that launches the macro is a keystroke. This event includes a code, the **KeyCode**, for each individual key. The **KeyCode** for the Return key is 1280, the one for the space is 1284. Like many other pieces of information, these items are retrieved through the Xray tool. If space or return is pressed, the spelling is checked. It is launched, in other words, at the end of each word. Only the test for the last word does not occur automatically.

Each time the macro runs, all words in the text are checked. Checking individual words might also be possible but would take a lot more work.

The text is split up into single words. The delimiter is the space character. Before that, words split by line breaks must be joined together again, or the pieces might be mistaken for complete words.

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

The individual words are read out. Their untrimmed length is needed for the following editing step. Only so can the position of the word within the whole text (which is necessary for the specific highlighting of spelling mistakes) be determined.

Trim is used to remove spaces, while **RTrimStr** removes commas and full stops at the end of the text and **LTrimChar** any punctuation marks at the beginning.

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

If the word is not null, a text cursor is created. This cursor is moved without highlighting to the beginning of the text in the entry field. Then it jumps forward to the right, still without highlighting, to the term stored in the variable **ink**. This variable starts as 0, but after the first loop has run, it is equal to the length of the word (+1 for the following space). Then the cursor is moved to the right by the length of the current word. The font properties are modified to create the highlighted region.

The **spellchecker** is launched. It requires the word and the country code as arguments; without a country code everything counts as correct. The array argument is usually empty.

If the word is not in the dictionary, it is marked with a red wavy line. This type of underlining is represented here by **'9'**. If the word is found, there is no underline (**'0'**). This step is necessary because otherwise a word recognized as false and then corrected would continue to be shown with the red wavy line. It would never be removed because no conflicting format was given.

Comboboxes as listboxes with an entry option

A table with a single record can be directly created by using comboboxes and invisible numeric fields and the corresponding primary key entered into another table.

Tip

For the use of combo boxes instead of list boxes, see the database Example_Combobox_Listfield.odb associated with this book.

The Combobox control treats form fields for combined entry and choice of values (comboboxes) as listboxes with an entry option. For this purpose, in addition to the comboboxes in the form, the key field values which are to be transferred to the underlying table are stored in separate numeric fields. Fields can be declared as invisible. The keys from these fields are read in when the form is loaded and the combobox is set to show the corresponding content. If the content of the combobox is changed, it is saved and the new primary key is transferred into the corresponding numeric field to be stored in the main table.

If editable queries are used instead of tables, the text to be displayed in the combination fields can be directly determined from the query. A macro is then not required for this step.

An assumption for the functioning of the macro is that the primary key of the table which is the data source for the combination field is an automatically incrementing integer. It is also assumed that the field name for the primary key is ID.

Text display in comboboxes

This subroutine is to show text in the combobox according to the value of the invisible foreign key fields from the main form. It can also be used for listboxes which refer to two different tables. This might happen if, for example, the postcode in a postal address is stored separately from the town. In that case the postcode might be read from a table that contains only a foreign key for the town. The listbox should show postcode and town together.

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

This macro should be bound to the following form event: 'After record change'.

The macro is called directly from the form. The trigger event is the source for all the variables the macro needs. Some variables have already been declared globally in a separate module and are not declared here again.

```
   Dim oForm As Object
   Dim oFieldList As Object
   Dim stFieldValue As String
   Dim inCom As Integer
   Dim stQuery As String
   oForm = oEvent.Source
```

In the form there is a hidden control from which the names of all the different comboboxes can be obtained. One by one, these comboboxes are processed by the macro.

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

The additional information (Tag) attached to the hidden control contains this list of combobox names, separated by commas The names are written into an array and then processed within a loop. The loop ends with the **`NEXT`** term.

The combobox, which has replaced a listbox, is called **`oFieldList`**. To get the foreign key, we need the correct column in the table that underlies the form. This is accessible using the name of the table field, which is stored in the combobox’s additional information.

```
      oFieldList = oForm.getByName(Trim(aComboboxes(inCom)))
      stFieldID = oForm.getString(oForm.findColumn(oFieldList.Tag))
      oFieldList.Refresh()
```

The combobox is read in again using **Refresh()**in case the content of the field has been changed by the entry of new data.

The query needed to provide the visible content of the combobox is based on the field underlying the control and the value determined for the foreign key. To make the SQL code usable, any sort operation that might be present is removed. Then a check is made for any relationship definitions (which will begin with the word **WHERE**). By default the **InStr()** function does not distinguish between upper and lower case, so all case combinations are covered. If there is a relationship, it means that the query contains fields from two different tables. We need to find the table that provides the foreign key for the link. The macro depends here on the fact that the primary key in every table is called ID.

If there is no relationship defined, the query accesses only one table. The table information can be discarded and the condition formulated directly using the foreign key value.

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

Each field and table name must be entered into the SQL command with two sets of quotation marks. Quotation marks are normally interpreted by Basic as text string delimiters, so they no longer appear when the code is passed on to SQL. Doubling the quotation marks ensures that one set are passed on. **""ID""** signifies that the field **"ID"** will be accessed in the query, with the single set of quotes that SQL requires.

The query stored in the **stSql** variable is now carried out and its result saved in **oResult**.

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

The result of the query is read in a loop. As with a query in the GUI, several fields and records could be shown. But the construction of this query requires only one result, which will be found in the first column **(1)** of the query result set. It is the record which provides the displayed content of the combobox. The content is text (**getString()**), hence the command **oResult.getString(1)**.

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

The combobox must now be set to the text value retrieved by the query.

```
         oFieldList.Text = stFieldValue
      Else
```

If there is no value in the field for the foreign key **oField**, the query has failed and the combobox is set to an empty string.

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

This procedure manages the contact between the combobox and the foreign key available in a field of the form's data source. This should be enough to show the correct values in comboboxes. Storage of new values would require a further procedure.

Transfering a foreign key value from a combobox to a numeric field

If a new value is entered into the combobox (and this after all is the purpose for which this macro was constructed), the corresponding primary key must be entered into the form's underlying table as a Foreign key.

```
Sub TextSelectionSaveValue(oEvent As Object)
```

This macro should be bound to the following form event: 'Before record action'.

After the variables have been declared (not shown here), we must first determine exactly which event should launch the macro. Before record action, two implementations are called in succession. It is important for the macro itself to retrieve the form object. This can be done in both implementations but in different ways. Here the implementation called OdatabaseForm is filtered out.

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

This loop builds in the same start as the **Display_text** procedure:

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

The field **`oFieldList`** shows the text. It might lie inside a table control, in which case it is not possible to access it directly from the form. In such cases, the additional information for the hidden control comboboxes should contain the path to the field using “tablecontrol” combobox. Splitting this entry up will reveal how the combobox is to be accessed.

```
      a() = Split(Trim(aComboboxen(inCom)),">")
         If Ubound(a) > 0 Then
            oFieldList = oForm.getByName(a(0)).getByName(a(1))
         Else
            oFieldList = oForm.getByName(a(0))
         End If
```

Next the query is read from the combobox and split up into its individual parts. For simle comboboxes, the necessary items of information are the field name and table name:

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

This could in some cases be augmented by a sort instruction. Whenever two fields are to be put together in the combobox, more work will be required to separate them.

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

This query puts two fields together with a space between them. As the separator is a space, the macro will search for it and split the text into two parts accordingly. Naturally this will only work reliably if Field1 does not already contain text in which spaces are permitted. Otherwise, if the first name is “Anne Marie” and the surname “Müller”, the macro will treat “Anne” as the first name and “Marie Müller” as the surname. In such cases a more suitable separator should be used, which can then be found by the macro. In the case of names, this could be “Surname, Given name”.

Things get even more complicated if the two fields come from different tables:

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

Here the fields must be separated from one another, the table to which each field belongs must be established and the corresponding foreign keys determined.

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

The content of the query is stripped of unnecessary ballast. The parts are reassembled into an array with an unusual character combination as separator. FROM separates the visible field display from the table names. WHERE separates the condition from the table names. Joins are not supported.

```
            If Trim(UCASE(aFields(i))) = "ORDER BY" Then
               Exit For
            ElseIf Trim(UCASE(aFields(i))) = "FROM" Then
               stContent = stcontent+" §§ "
            ElseIf Trim(UCASE(aFields(i))) = "WHERE" Then
```

```
               stContent = stcontent+" §§ "
            Else
               stContent = stContent+Trim(aFields(i))
            End If
         Next i
         aContent() = Split(stContent, " §§ ")
```

In some cases the content of the visible field display comes from different fields:

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

The first part contains at least two fields. The fields begin with a table name. The second part contains two table names, which can be determined from the first part. The third part contains a relationship with a foreign key, separated by =:

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

The first part contains two field names without table names, possibly with separators. The second part contains the table names. There is no third part:

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

There is only one field from one table:

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

The maximum character length that an entry can have is given by the **`ColumnSize`** function. The combobox cannot be used to limit the size as it may need to contain two fields at the same time.

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

The content of the combobox is read out:

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

Leading and trailing spaces and non-printing characters are removed if necessary.

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

If a second table field exists, the content of the combobox must be split. To determine where the split is to occur, we use the field separator provided to the function as an argument.

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

The last parameter signifies that the maximum number of parts is 2.

Depending on which entry corresponds to field 1 and which to field 2, the content of the combobox is now allocated to the individual variables. "Position = 2" serves here as a sign that the second part of the content stands for Field 2.

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

It can happen that with two separable contents, the installed size of the combobox (text length) does not fit the table fields to be saved. For comboboxes that represent a single field, this is normally handled by suitably configuring the form control. Here by contrast, we need some way of catching such errors. The maximum permissible length of the relevant field is checked.

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

If the field length of the first or second part is too big, a default string is stored in one of the variables. The character **`Chr(13)`** is used to put in a line break.

```
      stmsgbox1 = "The field " + NameTableField1 + " must not exceed " + 
Field1Length + "characters in length." + Chr(13)
      stmsgbox2 = "The field " + NameTableField2 + " must not exceed " + 
Field2Length + "characters in length." + Chr(13)
```

If both field contents are too long, both texts are displayed.

```
      If (LengthField1 > 0 And Len(stContent) > LengthField1) And 
(LengthField2 > 0 And Len(stContentField2) > LengthField2) Then
         MsgBox("The entered text is too long." + Chr(13) + stmsgbox1 + 
stmsgbox2 + "Please shorten it.",64,"Invalid entry")
```

The display uses the **`MsgBox()`** function. This expects as its first argument a text string, then optionally a number (which determines the type of message box displayed), and finally an optional text string as a title for the window. The window will therefore have the title "Invalid entry" and the number '64' provides a box containing the Information symbol.

The following code covers any further cases of excessively long text that might arise.

```
      ElseIf (Field1Length > 0 And Len(stContent) > Field1Length) Then
         MsgBox("The entered text is too long." + Chr(13) + stmsgbox1 +
"Please shorten it.",64,"Invalid entry")
      Else
         MsgBox("The entered text is too long." + Chr(13) + stmsgbox2 +
"Please shorten it.",64,"Invalid entry")
      End If
   Else
```

If there is no excessively long text, the function can proceed. Otherwise it exits here.

Now the entries are masked so that any quotes that may be present will not generate an error.

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

First variables are preallocated which can subsequently be altered by the query. The variables inID1 and inID2 store the content of the primary key fields of the two tables. If a query yields no results, Basic assigns these integer variable a value of 0. However this value could also indicate a successful query returning a primary key value of 0; therefore the variable is preset to -1. HSQLDB cannot set this value for an autovalue field.

Next the database connection is set up, if it does not already exist.

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

If a second table field exists, a second dependency must first be declared.

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

If the content within the combobox is not present in the corresponding table, it is inserted there. The primary key value which results is then read. If it is present, the existing primary key is read in the same way. The function uses the automatically generated primary key fields (**`IDENTITY`**).

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

The primary key for the second value is temporarily stored in the variable **`inID2`** and then written as a foreign key into the table corresponding to the first value. According to whether the record from the first table was already available, the content is freshly saved (**`INSERT`**) or altered (**`UPDATE`**):

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

And the corresponding ID directly read out:

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

The primary key for the first table must finally be read again so that it can be transferred to the form's underlying table.

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

In the case where both the fields underlying the combobox are in the same table (for example Surname and Firstname in the Name table), a different query is needed:

```
      If NameTableField2 <> "" And NameTable2 = "" Then
```

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

... and a second table does not exist:

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

Then the primary key is read again.

```
         stSql = "CALL IDENTITY()"
         oResult = oSQL_Command.executeQuery(stSql)
            While oResult.next
               inID1 = oResult.getInt(1)
            Wend
      End If
   End If
   IF NameTableField2 = "" Then
```

Now we consider the simplest case: The second table field does not exist and the entry is not yet present in the table. In other words, a single new value has been entered into the combobox.

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

If there is no second field, the content of the box is inserted as a new record.

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

… and the resulting ID directly read out.

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

The value of the primary key field must be determined, so that it can be transferred to the main part of the form.

Next the primary key value that has resulted from all these loops is transferred to the invisible field in the main table and the underlying database. The table field linked to the form field is reached by using **'BoundField'**. **'updateInt'** places an integer (see under numerical type definitions) in this field.

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

If no primary key is to be entered, because there was no entry in the combobox or that entry was deleted, the content of the invisible field must also be deleted. **updateNull()** is used to fill the field with the database-specific expression for an empty field, **NULL**.

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

Function to measure the length of the combobox entry

The following function gives the number of characters in the respective table column, so that entries that are too long do not just get truncated. A **Function** is chosen here to provide return values. A **SUB** has no return value that can be passed on and processed elsewhere.

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

Generating database actions

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

This macro should be bound to the *When receiving focus* event of the listbox. It is necessary that in all cases where the listbox is changed, the change is stored. Without this macro, there would be no change in the actual table that Base could recognize, since the combobox is not bound to the form.

This macro directly alters the form properties:

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

This macro is not necessary for forms that use queries for the content of comboboxes. Changes in comboboxes are registered directly.

Navigation from one form to another

A form is to be opened when a particular event occurs.

In the form control properties, on the line "Additional information" (tag), enter the name of the form. Further information can also be entered here, and subsequently separated out by using the **`Split()`** function.

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

The array is declared and filled with the form names, first the form to be opened and secondly the current form, which will be closed after the other has been opened.

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

If instead, the other form is only to be opened when the current one is closed, for example where a main form exists and all other forms are controlled from it using buttons, the following macro should be bound to the form with **Tools > Customize > Events > Document closed**:

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

If the form documents are sorted within the ODB file into directories, the macro for changing the form needs to be more extensive:

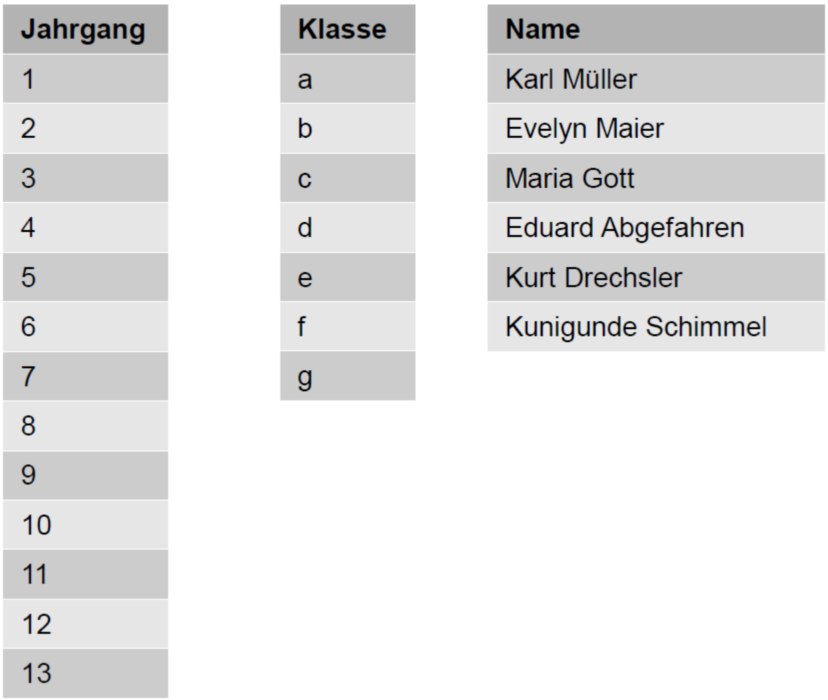
```
Sub From_form_to_form_with_folders(oEvent As Object)
   REM The form to be opened is given first.
   REM If a form is in a folder, use "/" to define the relationship
   REM so that the subfolder can be found.
   Dim stTag As String
   stTag = oEvent.Source.Model.Tag 'Tag is entered in the additional
information
   aForms() = Split(stTag, ",")    'Here the form name for the new form
comes first, then the one for the old form
   aForms1() = Split(aForms(0),"/")
   aForms2() = Split(aForms(1),"/")
   If UBound(aForms1()) = 0 Then
      ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0))
).open
   Else
      ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0))
).getByName( Trim(aForms1(1)) ).open
   End If
   If UBound(aForms2()) = 0 Then
      ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0))
).close
   Else
      ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0))
).getByName( Trim(aForms2(1)) ).close
   End If
End Sub
```

Form documents that lie in a directory are entered into the Additional Information field as directory/form. This must be converted to:

```
...getByName("Directory").getByName("Form").
```

Hierarchical listboxes

Settings in one listfield are intended to influence directly the settings of another. For simple cases, this has already been described above in the section on record filtering. But supposing that the first listbox is meant to affect the content of the second listbox, which then affects the content of a third listbox, and so on.



Jahrgang
1
2
3
4
5
6
7
8
9
10
11
12
13

Klasse
a
b
c
d
e
f
g

Name
Karl Müller
Evelyn Maier
Maria Gott
Eduard Abgefahren
Kurt Drechsler
Kunigunde Schimmel

Figure 431: Example list boxes for a hierarchical ordering of list boxes

In this example the first listbox (Jahrgang = Years) contains all school years. The Klasse (Classes) in each year are represented by letters. The Names are those of the class members.

Under normal circumstances, the Years listbox would show all 13 years, the Classes listbox all class letters and the Names listbox all pupils at the school.

If these are to be hierarchical listboxes, the choice of classes is restricted once a year has been selected. Only those class letters are shown that are actually present in that year. This might vary because, if pupil numbers are increasing, the number of classes in a year might also increase. The last listbox, Names, is very restricted. Instead of more than 1000 pupils, it would show only 30.

At the beginning, only the year can be selected. Once this has been done, the (restricted) list of classes is made available. Only at the end is the list of names given.

If the Years listbox is altered, the sequence must start again. If only the Classes listbox is altered, the year number for the last listbox remains valid

To create such a function, the form must be able to store an intermediate variable. This takes place in a hidden control.

The macro is bound to a change in the content of a listbox: **Properties Listbox > Events > Changed**. The necessary variables are stored in the additional information of the listbox.

Here is an example of the additional information provided:
`MainForm,Year,hidden control,Listbox_2`

The form is called MainForm. The current listbox is called Listbox1. This listbox shows the content of the table field Year and the following listboxes must be filtered according to this entry. The hidden control is designated by hidden_control and the existence of a second listbox (Listbox_2) is passed on to the filtering procedure.

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

  REM Tag goes into the Additional information field
  REM It contains:
  REM 0. Field name of field to be filtered in the table
  REM 1. Field name of the hidden control that will store the filtered value
  REM 2. Possible further listbox
  REM The tag is read from the element that launches the macro. The variable
is
  REM passed to the procedure, and if necessary to all further listboxes
   aFilter() = Split(stTag, ",")
   stFilter = ""
```

After the variables have been declared, the content of the tag is passed to an array, so that individual elements can be accessed. Then the access to the various fields in the form is declared.

The listbox that called the macro is determined and its value read. Only if this value is not NULL will it be combined with the name of the field to be filtered, in our example Year, to make a SQL command. Otherwise the filter will stay empty. If the listboxes are meant for filtering a form, no hidden control is available. In this case, the filter value is stored directly in the form.

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

If a filter already exists (for example one dealing with Listbox 2, which is now being accessed), the new content is attached to the previous content stored in the hidden control.

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

This must only happen when the same field has not yet been filtered. For example, if we are filtering for Year, a repetition of the filter will find no additional records for the Name listbox. A person can only be found in one year. We must therefore exclude the possibility that the filter name has already been used.

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

If a filter exists and the field that will be used for filtering is already present in the filter, the previous filtering on this fieldname must be deleted and a new filter created.

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

Then the filter is entered into the form. This filter can also be empty if the first listbox was selected and has no content.

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

The same procedure will run if the form does not need to be filtered immediately. In this case, the filter value is stored in the mean time in a hidden control.

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

If the Additional information has an entry numbered 4 (numbering begins at 0), the following listbox must be set to the corresponding entry from the caller listbox.

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

The necessary data for the filtering is read from the Additional information (Tag) in the corresponding listbox. It is not possible to write only the fresh SQL code into the listbox and then to read the listbox values. Instead the values corresponding to the query must be written into the listbox directly.

The creation of the code starts from the fact that the table to which the form refers is the same one to which the listboxes refer. Such a listbox is not designed to transfer foreign keys to the table.

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

The values are read into an array. The array is transferred directly into the listbox. The corresponding indices for the array are incremented within a loop.

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

The content of the listbox has been created afresh. Now it must be read in again. Then, using the Additional information property of the listbox that has been refreshed, each of the dependent listboxes that follows is emptied, launching a loop for all following listboxes until one is reached that has no fourth term in its Additional information.

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

The visible content of the listboxes are stored in `oField1.StringItemList`. If any additional value needs to be stored for transmission to the underlying table as a foreign key, as is usual for listboxes in forms, this value must be passed to the query separately and then stored with `oField1.ValueItemList`.

Such an extension requires additional variables such as, in addition to the table in which the values of the form are to be stored, the table from which the listbox contents are drawn.

Special care must be given to formulating the filter query.

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

This will work only if the underlying LibreOffice version is 4.1 or later, since it is the value which is to be stored that is given as `CurrentValue()`, and not the value that is displayed. To ensure that it works in different versions, set Property: Listbox > Data > Bound Field > '0'.

Entering times with milliseconds

To store times to millisecond precision requires a timestamp field in the table, separately adapted by SQL for the purpose (see “Table creation” in Chapter 3). Such a field can be represented on a form by a formatted field with the format **`MM:SS,00`**. However on the first attempt to write to it, record entry will fail. This can be corrected with the following macro, which should be bound to the form’s “Before record action” property:

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

The variables are declared first. The rest of the code is executed only when the Time field has something in it. Otherwise the internal mechanism of the form will act to set the field to **`NULL`**.

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

The entry in the Time field is broken down into its elements.

First the decimal part is separated out and right-padded with null characters to a total of nine digits. Such a high number can only be stored in a long variable.

Then the rest of the time is split into minutes and seconds, using the colon as a separator, and these are converted into integers.

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

The timestamp values are assigned to the standard LibreOffice date of 30.12.1899. Of course the actual current date can be stored alongside it.

Note

Getting and storing the current date:

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

Now the timestamp we have created is transferred to the field using **updateTimestamp** and stored in the form.

In earlier tutorials, **NanoSeconds** were called **HundrethSeconds**. This does not match the LibreOffice API and will cause an error message.

One event – several implementations

It can happen when using forms that a macro linked to a single event is run twice. This occurs because more than one process is linked simultaneously to, for example, the storage of a modified record. The differing causes for such an event can be determined in the following way:

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

When a modified record is stored, there are two implementations involved named **`org.openoffice.comp.svx.FormController`** and **`com.sun.star.comp.forms.ODatabaseForm`**. Using these names, we can ensure that a macro only runs through its code once. A duplicate run usually causes just a (small) pause in the program execution, but it can lead to things like a cursor being put back two records instead of one. Each implementation allows only specific commands, so knowing the name of the implementation can be important.

Saving with confirmation

For complicated record alterations, it makes sense to ask the user before execution whether the change should actually be carried out. If the answer in the dialog is No, the save is aborted, the change discarded, and the cursor remains on the current record.

```
Sub Save_confirmation(oEvent As Object)
   Dim oFormFeature As Object
   Dim oFormOperations As Object
   Dim inAnswer As Integer
   oFormFeature = com.sun.star.form.runtime.FormFeature
   Select Case oEvent.Source.ImplementationName
      Case "org.openoffice.comp.svx.FormController"
         inAnswer = MsgBox("Should the record be changed?" ,4,
"Change_record")
         Select Case inAnswer
            Case 6    ' Yes, no further action
            Case 7    ' No, interrupt save
               oFormOperations = oEvent.Source.FormOperations
               oFormOperations.execute(oFormFeature.UndoRecordChanges)
            Case Else
         End Select
      Case "com.sun.star.comp.forms.ODatabaseForm"
   End Select
End Sub
```

There are two trigger moments with different implementation names. These two implementations are distinguished in **`SELECT CASE`**. The code will be executed only for the **`FormController`** implementation. This is because only **`FormController`** has the variable **`FormOperations`**.

Apart from Yes and No, the user might also click on the close button. This however yields the same value as No, namely 7.

If the form is navigated with the tab key, the user sees only the dialog with the confirmation prompt. However, users who use the navigation bar will also see a message saying that the record will not be altered.

Primary key from running number and year

When invoices are prepared, yearly balances are affected. This often leads to a desire to separate the invoice tables of a database by year and to begin a new table each year.

The following macro solution uses a different method. It automatically writes the value of the ID field into the table but also takes account of the Year field which exists in the table as a secondary primary key. So the following primary keys might occur in the table:

Table 49: Example primary keys combining year and ID values

year	***ID***
2014	1
2014	2
2014	3
2015	1
2015	2

In this way an overview of the year is more easily obtained for documents.

```
Sub Current_Date_and_ID
   Dim oDatasource As Object
   Dim oConnection As Object
   Dim oSQL_Command As Object
   Dim stSql As String
   Dim oResult As Object
   Dim oDoc As Object
   Dim oDrawpage As Object
   Dim oForm As Object
   Dim oField1 As Object
   Dim oField2 As Object
   Dim oField3 As Object
   Dim inIDnew As Integer
   Dim inYear As Integer
   Dim unoDate
   oDoc = thisComponent
   oDrawpage = oDoc.drawpage
   oForm = oDrawpage.forms.getByName("MainForm")
   oField1 = oForm.getByName("fmt_year")
   oField2 = oForm.getByName("fmtID")
   oField3 = oForm.getByName("dat_date")
   If IsEmpty(oField2.getCurrentValue()) Then
      If IsEmpty(oField3.getCurrentValue()) Then
         unoDate = createUnoStruct("com.sun.star.util.Date")
         unoDate.Year = Year(Date)
         unoDate.Month = Month(Date)
         unoDate.Day = Day(Date)
         inYear = Year(Date)
      Else
         inYear = oField3.CurrentValue.Year
      End If
      oDatasource = ThisComponent.Parent.CurrentController
      If Not (oDatasource.isConnected()) Then
         oDatasource.connect()
      End If
      oConnection = oDatasource.ActiveConnection()
      oSQL_Command = oConnection.createStatement()
      stSql = "SELECT MAX( ""ID"" )+1 FROM ""orders"" WHERE ""year"" = '"
       + inYear + "'"
      oResult = oSQL_Command.executeQuery(stSql)
      While oResult.next
         inIDnew = oresult.getInt(1)
      Wend
      If inIDnew = 0 Then
         inIDnew = 1
      End If
      oField1.BoundField.updateInt(inYear)
      oField2.BoundField.updateInt(inIDnew)
      If IsEmpty(oField3.getCurrentValue()) Then
         oField3.BoundField.updateDate(unoDate)
      End If
   End If
End Sub
```

All variables are declared. The form controls in the main form are accessed. The rest of the code runs only if the entry for the fmtID field is still empty. Then, if no date has been entered, a date structure is created so that the current date and year can be carried across into the relevant fields. Then a connection is made to the database, if it does not exist already. The highest value of the ID field for the current year is incremented by 1. If the result set is empty, it means there are no entries in the ID field. At this point 0 could be entered in the fmtID control, but numbering for orders should begin at 1 so the inIDnew variable is given the value 1.

The returned value for the year, the ID and the current date (if no date has been entered) are transferred to the form.

In the form, the fields for the primary keys ID and Year are write-protected. Consequently they can only be given values using this macro.

Database tasks expanded using macros

Making a connection to a database

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

Here it would be possible to provide a username and a password, if one were necessary. In that case the brackets would contain ("Username","Password"). Instead of including the username and a password in clear text, the dialog for password protection is called up:

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

If however a form within the same Base file is accessing the database, you only need:

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

Here the database is known so a username and a password are not necessary, as these are already switched off in the basic HSQLDB configuration for internal version.

For forms outside Base, the connection is made through the first form:

```
oDataSource = Thiscomponent.Drawpage.Forms(0)
oConnection = oDataSource.activeConnection
```

Copying data from one database to another

The internal database is a single-user database. The records are stored inside the *.odb file. The exchange of data between different database files was not allowed for but is nevertheless possible using export and import.

But often *.odb files are set up to allow automatic data exchange between databases. The following procedure can be helpful here.

After the variables have been declared, the path to the current database is read from a button on the form. The database name is separated from the rest of the path. The target file for the records is also present in this folder. The name of this file is attached to the path to allow a connection to be made to the target database.

The connection to the source database is determined relative to the form that contains the button: **`ThisComponent.Parent.CurrentController`**. The connection to the external database is set up using the **`DatabaseContext`** and the path.

```
Sub DataCopy
   Dim oDatabaseContext As Object
   Dim oDatasource As Object
   Dim oDatasourceZiel As Object
   Dim oConnection As Object
   Dim oConnection Ziel As Object
   Dim oDB As Object
   Dim oSQL_Command As Object
   Dim oSQL_CommandTarget As Object
   Dim oResult As Object
   Dim oResultTarget As Object
   Dim stSql As String
   Dim stSqlTarget As String
   Dim inID As Integer
   Dim inIDTarget As Integer
   Dim stName As String
   Dim stTown As String
   oDB = ThisComponent.Parent
   stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
   stDir = ConvertToUrl(stDir & "TargetDB.odb")
   oDatasource = ThisComponent.Parent.CurrentController
   If Not (oDatasource.isConnected()) Then
      oDatasource.connect()
   End If
   oConnection = oDatasource.ActiveConnection()
   oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
   oDatasourceTarget = oDatabaseContext.getByName(stDir)
   oConnectionTarget = oDatasourceTarget.GetConnection("","")
   oSQL_Command = oConnection.createStatement()
   stSql = "SELECT * FROM ""table"""
   oResult = oSQL_Command.executeQuery(stSql)
   While oResult.next
      inID = oResult.getInt(1)
      stName = oResult.getString(2)
      stTown = oResult.getString(3)
      oSQL_CommandTarget = oConnectionTarget.createStatement()
      stSqlTarget = "SELECT ""ID"" FROM ""table"" WHERE ""ID"" = '"+inID+"'"
      oResultTarget = oSQL_CommandZiel.executeQuery(stSqlTarget)
      inIDZiel = - 1
      While oResultTarget.next
         inIDTarget = oResultTarget.getInt(1)
      Wend
      If inIDTarget = - 1 Then
         stSqlTarget = "INSERT INTO ""table"" (""ID"",""name"",""town"")
VALUES
           ('"+inID+"','"+stName+"','"+stTown+"')"
         oSQL_CommandTarget.executeUpdate(stSqlZiel)
      End If
   Wend
End Sub
```

The complete tables of the source database are read and inserted, line by line, into the tables of the target database using the connection that has been set up. Before the insertion, a check is made to see whether a value has been set for the primary key. If so, the record is not copied.

It can also be arranged that, instead of a new record being copied over, an existing record will be updated. In all cases, this makes certain that the target database contains records with the correct primary key of the source database.

Access to queries

It is easier to create queries in the graphical user interface than to transfer their text into macros, with the additional complication that duplicate double quotes are needed for all table and field names.

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

Here the content of a *.odb file is accessed from a form. The query is reached using **getQueryDefinitions**(). The SQL code for the query is in its **Command** field. This can then be used to utilize the command further within a macro.

When you are using the SQL code of the query, you must take care that the code does not refer to another query. That leads inevitably to the message that the (apparent) table from the database is unknown. Because of this, it is simpler to create views from queries and then access the views in the macro.

Securing your database

It can sometimes happen, especially when a database is being created, that the ODB file is unexpectedly truncated. Frequent saving after editing is therefore useful, especially when using the Reports module.

When the database is in use, it can be damaged by operating system failure, if this occurs just as the Base file is being terminated. This is when the content of the database is being written into the file.

In addition, there are the usual suspects for files that suddenly refuse to open, such as hard drive failure. It does no harm therefore to have a backup copy which is as up-to-date as possible. The state of the data does not change as long as the ODB file remains open. For this reason, safety subroutines can be directly linked to the opening of the file. You simply copy the file using the *Backups* path given in **Tools > Options > LibreOffice > Paths**. The macro begins to overwrite the oldest version after a specific number of copies (**`inMax`**).

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

If the macro is run when you launch the ODB file, sTitle and sUrl_Start will be correct. However, if the macro is carried out by a form, it must first determine whether a URL is available. If the URL is empty, a higher level (oDoc.Parent) for a value is looked up.

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

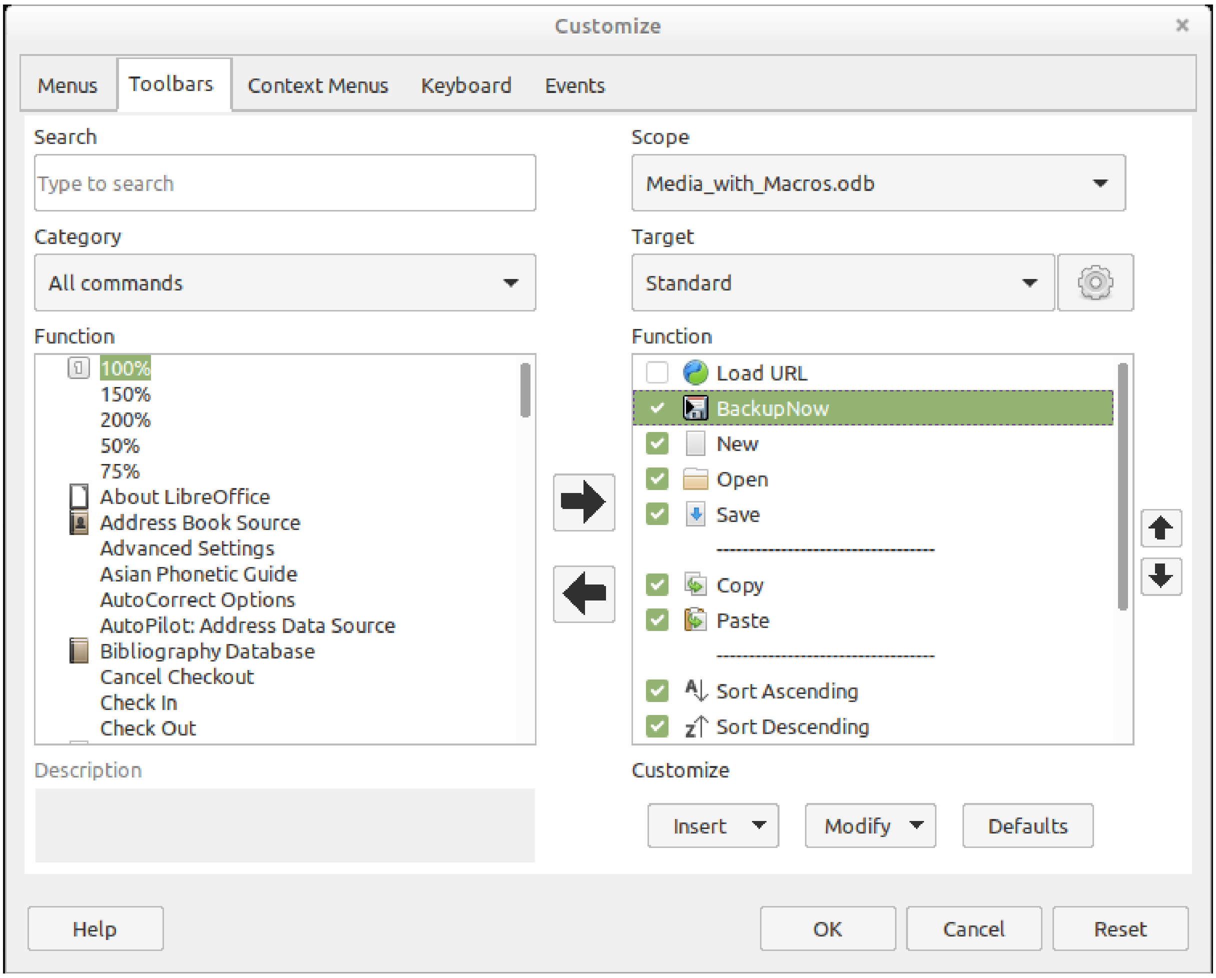
You can also do a backup while Base is running, provided that the data can be written back out of the cache into the file before the Databasebackup subroutine is carried out. It might be useful to do this, perhaps after a specific elapsed time or when an on-screen button is pressed. This cache-clearing is handled by the following subroutine:

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

If all this is to be launched from a single button on a form, both procedures must be called by a further procedure:

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

Especially for a security macro, it might make sense to make the macro accessible via the database’s toolbar. This is done in the main window of the Base file using **Tools > Customize > Toolbars**.



Search: Type to search

Category: All commands

Function:
100%
150%
200%
50%
75%
About LibreOffice
Address Book Source
Advanced Settings
Asian Phonetic Guide
AutoCorrect Options
AutoPilot: Address Data Source
Bibliography Database
Cancel Checkout
Check In
Check Out

Description

Scope: Media_with_Macros.odb

Target: Standard

Function:
□ Load URL
✓ BackupNow
☑ New
☑ Open
☑ Save

☑ Copy
☑ Paste

☑ Sort Ascending
☑ Sort Descending

Customize

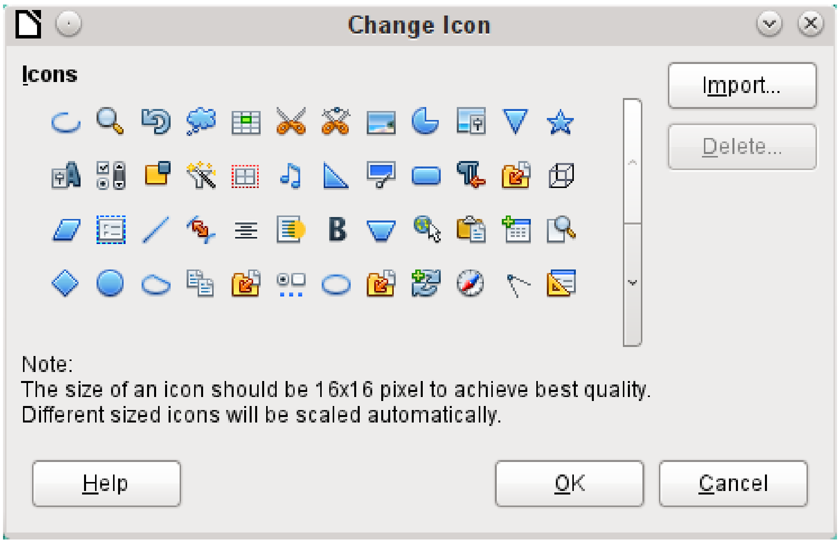
Figure 432: Using Toolbars tab of Customize dialog to make a macro accessible on Standard toolbar

On the Customize dialog, under Scope, the command must be stored in the Base file, which in this case is Media_with_Macros.odb.

Under Target, select the Standard toolbar, which works in all parts of Base.

The dialog now shows relevant functions in the right-hand list. Select the procedure BackupNow.

The command is now available for use in the top-level toolbar. To assign an icon to the command, choose **Modify > Change Icon** to open the dialog shown in Figure 433.



Note:
The size of an icon should be 16x16 pixel to achieve best quality.
Different sized icons will be scaled automatically.

Figure 433: Change Icon dialog

Select a suitable icon. You can also create and add your own icon.

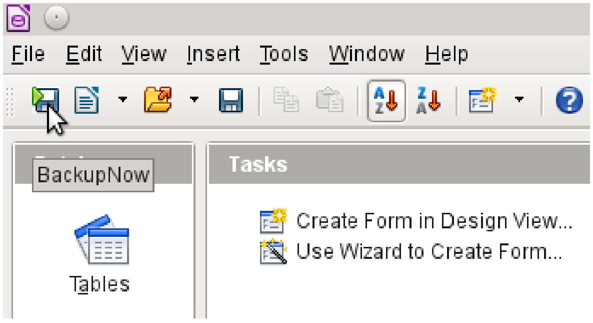

File Edit View Insert Tools Window Help
BackupNow
Tables
Tasks
Create Form in Design View...
Use Wizard to Create Form...

Figure 434: New icon visible on the Standard toolbar

The icon now appears instead of the name of the procedure. The name becomes a tooltip.

To carry out the procedure, just click the icon on the toolbar.

Database compaction

This is simply a SQL command (**SHUTDOWN COMPACT**), which should be carried out now and again, especially after a lot of data has been deleted. The database stores new data, but still reserves the space for the deleted data. In cases where the data have been substantially altered, you therefore need to compact the database.

Note

Since LibreOffice version 3.6, this command is automatically carried out for the internal HSQLDB when the database is closed. Therefore this macro is no longer necessary for the internal database.

Once compaction is carried out, the tables are no longer accessible. The file must be reopened. Therefore this macro closes the form from which it is called. However, you cannot close the document itself without causing a recovery when it is opened again. Therefore this function is commented out.

```
Sub Database_compaction
   Dim stMessage As String
   oDataSource = ThisComponent.Parent.CurrentController   ' Accessible
from the form
   If Not (oDataSource.isConnected()) Then
      oDataSource.connect()
   End If
   oConnection = oDataSource.ActiveConnection()
   oSQL_Statement = oConnection.createStatement()
   stSql = "SHUTDOWN COMPACT"   ' The database is being compacted and shut
down
   oSQL_Statement.executeQuery(stSql)
   stMessage = "The database is being compacted." + Chr(13) + "The form will
now close."
   stMessage = stMessage + Chr(13) + "Following this, the database file
should be closed."
   stMessage = stMessage + Chr(13) + "The database can only be accessed
after reopening the database file."
   MsgBox stMessage
   ThisDatabaseDocument.FormDocuments.getByName( "Maintenance" ).close
   REM The closing of the database file causes a recovery operation when you
open it again.
'   ThisDatabaseDocument.close(True)
End Sub
```

Decreasing the table index for autovalue fields

If a lot of data is deleted from a table, users are often concerned that the sequence of automatically generated primary keys simply continues upwards instead of starting again at the highest current value of the key. The following subroutine reads the currently highest value of the ID field in a table and sets the next initial key value 1 higher than this maximum.

If the primary key field is not called ID, the macro must be edited accordingly.

```
Sub Table_index_down(stTable As String)
   REM This subroutine sets the automatically incrementing primary key field
mit the preset name of "ID" to the lowest possible value.
   Dim inCount As Integer
   Dim inSequence_Value As Integer
   oDataSource = ThisComponent.Parent.CurrentController   ' Accessible
through the  form
   If Not (oDataSource.isConnected()) Then
      oDataSource.connect()
   End If
   oConnection = oDataSource.ActiveConnection()
   oSQL_Statement = oConnection.createStatement()
   stSql = "SELECT MAX(""ID"") FROM """+stTable+""""   ' The highest value
in "ID" is determined
```

```
   oQuery_result = oSQL_Statement.executeQuery(stSql)   ' Query is launched and the return value stored in the variable oQuery_result
   If Not IsNull(oQuery_result) Then
      While oQuery_result.next
      inCount = oQuery_result.getInt(1)   ' First data field is read
      Wend    ' next record, in this case none as only one record exists
      If inCount = "" Then   ' If the highest value is not a value, meaning the table is empty, the highest value is set to -1
         inCount = -1
      End If
      inSequence_Value = inCount+1   ' The highest value is increased by 1
     REM A new command is prepared for the database. The ID will start afresh from inCount+1.
     REM This statement has no return value, as no record is being read
      oSQL_statement = oConnection.createStatement()
      oSQL_statement.executeQuery("ALTER TABLE """ + stTable + """ ALTER COLUMN ""ID"" RESTART WITH " + inSequence_Value + "")
   End If
End Sub
```

Printing from Base

The standard way of getting a printable document frrom Base is to use a report. Alternatively, tables and queries can be copied into Calc and prepared for printing there. Of course direct printing of a form from the screen is also possible.

Printing a report from an internal form

Normally the generation of reports is done from the Base user interface. A click on the report name launches the preparation of the report. It would be easier of course if the report could be launched directly from a form.

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

All the reports are accessed by name from their container **ReportDocuments**. They are opened with **open**. If a report is bound to a query that is filtered through the form, this method allows the current record to be printed.

Launching, formatting, directly printing, and closing a report

It would be even nicer if the report could be sent directly to the printer. The following combination of procedures adds a few little features. It first selects the active record in the form, reformats the report so that the text fields are set automatically for the correct height, and then launches the report. Finally the report is printed and optionally stored as a pdf. And all this happens almost completely in the background, as the report is switched to invisible directly after the form is opened and is closed again after printing. Suggestions for the various procedures were made by Andrew Pitonyak, Thomas Krumbein, and Lionel Elie Mamane.

```
Sub ReportStart(oEvent As Object)
```

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

The ReportStart procedure is linked to a button in the form. Using this button, the primary key of the current record can be read. From the event that launches the macro, we can reach the form (**`oForm`**). The name of the primary key field is given here as **`"ID"`**. Using **`oForm.getInt(oForm.findColumn("ID"))`**, the key is read from the field as an integer. This value is stored in a filter table. The filter table controls a query to ensure that only the current record will be used for the report.

The report can be opened without reference to the form. It is then accessible as an object (**`oReport`**). The report window is made invisible. Note that it cannot be invisible when it is called up, so it appears briefly, then is filled with the appropriate content in the background.

Next the ReportLineHeightAuto procedure is launched. This procedure is passed a reference to the open report as an argument.

The height of the record line can be set automatically at print time. If there is likely to be too much text in a particular field, the text is truncated and the remainder indicated by a red triangle. When this is not working, the following procedure will ensure that in all tables with the name Detail, automatic height control will be switched on.

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

When the report is created, care must be taken that all fields on the same line of the Detail section have the same height. Otherwise, the automatic height control can suddenly set a line to double height.

Once all tables with the name Detail have had automatic height control set, the report is sent to the printer by the PrintCloseReport procedure.

The Props array contains the values that are associated with a printer in a document. For the print command, the name of the default printer is important. The report should remain open until the printing is actually completed. This is ensured by giving the printer name and the “Wait until I’m finished” (**Wait**) command as arguments.

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

Only when the print has been completely sent to the printer is the document closed.

For printer settings, see the Printer and print settings section from the wiki.

If, instead of (or in addition to) a print-out, you want a pdf of the document as a security copy, the **storeToURL()** method can be used:

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

The URL must of course be a complete URL address. Better still, this address should be linked to a permanent record of the printed document such as an invoice number. Otherwise it could happen that a security file could simply be overwritten by the next print.

Printing reports from an external form

There are problems when external forms are being used. The reports lie within the *.odb file and are not available using the datasource browser.

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

The report is launched from a button on the external form. The button tells the form the path to the *.odb file: **oForm.DataSourceName**. Then the file is opened using **loadComponentFromUrl**. The file should remain in the background, so the document view is accessed and the interface is set to **Visible = False**. Ideally this should have been done directly using the argument list **Arg()**, but tests show that this does not give the correct result.

The report cannot be called up immediately from the opened document as the connection is not yet ready. The report appears with a gray background and then LibreOffice crashes. A short wait of 100 milliseconds solves this problem. Practical tests are necessary to determine the minimum waiting time. Now the report is launched. As the report will be in a separate text file, the open *.odb file can be closed again. The **`oDocument.close(True)`** method passes this instruction to the *.odb file. The file will only be closed when it is no longer active, i.e. no more records are to be passed to the report.

A similar access can be launched from forms within the *.odb file, but in this case the document should not be closed.

You can obtain good quality prints significantly faster than with the Report Builder by using macros combined with the mailmerge function or text fields.

Doing a mail merge from Base

Sometimes a report is simply inadequate to produce good-quality letters to addressees. The text fields in a report are of very limited use in practice. Instead, a mail merge letter can be created in Writer. It is not however necessary to open Writer first, do all the entry and customization there and then print. You can do all that directly from Base, using a macro.

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

The name given for the data source is the one under which the database is registered in LibreOffice. This name need not be identical with the file name. The registered name in this example is Addresses.

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

The path to the mailmerge file must be formatted according to the conventions of your operating system, in this example an absolute path in a Linux system.

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

The type of command is set out. 0 stands for a table, 1 for a query and 2 for a direct SQL command.

```
   oMailMerge.CommandType = 1
```

Here a query has been chosen with the name MailmergeQuery.

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

A filter is used to determine which records are to be used for the mailmerge print. This filter might, for example, be specified using a form control and passed from Base to the macro. Using the primary key of a record could cause a single document to be printed.

In this example, the field Gender in the MailmergeQuery is selected and then searched for records that have ‘m’ in this field.

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

Available output types are Printer (1), File (2) and Mail (3). Here for test purposes, an output file is chosen. This file is stored on the given path. For each mailmerge record there will be one print. To distinguish this print, the surname field is incorporated into the filename.

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

If the filter is provided with its data via the form, this provides a way of doing mailmerges without opening Writer.

Printing via text fields

Using **Insert > Field > More Fields > Functions > Placeholder**, a model can be created in Writer for a document that is to be printed in the future. The placeholders should be provided with the same names as the fields in the database table or query underlying the form from which the macro is called.

For the simple case, the type to choose for the placeholder is Text.

The path to the model must be provided in the macro. A new document Unknown1.odt is created. The macro fills the placeholders with the contents of the current record from the query. The open document can then be edited as required.

The example database Example_database_mailmerge_direct.odb shows how a complete invoice can be produced with the help of text fields and access to a prepared table within the model document. Unlike the invoices created with the Report Builder, this type of invoice creation does not have height limitations for the fields from the table. All text is displayed.

Here is part of the code, mainly supplied by DPunch: http://de.openoffice.info/viewtopic.php?f=8&t=45868#p194799

```
Sub Filling_Textfields
   oForm = thisComponent.Drawpage.Forms.MainForm
   If oForm.RowCount = 0 Then
      MsgBox "No available record for printing"
      Exit Sub
   End If
```

The main form is activated. The button that launches the macro could also be used to find the form. Then the macro establishes that the form actually contains printable data.

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

Direct access to the URL from the form is not possible. It must be done using the higher-level reference to the database.

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

The database title is separated from the URL.

```
   stDir = stDir & "Beispiel_Textfelder.ott"
```

The model is found and opened

```
   Dim args(0) As New com.sun.star.beans.PropertyValue
   args(0).Name = "AsTemplate"
   args(0).Value = True
   oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

The text fields are written in.

```
   oTextfields = oNewDoc.Textfields.createEnumeration
   Do While oTextfields.hasMoreElements
```

```
      oTextfield = oTextfields.nextElement
      If oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit")
Then
         stColumnname = oTextfield.PlaceHolder
```

Placeholder represents the text field.

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

If the name of the text field is the same as the column name in the underlying dataset, the content of the database is transferred to the field in the text document.

```
            inIndex = oForm.findColumn(stColumnname)
            oTextfield.Anchor.String = oForm.getString(inIndex)
         End If
      End If
   Loop
End Sub
```

Calling applications to open files

This procedure allows a single click in a text field to call up the program that is linked to the filename suffix in the operating system. In this way internet links can be followed or an email program launched for a specific address stored in the database.

For this section see also the example database Example_Mail_File_activate.odb.

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

The content of the named field is read. This could be a web address beginning with **`'http://'`**, an email address beginning with **`'@'`** or a path to a document (for example an externally stored image or PDF file).

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

If the field is empty, the macro exits immediately. During data entry, it often happens that fields are accessed using the mouse, but clicking the field for the purpose of writing into it for the first time should not lead to the macro code being executed.

Now the field is searched for a **'@'** character. This would indicate an email address. The email program should be launched to send mail to this address.

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

If there is no **'@'**, the term is converted into a URL. If this starts with **'http://'**, we are not dealing with a file in the local filesystem but with an Internet resource that must be looked up with a web browser. Otherwise the path will begin with the term **'file:///'**.

```
   Else
      stFeld = convertToUrl(stField)
   End If
```

Now the program assigned by the operating system to such files is searched for. For the keyword **'mailto:'** this is the mail program, for **'http://'** the browser, and otherwise the system must decide using the filename suffix.

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

Calling a mail program with predefined content

The previous example can be extended to launch a mail program with a predefined subject and content.

For this section see also the example database Example_Mail_File_activate.odb.

The mail program is launched using **'mailto:recipient?subject= &body= &cc= &bcc= '**. The last two entries are not present in the form. Attachments are not provided for in the definition of **'mailto'** but sometimes **'attachment='** works.

```
Sub Mai*l_activate
   Dim oDoc As Object
   Dim oDrawpage As Object
   Dim oForm As Object
   Dim oField1 As Object
   Dim oField2 As Object
   Dim oField3 As Object
   Dim oField4 As Object
   Dim oShell As Object
   Dim stField1 As String
   Dim stField2 As String
   Dim stField3 As String
   Dim stField4 As String
   oDoc = thisComponent
   oDrawpage = oDoc.Drawpage
   oForm = oDrawpage.Forms.getByName("form")
   oField1 = oForm.getByName("mail_to")
   oField2 = oForm.getByName("mail_subject")
   oField3 = oForm.getByName("mail_body")
   stField1 = oField1.Text
   If stField1 = "" Then
      MsgBox "Missing email address." & Chr(13) &
         "Email program would not be activated" , 48, "Send Email"
      Exit Sub
   End If
```

The conversion to URL is necessary to prevent special characters and line breaks from interfering with the call. This does however prefix the term **'file:///'** to the path. These 8 characters at the beginning are not transferred.

```
   stField2 = Mid(ConvertToUrl(oFeld2.Text),9)
   stField3 = Mid(ConvertToUrl(oFeld3.Text),9)
```

In contrast to a simple program launch, the details of the mail invocation are given here as part of the execute call.

```
   oShell = createUnoService("com.sun.star.system.SystemShellExecute")
   oShell.execute("mailto:" + stField1 + "?subject=" + stField2 + "&body="
+ stField3,,0)
End Sub
```

Note

Sending email with the help of a mail program can also be done using the following code, but the actual content of the email cannot be inserted this way.

```
Dim attachs(0)
oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
oMailProgramm = oMailer.querySimpleMailClient()
oNewmessage = oMailProgramm.createSimpleMailMessage()
oNeemessage.setRecipient(stField1)
oNewmessage.setSubject(stField2)
attachs(0) = "file:///..."
oNeueNachricht.setAttachement(attachs())
oMailprogramm.sendSimpleMailMessage(oNeuenachricht, 0 )
```

For possible parameters, see details of the *com::sun::star::system::XSimpleMailMessage* interface in the API Reference.

Changing the mouse pointer when traversing a link

This is normal on the Internet and Base recreates it: the mouse pointer traverses a link and changes into a pointing hand. The link text might also change its properties, becoming blue and underlined. The resemblance to an Internet link is perfect. Any user will expect a click to open an external program.

For this section see the example database Example_Mail_File_activate.odb.

This short procedure should be bound to the textbox's **'Mouse inside'** event

```
Sub Mouse_pointer(Event As Object)
   REM See also Standardlibraries: Tools → ModuleControls → SwitchMousePointer
   Dim oPointer As Object
   oPointer = createUnoService("com.sun.star.awt.Pointer")
   oPointer.setType(27)   'Types see com.sun.star.awt.SystemPointer
   Event.Source.Peer.SetPointer(oPointer)
End Sub
```

Showing forms without a toolbar

New Base users are often irritated that a toolbar exists but is not usable within a form. These toolbars can be removed in various ways. The best ways in all LibreOffice versions are the two described below.

Window sizes and toolbars are usually controlled by a macro that is launched from a form document using **Tools > Customize > Events > Open Document**. This refers to the whole document, not an individual main or subform.

Forms without a toolbar in the window

The size of a window can be varied. Using the appropriate button it can also be closed. These tasks are carried out by your system’s window manager. The position and size of a window on the screen can be supplied by a macro when the program starts.

```
Sub Hide_toolbar
   Dim oFrame As Object
   Dim oWin As Object
   Dim oLayoutMng As Object
   Dim aElements()
   oFrame = StarDesktop.getCurrentFrame()
```

The form title is to be shown in the window’s title bar.

```
   oFrame.setTitle "My Form"
   oWin = oFrame.getContainerWindow()
```

The window is maximized. This is not the same thing as full-screen mode, since the taskbar is still visible and the window has a title bar, which can be used to change its size or close it..

```
   oWin.IsMaximized = true
```

It is possible to create a window with a specific size and position. This is carried out with **`'oWin.setPosSize(0,0,600,400,15)'`**. Here the window appears at the top left corner of the screen with a width of 600 pixels and a height of 400. The last number indicates that all pixels are given. It is called **`'Flag'`**. **`'Flag'`** is calculated from the sum of the following values: x=1, y=2, breadth=4, height=8. As x, y, breadth and height are all given, **`'Flag'`** has the size 1+2+4+8=15.

```
   oLayoutMng = oFrame.LayoutManager
   aElements = oLayoutMng.getElements()
   For i = LBound(aElements) To UBound(aElements)
      If aElements(i).ResourceURL =
        "private:resource/toolbar/formsnavigationbar" Then
      Else
         oLayoutMng.hideElement(aElements(i).ResourceURL)
      End If
   Next
End Sub
```

In the case of a form navigation bar, nothing is to be done. The form must after all remain usable in cases where a navigation bar control has not been built in (which would cause the navigation bar to be hidden anyway). Only toolbars other than the navigation bar should be hidden. For this reason there is no action for this case.

If the toolbars are not restored directly after leaving the form, they will still be hidden. They can of course be restored using **View > Toolbars**. But it would be rather annoying if the standard toolbar (**View > Toolbars > Standard**) or the status bar (**View > Status Bar**) was missing.

This procedure restores (**`'showElement'`**) the toolbars from their hidden state (**`'hideElement'`**). The comments contain the bars whose absence is most likely to be noticed.

```
Sub Show_toolbar
   Dim oFrame As Object
   Dim oLayoutMng As Object
   Dim aElements()
   oFrame = StarDesktop.getCurrentFrame()
   oLayoutMng = oFrame.LayoutManager
   aElements = oLayoutMng.getElements()
   For i = LBound(aElements) To UBound(aElements)
      oLayoutMng.showElement(aElements(i).ResourceURL)
   Next
  ' important elements which may be absent:
  '   "private:resource/toolbar/standardbar"
  '   "private:resource/statusbar/statusbar"
End Sub
```

The macros are bound to: **Tools > Customize > Events > Open Document > Hide_toolbar** and **Close Document > Show_toolbar**.

The toolbars often fail to come back. In the worst cases, it can be helpful not to read out those elements that the layout manager already knows, but first to create particular toolbars and then show them:

```
Sub Hide_toolbar
   Dim oFrame As Object
   Dim oLayoutMng As Object
   Dim i As Integer
   Dim aElements(5) As String
   oFrame = StarDesktop.getCurrentFrame()
   oLayoutMng = oFrame.LayoutManager
   aElements(0) = "private:resource/menubar/menubar"
   aElements(1) = "private:resource/statusbar/statusbar"
   aElements(2) = "private:resource/toolbar/formsnavigationbar"
   aElements(3) = "private:resource/toolbar/standardbar"
   aElements(4) = "private:resource/toolbar/formdesign"
   aElements(5) = "private:resource/toolbar/formcontrols"
   For Each i In aElemente()
      IF Not(oLayoutMng.requestElement(i)) Then
         oLayoutMng.createElement(i)
      End If
   oLayoutMng.showElement(i)
   Next i
End Sub
```

The toolbars that are to be created are named explicitly. If a corresponding toolbar is not available to the layout manager, it is created using **`createElement`** and then displayed using **`showElement`**.

Forms in full-screen mode

In full-screen mode, the whole screen is covered by the form. There is no taskbar or other elements which might show if other programs are running.

```
Function Fullscreen(boSwitch As Boolean)
   Dim oDispatcher As Object
   Dim Props(0) As New com.sun.star.beans.PropertyValue
   oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
   Props(0).Name = "FullScreen"
   Props(0).Value = boSwitch
   oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame,
      ".uno:FullScreen", "", 0, Props())
End Function
```

This function is launched using the following procedure. In the procedure, the previous procedure also runs simultaneously to remove the toolbars – otherwise the toolbar will appear and the full-screen mode can be switched off using it. This is also a toolbar, although it has only one symbol.

```
Sub Fullscreen_on
   Fullscreen(true)
   Hide_toolbar
End Sub
```

You exit from full-screen mode by pressing the **`'ESC'`** key. If instead, a specific button is to be used for this command, the following line can be used:

```
Sub Fullscreen_off
   Fullscreen(false)
   Show_toolbar
End Sub
```

Launching forms directly from the opening of the database

When the toolbars are gone or a form is to be shown in full-screen mode, the database file must launch the form directly when it opens. A simple command to open a form will not work, as the database connection does not yet exist when the file is opened.

The following macro is launched from **Tools > Customize > Events > Open Document**. Use the option **Save in > Databasefile.odb**.

```
Sub Form_Directstart
   Dim oDatasource As Object
   oDatasource = ThisDatabaseDocument.CurrentController
   If Not (oDatasource.isConnected()) Then
      oDatasource.connect()
   End If
   ThisDatabaseDocument.FormDocuments.getByName("Formname").open
End Sub
```

First a connection to the database must be made. The controller is part of **`ThisDatabaseDocument`**, just as the form is. Then the form can be launched and can read its data out of the database.

Accessing a MySQL database with macros

All the macros shown up to now have been part of an internal HSQLDB database. When working with external databases, a few changes and extensions are necessary.

MySQL code in macros

When the internal database is being accessed, tables and fields must be enclosed in duplicate double quotes, compared with the SQL:

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

As these SQL commands must be prepared inside macros, the double quotes must be masked:

```
stSQL = "SELECT ""Field"" FROM ""Table"""
```

MySQL queries use a different form of masking:

```
SELECT `Field` FROM `Database`.`Table`
```

Inside the macro code, this form of masking appears as:

```
stSql = "SELECT `Field` FROM `Database`.`Table`"
```

Temporary tables as individual intermediate storage

In the previous chapter, a one-line table was frequently used for searching or filtering tables. This will not work in a multi-user system, as other users would then be dependent on someone else's filter value. Temporary tables in MySQL are only accessible to the user of the active connection, so these tables can be accessed for searching and filtering.

Naturally such tables cannot be created in advance. They must be created when the Base file is opened. Therefore the following macro should be bound to the opening of the *.odb file.

```
Sub CreateTempTable
   oDatasource = thisDatabaseDocument.CurrentController
   If Not (oDatasource.isConnected()) Then oDatasource.connect()
   oConnection = oDatasource.ActiveConnection()
   oSQL_Statement = oConnection.createStatement()
   stSql = "CREATE TEMPORARY TABLE IF NOT EXISTS `Searchtmp` (`ID` INT
PRIMARY KEY,
     `Name` VARCHAR(50))"
   oSQL_Statement.executeUpdate(stSql)
End Sub
```

When the *.odb file is first opened, there is no connection to an external MySQL database. The connection must be created. Then a temporary table with the necessary fields can be set up.

Dialogs

In Base you can use dialogs rather than forms for data entry, data modification, or database maintenance. Dialogs can be directly customized for the current application environment, but naturally they are not as comfortably defined in advance as forms are. Here is a short introduction ending in a quite complicated example for use in database maintenance.

Launching and ending dialogs

First the dialog must be created on the appropriate computer. This is done using **Tools > Macros > Organize Dialogs > Database filename > Standard > New**. The dialog appears with a continuous gray surface and a titlebar with a close icon. This empty dialog can now be called up and then closed again.

When the dialog is clicked, there is a possibility under general properties to set a size and position. Also the content of the title Start Dialogs can be entered.

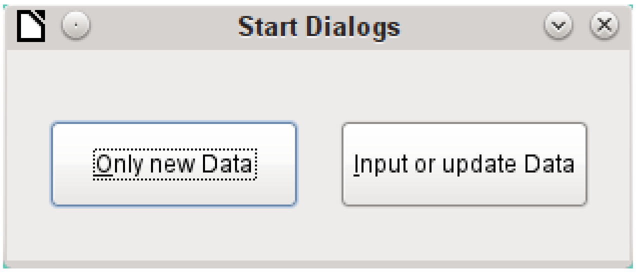


Figure 435: Start Dialogs dialog

The toolbar at the bottom edge of the window contains various form controls. From this, two buttons have been selected for our dialog, allowing it to launch other dialogs. The editing of content and the binding of macros to events is carried out in the same way as for buttons in forms.

The positioning of variable declarations for dialogs requires special care. The dialog is declared as a global variable so that it can be accessed by different procedures. In this case, the dialog is called oDialog0 because there will be further dialogs with higher sequence numbers.

```
Dim oDialog0 As Object
```

First the library for the dialog is loaded. It is in the Standard directory, if no other name was chosen when the dialog was created. The dialog itself can be reached in this library by using the name Dialog0. **`Execute()`** launches the dialog.

```
Sub Dialog0Start
   DialogLibraries.LoadLibrary("Standard")
   oDialog0 = createUnoDialog(DialogLibraries.Standard.Dialog0)
   oDialog0.Execute()
End Sub
```

In principle, a dialog can be closed using the Close button on the frame. However, if you want another specific button for this, the command **`EndExecute()`** should be used within the procedure.

```
Sub Dialog0Ende
   oDialog0.EndExecute()
End Sub
```

Within this framework, any number of dialogs can be launched and closed again.

Simple dialog for entering new records

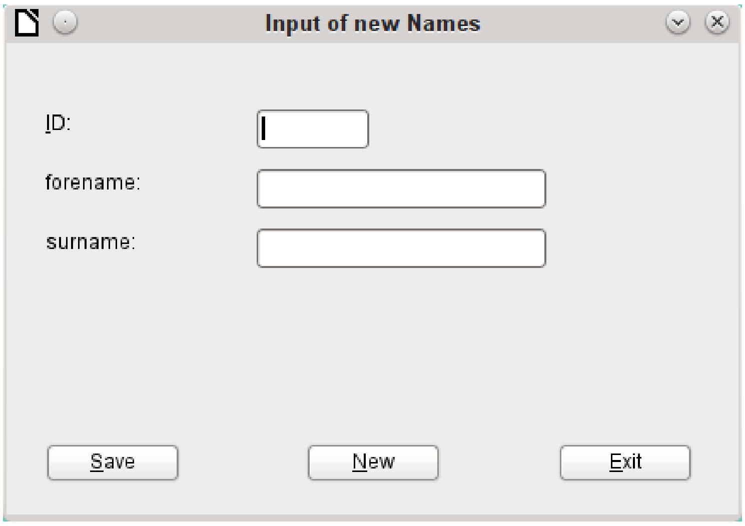


Figure 436: A simple dialog for entering new records

This dialog is a first step for the following dialog for editing records. First the basic approach to managing tables is clarified. Here we are dealing with the storage of records with new primary keys or the complete new entry of records. How far a little dialog like this can suffice for input into a particular database depends on the requirements of the user.

```
Dim oDialog1 As Object
```

directly creates a global variable for the dialog at the top level of the module before all procedures.

The dialog is opened and closed in the same way as for the previous dialog. Only the name is changed from Dialog0 to Dialog1. The procedure for closing the dialog is bound to the Exit button.

The New button is used to clear all controls in the dialog from earlier entries, using the DatafieldsClear procedure.

```
Sub DatafieldsClear
   oDialog1.getControl("NumericField1").Text = ""
   oDialog1.getControl("TextField1").Text = ""
   oDialog1.getControl("TextField2").Text = ""
End Sub
```

Each control that has been inserted into a dialog is accessible by name. The user interface will ensure that names are not duplicated, which is not the case with controls in a form.

The **getControl** method is used with the name of the control. Numeric fields too have a **Text** property which can be used here. That is the only way a numeric field can be emptied. Empty text exists but there is no such thing as an empty number. Instead a 0 must be written in the primary key field.

The **Save** button launches the Data1Save procedure:

```
Sub Data1Save
   Dim oDatasource As Object
   Dim oConnection As Object
   Dim oSQL_Command As Object
   Dim loID As Long
   Dim stForename As String
   Dim stSurname As String
   loID = oDialog1.getControl("NumericField1").Value
   stForename = oDialog1.getControl("TextField1").Text
   stSurname = oDialog1.getControl("TextField2").Text
   If loID > 0 And stSurname <> "" Then
      oDatasource = thisDatabaseDocument.CurrentController
      If Not (oDatasource.isConnected()) Then
         oDatasource.connect()
      End If
      oConnection = oDatasource.ActiveConnection()
      oSQL_Command = oConnection.createStatement()
      stSql = "SELECT ""ID"" FROM ""name"" WHERE ""ID"" = '"+loID+"'"
      oResult = oSQL_Command.executeQuery(stSql)
      While oResult.next
         MsgBox ("The value for field 'ID' already exist",16,
            "Duplicate Value")
         Exit Sub
      Wend
      stSql = "INSERT INTO ""name"" (""ID"", ""forename"", ""surname"")
        VALUES ('"+loID+"','"+stForename+"','"+stSurname+"')"
      oSQL_Command.executeUpdate(stSql)
      DatafieldsClear
   End If
End Sub
```

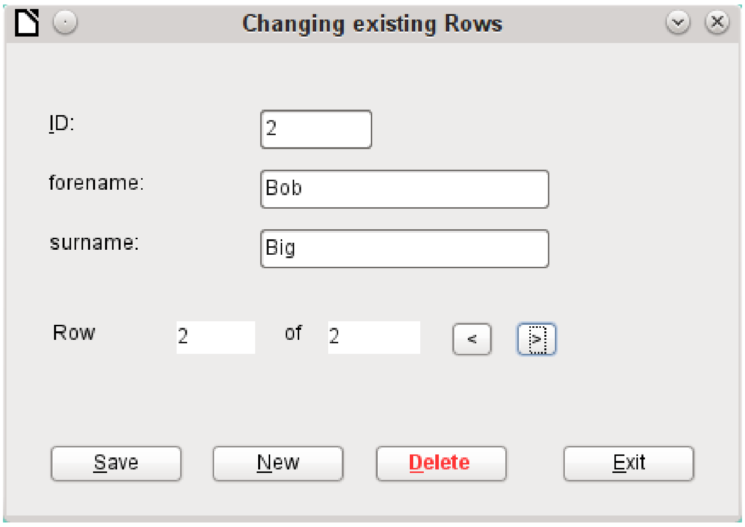
As in the DatafieldsClear procedure, the entry fields are accessed. This time the access is for reading only. Only if the ID field has an entry greater than 0 and the Surname field also contains text will the record be passed on. A null value for the ID can be excluded because a numeric variable for integer numbers is always initialized to 0. An empty field is therefore stored with a zero value.

If both fields have been supplied with content, a connection is made to the database. As the controls are not in a form, the database connection must be made using **`thisDatabaseDocument.CurrentController`**.

First the database is queried to see if a record with the given primary key already exists. If this query produces a result, a message box appears containing a Stop symbol (code: **16**) and the message “Duplicate record entry”. Then the procedure exits with **`Exit SUB`**.

If the query finds no record with the same primary key, the new record is inserted into the database using the insert command. Then the DatafieldsClear procedure is called to provide a new empty form.

Dialog for editing records in a table



ID: 2

forename: Bob

surname: Big

Row 2 of 2

Figure 437: Dialog for creating, editing, and deleting records

This dialog clearly offers more possibilities than the previous one. Here all records can be displayed and you can navigate through them, create new ones or delete records. Naturally the code is much more complicated.

The Exit button is bound to the procedure, modified for Dialog2, which was described in the previous dialog for entering new records. Here the remaining buttons and their functions are described.

Data entry in the dialog is restricted in that the ID field must have a minimum value of 1. This limitation has to do with the handling of variables in Basic: numeric variables are by definition initialized to 0. Therefore if numeric values from empty fields and those from fields containing 0 are read out, Basic can detect no difference between them. This means that if a 0 were to be used in the ID field, it would have to be read first as text and perhaps converted to a number later.

The dialog is loaded under the same conditions as before. However the loading procedure is made dependent on a zero value for the variable passed by the DataLoad procedure.

```
Sub DataLoad(loID As Long)
   Dim oDatasource As Object
   Dim oConnection As Object
   Dim oSQL_Command As Object
   Dim stForename As String
   Dim stSurname As String
   Dim loRow As Long
   Dim loRowMax As Long
   Dim inStart As Integer
   oDatasource = thisDatabaseDocument.CurrentController
   If Not (oDatasource.isConnected()) Then
      oDatasource.connect()
   End If
   oConnection = oDatasource.ActiveConnection()
   oSQL_Command = oConnection.createStatement()
   If loID < 1 Then
      stSql = "SELECT MIN(""ID"") FROM ""name"""
      oResult = oSQL_Command.executeQuery(stSql)
      While oResult.next
         loID = oResult.getInt(1)
      Wend
      inStart = 1
   End If
```

The variables are declared. The database connection for the dialog is established as described above. At the beginning, **`loID is 0`**. This case provides the lowest primary key value allowed by SQL. The corresponding record will later be displayed in the dialog. At the same time the **`inStart`** variable is set to 1, so that the dialog can be launched later. If the table does not contain any records, **`loID`** will remain **`0`**. In that case, there will be no need to search for the number and contents of any corresponding records.

Only if **`loID`** is greater than 0 will a query test to see which records are available in the database. Then a second query will count all records that are to be displayed. The third query gives the position of the current record by counting all records with the current primary key or less.

```
   If loID > 0 Then
      stSql = "SELECT * FROM ""name"" WHERE ""ID"" = '"+loID+"'"
      oResult = oSQL_Command.executeQuery(stSql)
      While oResult.next
         loID = oResult.getInt(1)
         stForename = oResult.getString(2)
         stSurname = oResult.getString(3)
      Wend
      stSql = "SELECT COUNT(""ID"") FROM ""name"""
      oResult = oSQL_Command.executeQuery(stSql)
      While oResult.next
         loRowMax = oResult.getInt(1)
      Wend
      stSql = "SELECT COUNT(""ID"") FROM ""name"" WHERE ""ID"" <=
'"+loID+"'"
      oResult = oSQL_Command.executeQuery(stSql)
      While oResult.next
         loRow = oResult.getInt(1)
      Wend
      oDialog2.getControl("NumericField1").Value = loID
      oDialog2.getControl("TextField1").Text = stForename
      oDialog2.getControl("TextField2").Text = stSurname
   End If
   oDialog2.getControl("NumericField2").Value = loRow
   oDialog2.getControl("NumericField3").Value = loRowMax
   If loRow = 1 Then
      ' previous Row
      oDialog2.getControl("CommandButton4").Model.enabled = False
   Else
      oDialog2.getControl("CommandButton4").Model.enabled = True
   End If
   If loRow <= loRowMax Then
      ' next Row | new Row | delete
      oDialog2.getControl("CommandButton5").Model.enabled = True
      oDialog2.getControl("CommandButton2").Model.enabled = True
      oDialog2.getControl("CommandButton6").Model.enabled = True
   Else
      oDialog2.getControl("CommandButton5").Model.enabled = False
      oDialog2.getControl("CommandButton2").Model.enabled = False
      oDialog2.getControl("CommandButton6").Model.enabled = False
   End If
   IF inStart = 1 Then
      oDialog2.Execute()
   End If
End Sub
```

The retrieved values are transferred to the dialog fields. The entries for the current record number and the total number of records retrieved are always written in, replacing the default numeric value of 0.

The navigation buttons (CommandButton5 and CommandButton4) are only usable when it is possible to reach the corresponding record. Otherwise they are temporarily deactivated with **enabled = False**. The same is true for the New and Delete buttons. They should not be available when the number of the displayed row is higher than the maximum number of rows that was determined. This is the default setting of this dialog when entering records.

If possible, the dialog should only be launched when it is to be created directly from a starting file using **`DataLoad(0)`**. That is why the special variable **`inStart`** is given the value 1 at the beginning of the procedure.

The < button is used to navigate to the previous record. Therefore this button is active only when the record displayed is not the first in the list. Navigation requires the primary key for the current record to be read from the field NumericField1.

Here there are two possible cases:

1) You are moving forward to a new entry, so the corresponding field has no value. In this case, **`loID`** has the default value which, according to the definition of an integer variable, is 0.
2) Otherwise **`loID`** will contain a value that is greater than 0. Then a query can determine the ID value directly below the current one.

```
Sub PreviousRow
   Dim loID As Long
   Dim loIDnew As Long
   loID = oDialog2.getControl("NumericField1").Value
   oDatasource = thisDatabaseDocument.CurrentController
   If Not (oDatasource.isConnected()) Then
      oDatasource.connect()
   End If
   oConnection = oDatasource.ActiveConnection()
   oSQL_Command = oConnection.createStatement()
   If loID < 1 Then
      stSql = "SELECT MAX(""ID"") FROM ""name"""
   Else
      stSql = "SELECT MAX(""ID"") FROM ""name"" WHERE ""ID"" < '"+loID+"'"
   End If
   oResult = oSQL_Command.executeQuery(stSql)
   While oResult.next
      loIDnew = oResult.getInt(1)
   Wend
   If loIDnew > 0 Then
      DataLoad(loIDnew)
   End If
End Sub
```

If the ID field is empty, the display should change to the highest value of the primary key number. If, on the other hand, the ID field refers to a record, the previous value of ID should be returned.

The result of this query is used to run the DataLoad procedure again with the corresponding key value.

The > button is used to navigate to the next record. This possibility should exist only when the dialog has not been emptied for the entry of a new record. This will naturally be the case when the dialog is launched and also with an empty table.

A value in NumericField1 is mandatory. Starting from this value, SQL can determine which primary key is the next highest in the table. If the query's result set is empty because there is no corresponding record, the value for **`loIDnew = 0`**. Otherwise the content of the next record is read using DataLoad.

```
Sub NextRow
   Dim loID As Long
   Dim loIDnew As Long
   loID = oDialog2.getControl("NumericField1").Value
   oDatasource = thisDatabaseDocument.CurrentController
   If Not (oDatasource.isConnected()) Then
      oDatasource.connect()
   End If
   oConnection = oDatasource.ActiveConnection()
   oSQL_Command = oConnection.createStatement()
   stSql = "SELECT MIN(""ID"") FROM ""name"" WHERE ""ID"" > '"+loID+"'"
   oResult = oSQL_Command.executeQuery(stSql)
   While oResult.next
      loIDnew = oResult.getInt(1)
   Wend
   If loIDnew > 0 Then
      DataLoad(loIDnew)
   Else
      Datafields2Clear
   End If
End Sub
```

If when navigating to the next record, there is no further record, the navigation key launches the following procedure Datafields2Clear, which serves to prepare for the input of a new record.

The Datafields2Clear procedure does not just empty the data fields themselves. The position of the current record is set to one higher than the maximum record number, making it clear that the record currently being worked on is not yet included in the database.

As soon as Datafields2Clear has been launched, the possibility of jumping to the previous record is activated, Jumps to a following record, and the use of the procedures for New and Delete are deactivated.

```
Sub Datafields2Clear
   loRowMax = oDialog2.getControl("NumericField3").Value
   oDialog2.getControl("NumericField1").Text = ""
   oDialog2.getControl("TextField1").Text = ""
   oDialog2.getControl("TextField2").Text = ""
   oDialog2.getControl("NumericField2").Value = loRowMax + 1
   oDialog2.getControl("CommandButton4").Model.enabled = True    ' Previous
record
   oDialog2.getControl("CommandButton5").Model.enabled = False    ' Next
record   oDialog2.getControl("CommandButton2").Model.enabled = False    ' New
record
   oDialog2.getControl("CommandButton6").Model.enabled = False    ' Delete
End Sub
```

Saving records should only be possible when the ID and Surname fields contain entries. If this condition is met, the procedure tests whether this is a new record. This makes use of the record pointer which is set for new records to be one higher than the maximum number of records

For new records, checks are made to ensure that the save operation will be successful. If the number used for the primary key has been used before, a warning is displayed. If the associated question is answered with Yes, the existing record with this number is overwritten. Otherwise, the save will be aborted. If there are no existing entries in the database (**`loRowMax = 0`**), this test is unnecessary and the new record can be saved directly. For a new record, the number of records is incremented by 1 and the entries are cleared for the next record.

Existing records are simply overwritten with an update command.

```
Sub Data2Save(oEvent As Object)
   Dim oDatasource As Object
   Dim oConnection As Object
   Dim oSQL_Command As Object
   Dim oDlg As Object
   Dim loID As Long
   Dim stForename As String
   Dim stSurname As String
   Dim inMsg As Integer
   Dim loRow As Long
   Dim loRowMax As Long
   Dim stSql As String
   oDlg = oEvent.Source.getContext()
   loID = oDlg.getControl("NumericField1").Value
   stForename = oDlg.getControl("TextField1").Text
   stSurname = oDlg.getControl("TextField2").Text
   If loID > 0 And stSurname <> "" Then
      oDatasource = thisDatabaseDocument.CurrentController
      If Not (oDatasource.isConnected()) Then
         oDatasource.connect()
      End If
      oConnection = oDatasource.ActiveConnection()
      oSQL_Command = oConnection.createStatement()
      loRow = oDlg.getControl("NumericField2").Value
      loRowMax = oDlg.getControl("NumericField3").Value
      If loRowMax < loRow Then
         If loRowMax > 0 Then
            stSql = "SELECT ""ID"" FROM ""name"" WHERE ""ID"" = '"+loID+"'"
            oResult = oSQL_Command.executeQuery(stSql)
            While oResult.next
               inMsg = MsgBox ("The value for field 'ID' already exist." &
                  CHR(13) & "Should the row be updated?",20,
                  "Duplicate Value")
               If inMsg = 6 Then
                  stSql = "UPDATE ""name"" SET
""forename""='"+stForename+"',
                   ""surname""='"+stSurname+"' WHERE ""ID"" = '"+loID+"'"
                  oSQL_Command.executeUpdate(stSql)
                  DataLoad(loID)   ' With update a row has been rewritten.
Rowcount                           must be resetted
               End If
               Exit Sub
            Wend
         End If
         stSql = "INSERT INTO ""name"" (""ID"", ""forename"", ""surname"")
VALUES
           ('"+loID+"','"+stForename+"','"+stSurname+"')"
         oSQL_Command.executeUpdate(stSql)
         oDlg.getControl("NumericField3").Value = loRowMax + 1
            ' After instert one row is added
         Datafields2Clear
            ' After insert would be moved to next insert
      Else
         stSql = "UPDATE ""name"" SET ""forename""='"+stForename+"',
           ""surname""='"+stSurname+"' WHERE ""ID"" = '"+loID+"'"
         oSQL_Command.executeUpdate(stSql)
      End If
   End If
End Sub
```

The delete procedure is provided with a supplementary question to prevent accidental deletions. Since this button is deactivated when the entry fields are empty, an empty NumericField1 should never occur. Therefore the check condition **IF loID > 0** can be omitted.

Deletion causes the number of records to be decremented by 1. This must be corrected using **loRowMax – 1**. Then the record following the current one is displayed.

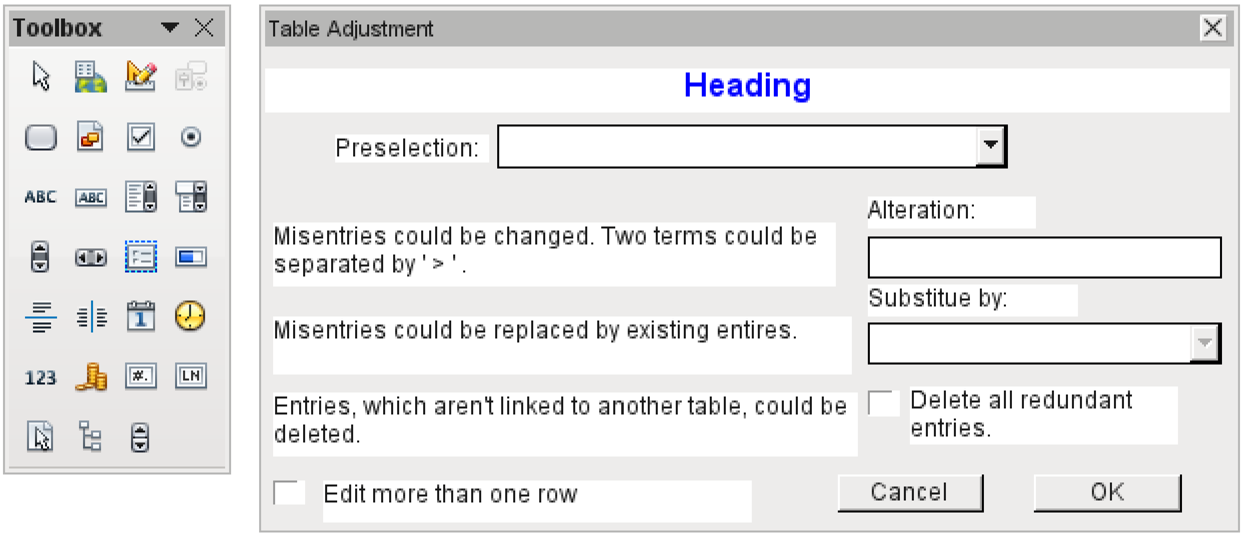
```
Sub DataDelete(oEvent As Object)
   Dim oDatasource As Object
   Dim oConnection As Object
   Dim oSQL_Command As Object
   Dim oDlg As Object
   Dim loID As Long
   oDlg = oEvent.Source.getContext()
   loID = oDlg.getControl("NumericField1").Value
   If loID > 0 Then
      inMsg = MsgBox ("Should current data be deleted?",20,
         "Delete current row")
      If inMsg = 6 Then
         oDatasource = thisDatabaseDocument.CurrentController
         If Not (oDatasource.isConnected()) Then
            oDatasource.connect()
         End If
         oConnection = oDatasource.ActiveConnection()
         oSQL_Command = oConnection.createStatement()
         stSql = "DELETE FROM ""name"" WHERE ""ID"" = '"+loID+"'"
         oSQL_Command.executeUpdate(stSql)
         loRowMax = oDlg.getControl("NumericField3").Value
         oDlg.getControl("NumericField3").Value = loRowMax - 1
         NextRow
      End If
   ELSE
      MsgBox ("No row deleted." & CHR(13) &
         "No data selected.",64,"Delete impossible")
   End If
End Sub
```

This little dialog has shown that the use of macro code can provide a basis for processing records. Access via forms is much easier, but a dialog can be very flexible in adapting to the requirements of the program. However it is not suitable for the quick creation of a database interface.

Using a dialog to clean up bad entries in tables

Input errors in fields are often only noticed later. Often it is necessary to modify identical entries in several records at the same time. It is awkward to have to do this in normal table view, especially when several records must be edited, as each record requires an individual entry to be made.

Forms can use macros to do this kind of thing, but to do it for several tables, you would need identically constructed forms. Dialogs can do the job. A dialog can be supplied at the beginning with the necessary data for appropriate tables and can be called up by several different forms.



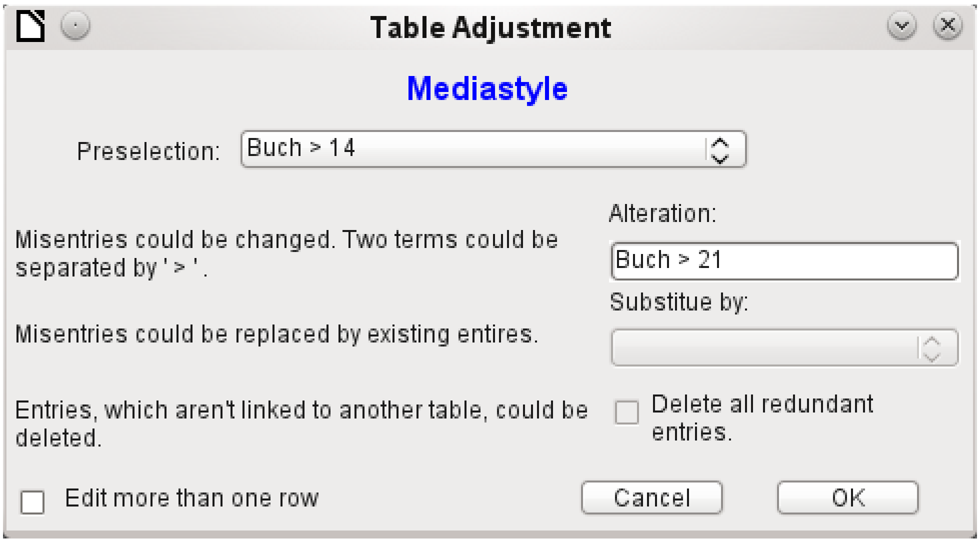
Misentries could be changed. Two terms could be separated by ' > '.

Misentries could be replaced by existing entires.

Entries, which aren't linked to another table, could be deleted.

Figure 438:Designing a Table Adjustment dialog using the Toolbox toolbar (left)

Dialogs are saved along with the modules for macros. Their creation is similar to that of a form. Very similar control fields are available. Only the table control of forms is absent as a special entry possibility.



Preselection: Buch > 14

Misentries could be changed. Two terms could be separated by ' > '.

Alteration: Buch > 21

Misentries could be replaced by existing entires.

Substitue by:

Entries, which aren't linked to another table, could be deleted.

□ Delete all redundant entries.

□ Edit more than one row

Figure 439: Using the Table Adjustment dialog

The appearance of dialog controls is determined by the settings for the graphical user interface.

The dialog shown above serves in the example database to edit tables which are not used directly as the basis of a form. So, for example, the media type is accessible only through a listbox (in the macro version it becomes a combobox). In the macro version, the field contents can be expanded by new content but an alteration of existing content is not possible. In the version without macros, alterations are carried out using a separate table control.

While alterations in this case are easy to carry out without macros, it is quite difficult to change the media type of many media at once. Suppose the following types are available: "Book, bound", "Book, hard-cover", "Paperback", and "Ringfile". Now it turns out, after the database has been in use for a long time, that more active contemporaries foresaw similar additional media types for printed works. The task of differentiating them has become excessive. We therefore wish to reduce them, preferably to a single term. Without macros, the records in the media table would have to be found (using a filter) and individually altered. If you know SQL, you can do it much better using a SQL command. You can change all the records in the Media table with a single entry. A second SQL command then removes the now surplus media types which no longer have any link to the Media table. Precisely this method is applied using this dialog's Replace With box – only the SQL command is first adapted to the Media Type table using a macro that can also edit other tables.

Often entries slip into a table which with hindsight can be changed in the form, and so are no longer needed. It does no harm simply to delete such orphaned entries, but they are quite hard to find using the graphical user interface. Here again a suitable SQL command is useful, coupled with a delete instruction. This command for affected tables is included in the dialog under Delete all superfluous entries.

If the dialog is to be used to carry out several changes, this is indicated by the Edit multiple records checkbox. Then the dialog will not simply terminate when the OK button is clicked.

The macro code for this dialog can be seen in full in the example database. Only excerpts are explained below.

```
Sub Table_purge(oEvent As Object)
```

The macro should be launched by entering into the Additional information section for the relevant buttons:

```
0: Form, 1: Subform, 2: SubSubform, 3: Combobox or table control, 4: Foreign key field
in a form, empty for a table control, 5: Table name of auxiliary table, 6: Table field1
of auxiliary table, 7: Table field2 of auxiliary table, or 8: Table name of auxiliary
table for table field2
```

The entries in this area are listed at the beginning of the macro as comments. The numbers bound to them are transferred and the relevant entry is read from an array. The macro can edit listboxes, which have two entries, separated by **">"**. These two entries can also come from different tables and be brought together using a query, as for instance in the Postcode table, which has only the foreign key field Town_ID for the town, requiring the Town table to display the names of towns.

```
   Dim aForeignTable(0, 0 to 1)
   Dim aForeignTable2(0, 0 to 1)
```

Among the variables defined at the beginning are two arrays. While normal arrays can be created by the **Split()** command during execution of the subroutine, two-dimensional arrays must be defined in advance. Two-dimensional arrays are necessary to store several records from one query when the query itself refers to more than one field. The two arrays declared above must be able to interpret queries that refer to two table fields. Therefore they are defined for two different contents by using 0 to 1 for the second dimension.

```
   stTag = oEvent.Source.Model.Tag
   aTable() = Split(stTag, ", ")
   For i = LBound(aTable()) To UBound(aTable())
      aTable(i) = trim(aTable(i))
   Next
```

The variables provided are read. The sequence is that set up in the comment above. There is a maximum of nine entries, and you need to declare if an eighth entry for the table field2 and a nineth entry for a second table exist.

If values are to be removed from a table, it is first necessary to check that they do not exist as foreign keys in some other table. In simple table structures a given table will have only one foreign key connection to another table. However, in the given example database, there is a Town table which is used for both the place of publication of media and the town for addresses. Thus the primary key of the Town table is entered twice into different tables. These tables and foreign key names can naturally also be entered using the Additional Information field. It would be nicer though if they could be provided universally for all cases. This can be done using the following query.

```
   stSql = "SELECT ""FKTABLE_NAME"", ""FKCOLUMN_NAME"" FROM
""INFORMATION_SCHEMA"".""SYSTEM_CROSSREFERENCE"" WHERE ""PKTABLE_NAME"" = '" +
aTable(5) + "'"
```

In the database, the INFORMATION_SCHEMA area contains all information about the tables of the database, including information about foreign keys. The tables that contain this information can be accessed using "INFORMATION_SCHEMA"."SYSTEM_CROSSREFERENCE". KTABLE_NAME" gives the table that provides its primary key for the connection. FKTABLE_NAME gives the table that uses this primary key as a foreign key. Finally FKCOLUMN_NAME gives the name of the foreign key field.

The table that provides its primary key for use as a foreign key is in the previously created array at position 6. A the count begins with 0, the value is read from the array using **aTable(5)**.

```
   inCount = 0
   stForeignIDTab1Tab2 = "ID"
   stForeignIDTab2Tab1 = "ID"
   stAuxiltable = aTable(5)
```

Before the reading of the arrays begins, some default values must be set. These are the index for the array in which the values from the auxiliary table will be written, the default primary key if we do not need the foreign key for a second table, and the default auxiliary table, linked to the main table, for postcode and town, the Postcode table.

When two fields are linked for display in a listbox, they can, as described above, come from two different tables. For the display of Postcode and town the query is:

```
SELECT "Postcode"."Postcode" || ' > ' || "Town"."Town" FROM "Postcode",
"Town" WHERE "Postcode"."Town_ID" = "Town"."ID"
```

The table for the first field (Postcode), is linked to the second table by a foreign key. Only the information from these two tables and the Postcode and Town fields is passed to the macro. All primary keys are by default called ID in the example database. The foreign key of Town in Postcode must therefore be determined using the macro.

In the same way the macro must access each table with which the content of the listbox is connected by a foreign key.

```
   oQuery_result = oSQL_Statement.executeQuery(stSql)
   If Not IsNull(oQuery_result) Then
      While oQuery_result.next
         ReDim Preserve aForeignTable(inCount,0 to 1)
```

The array must be freshly dimensioned each time. In order to preserve the existing contents, they are backed up using (Preserve).

```
         aForeignTables(inCount,0) = oQuery_result.getString(1)
```

Reading the first field with the name of the table which contains the foreign key. The result for the Postcode table is the Address table.

```
         aForeignTables(inCount,1) = oQuery_result.getString(2)
```

Reading the second field with the name of the foreign key field. The result for the Postcode table is the field Postcode_ID in the Address table.

In cases where a call to the subroutine includes the name of a second table, the following loop is run. Only when the name of the second table occurs as the foreign key table for the first table is the default entry changed. In our case this does not occur, as the Town table has no foreign key from the Postcode table. The default entry for the auxiliary table therefore remains Postcode; finally the combination of postcode and town is a basis for the Address table, which contains a foreign key from the Postcode table.

```
         If UBound(aTable()) = 8 Then
            If aTable(8) = aForeignTable(inCount,0) Then
               stForeignIDTab2Tab1 = aForeignTable(inCount,1)
               stAuxiltable = aTable(8)
            End If
         End If
         inCount = inCount + 1
```

As further values may need to be read in, the index is incremented to redimension the arrays. Then the loop ends.

```
      Wend
   End If
```

If, when the subroutine is called, a second table name exists, the same query is launched for this table:

```
   If UBound(aTable()) = 8 Then
```

It runs identically except that the loop tests whether perhaps the first table name occurs as a foreign key table name. That is the case here: the Postcode table contains the foreign key Town_ID from the Town table. This foreign key is now assigned to the variable stForeignIDTab1Tab2, so that the relationship between the tables can be defined.

```
            If aTable(5) = aForeignTable2(inCount,0) Then
               stForeignIDTab1Tab2 = aForeignTable2(inCount,1)
            End If
```

After a few further settings to ensure a return to the correct form after running the dialog (determining the line number of the form, so that we can jump back to that line number after a new read), the loop begins, which recreates the dialog when the first action is completed but the dialog is required to be kept open for further actions. The setting for repetition takes place using the corresponding checkbox.

```
   Do
```

Before the dialog is launched, first of all the content of the listboxes is determined. Care must be taken if the listboxes represent two table fields and perhaps even are related to two different tables.

```
      If UBound(aTable()) = 6 Then
```

The listbox relates to only one table and one field, as the argument array ends at Tablefield1 of the auxiliary table.

```
         stSql = "SELECT """ + aTable(6) + """ FROM """ + aTable(5) + """
ORDER BY """ + aTable(6) + """"
      ElseIf UBound(aTable()) = 7 Then
```

The listbox relates to two table fields but only one table, as the argument array ends at Tablefield2 of the auxiliary table.

```
         stSql = "SELECT """ + aTable(6) + """||' > '||""" + aTable(7) +
""" FROM """ + aTable(5) + """ ORDER BY """ + aTable(6) + """"
```

```
      Else
```

The listbox is based on two table fields from two tables. This query corresponds to the example with the postcode and the town.

```
         stSql = "SELECT """ + aTable(5) + """.""" + aTable(6) + """||' > 
'||""" + aTable(8) + """.""" + aTable(7) + """ FROM """ + aTable(5) + """, 
""" + aTable(8) + """ WHERE """ + aTable(8) + """.""" + 
stForeignIDTab2Tab1 + """ = """ + aTable(5) + """.""" + 
stForeignIDTab1Tab2 + """ ORDER BY """ + aTable(6) + """"
      End If
```

Here we have the first evaluation to determine the foreign keys. The variables stForeignIDTab2Tab1 and stForeignIDTab1Tab2 start with the value ID. For stForeignIDTab1Tab2 evaluation of the previous query yields a different value, namely the value of Town_ID. In this way the previous query construction yields exactly the content already formulated for postcode and town – only enhanced by sorting.

Now we must make contact with the listboxes, to supply them with the content returned by the queries. These listboxes do not yet exist, since the dialog itself has not yet been created. This dialog is created first in memory, using the following lines, before it is actually drawn on the screen.

```
      DialogLibraries.LoadLibrary("Standard")
      oDlg = 
CreateUnoDialog(DialogLibraries.Standard.Dialog_Table_purge)
```

Next come the settings for the fields of the dialog. Here, for example, is the listbox which is to be supplied with the results of the above query:

```
      oCtlList1 = oDlg.GetControl("ListBox1")
      oCtlList1.addItems(aContent(),0)
```

Access to the fields of the dialog is accomplished by using **`GetControl`** with the appropriate name. In dialogs it is not possible for two fields to use the same name as this would create problems when evaluating the dialog.

The listbox is supplied with the contents of the query, which have been stored in the array aContent(). The listbox contains only the content to be displayed as a field, so only the position 0 is filled.

After all fields with the desired content have been filled, the dialog is launched.

```
      Select Case oDlg.Execute()
      Case 1     'Case 1 means the "OK" button has been clicked
      Case 0 'If it was the "Cancel" button
         inRepetition = 0
      End Select
   Loop While inRepetition = 1
```

The dialog runs repeatedly as long as the value of "inRepetition" is 1. This is set by the corresponding checkbox.

Here, in brief, is the content after the "OK" button is clicked:

```
   Case 1
      stInhalt1 = oCtlList1.getSelectedItem() 'Read value from 
Listbox1 ...
```

```
    REM ... and determine the corresponding ID-value.
```

The ID value of the first listbox is stored in the variable "inLB1".

```
      stText = oCtlText.Text   ' Read the field value.
```

If the text field is not empty, the entry in the text field is handled. Neither the listbox for a replacement value nor the checkbox for deleting all orphaned records are considered. This is made clear by the fact that text entry sets these other fields to be inactive.

```
      If stText <> "" Then
```

If the text field is not empty, the new value is written in place of the old one using the previously read ID field in the table. There is the possibility of two entries, as is also the case in the listbox. The separator is >. For two entries in different tables, two UPDATE-commands must be launched, which are created here simultaneously and forwarded, separated by a semicolon.

```
      ElseIf oCtlList2.getSelectedItem() <> "" Then
```

If the text field is empty and the listbox 2 contains a value, the value from listbox 1 must be replaced by the value in listbox 2. This means that all records in the tables for which the records in the listboxes are foreign keys must be checked and, if necessary, written with an altered foreign key.

```
      stInhalt2 = oCtlList2.getSelectedItem()
      REM Read value from listbox.
      REM Determine ID for the value of the listbox.
```

The ID value of the second listbox is stored in the variable inLB2. Here too, things develop differently depending on whether one or two fields are contained in the listbox, and also on whether one or two tables are the basis of the listbox content.

The replacement process depends on which table is defined as the table which supplies the foreign key for the main table. For the above example, this is the Postcode table, as the Postcode_ID is the foreign key which is forwarded through Listbox 1 and Listbox 2.

```
   If stAuxilTable = aTable(5) Then
      For i = LBound(aForeignTables()) To UBound(aForeignTables())
```

Replacing the old ID value by the new ID value becomes problematic in n:m-relationships, as in such cases, the same value can be assigned twice. That might be what you want, but it must be prevented when the foreign key forms part of the primary key. So in the table rel_Media_Author a medium cannot have the same author twice because the primary key is constructed from Media_ID and Author_ID. In the query, all key fields are searched which collectively have the property UNIQUE or were defined as foreign keys with the UNIQUE property using an index.

So if the foreign key has the UNIQUE property and is already represented there with the desired future inLB2, that key cannot be replaced.

```
stSql = "SELECT ""COLUMN_NAME"" FROM
""INFORMATION_SCHEMA"".""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" = '" +
aForeignTables(i,0) + "' AND ""NON_UNIQUE"" = False AND ""INDEX_NAME"" =
(SELECT ""INDEX_NAME"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_INDEXINFO"" WHERE
""TABLE_NAME"" = '" + aForeignTables(i,0) + "' AND ""COLUMN_NAME"" = '" +
aForeignTables(i,1) + "')"
```

`' "NON_UNIQUE" = False '` gives the names of columns that are UNIQUE. However not all column names are needed but only those which form an index with the foreign key field. This is handled by the Subselect with the same table names (which contain the foreign key) and the names of the foreign key fields.

If now the foreign key is present in the set, the key value can only be replaced if other fields are used to define the corresponding index as UNIQUE. You must take care when carrying out replacements that the uniqueness of the index combination is not compromised.

```
   If aForeignTables(i,1) = stFieldname Then
      inUnique = 1
   Else
      ReDim Preserve aColumns(inCount)
      aColumns(inCount) = oQuery_result.getString(1)
      inCount = inCount + 1
   End If
```

All column names, apart from the known column names for foreign key fields as Index with the UNIQUE property, are stored in the array. As the column name of the foreign key field also belongs to the group, it can be used to determine whether uniqueness is to be checked during data modification.

```
If inUnique = 1 Then
   stSql = "UPDATE """ + aForeignTables(i,0) + """ AS ""a"" SET """ + 
aForeignTables(i,1) + """='" + inLB2 + "' WHERE """ + aForeignTables(i,1) 
+ """='" + inLB1 + "' AND ( SELECT COUNT(*) FROM """ + aForeignTables(i,0) 
+ """ WHERE """ + aForeignTables(i,1) + """='" + inLB2 + "' )"
   If inCount > 0 Then
      stFieldgroup = Join(aColumns(), """||  ||""")
```

If there are several fields, apart from the foreign key field, which together form a UNIQUE index, they are combined here for a SQL grouping. Otherwise only aColumns(0) appears as stFieldgroup.

```
      stFieldname = ""
      For ink = LBound(aColumns()) To UBound(aColumns())
         stFieldname = stFieldname + " AND """ + aColumns(ink) + """ = 
""a"".""" + aColumns(ink) + """ "
```

The SQL parts are combined for a correlated subquery.

```
      Next ink
   stSql = Left(stSql, Len(stSql) – 1)
```

The previous query ends with a bracket. Now further content is to be added to the subquery, so this closure must be removed again. After that, the query is expanded with the additional conditions.

```
   stSql = stSql + stFeldbezeichnung + "GROUP BY (""" + stFeldgruppe + 
""") ) < 1"
End If
```

If the foreign key has no connection with the primary key or with a UNIQUE index, it does not matter if content is duplicated.

```
Else
   stSql = "UPDATE """ + aForeignTables(i,0) + """ SET """ + 
aForeignTables(i,1) + """='" + inLB2 + "' WHERE """ + aForeignTables(i,1) 
+ """='" + inLB1 + "'"
```

```
End If
oSQL_Statement.executeQuery(stSql)
NEXT
```

The update is carried out for as long as different connections to other tables occur; that is, as long as the current table is the source of a foreign key in another table. This is the case twice over for the Town table: in the Media table and in the Postcode table.

Afterwards the old value can be deleted from listbox 1, as it no longer has any connection to other tables.

```
stSql = "DELETE FROM """ + aTable(5) + """ WHERE ""ID""='" + inLB1 + "'"
oSQL_Statement.executeQuery(stSql)
```

In some cases, the same method must now be carried out for a second table that has supplied data for the listboxes. In our example, the first table is the Postcode table and the second is the Town table.

If the text field is empty and listbox 2 also contains nothing, we check if the relevant checkbox indicates that all surplus entries are to be deleted. This means the entries which are not bound to other tables by a foreign key.

```
ElseIf oCtlCheck1.State = 1 Then
   stCondition = ""
   If stAuxilTable = aTable(5) Then
      For i = LBound(aForeignTables()) To UBound(aForeignTables())
         stCondition = stCondition + """ID"" NOT IN (SELECT """ +
aForeignTables(i,1) + """ FROM """ + aForeignTables(i,0) + """) AND "
      Next
   Else
      For i = LBound(aForeignTables2()) To UBound(aForeignTables2())
         stCondition = stCondition + """ID"" NOT IN (SELECT """ +
aForeignTables2(i,1) + """ FROM """ + aForeignTables2(i,0) + """) AND "
      Next
   End If
```

The last AND must be removed, since otherwise the delete instruction would end with AND.

```
   stCondition = Left(stCondition, Len(stCondition) - 4)    '
   stSql = "DELETE FROM """ + stAuxilTable + """ WHERE " + stCondition +
""
   oSQL_Statement.executeQuery(stSql)
```

As the table has already been purged once, the table index can be checked and optionally corrected downwards. See the subroutine described in one of the previous sections.

```
Table_index_down(stAuxilTable)
```

Afterwards, if necessary the listbox in the form from which the Table_purge dialog was called can be updated. In some cases, the whole form needs to be reread. For this purpose, the current record is determined at the beginning of the subroutine so that after the form has been refreshed, the current record can be reinstated.

```
   oDlg.endExecute()   'End dialog ...
   oDlg.Dispose()   '... and remove from storage
```

```
End Sub
```

Dialogs are terminated with the endExecute() command and completely removed from memory with Dispose().

Writing macros with Access2Base

Versions of LibreOffice from 4.2 onwards have integrated Access2Base. This library introduces a Basic layer with its specific API between the user’s code and the usual UNO interface. The provided API does not bring in itself new functionalities but, in many cases, it is more readable, concise, and easier to use than UNO.

The API looks very much like that designed by Microsoft for the Access software. Base and Access have a lot in common, but certainly not their native programming styles. Access2Base fills the gap.

English language documentation with examples can be found at www.access2base.com

To briefly illustrate how Access2Base hides the complexity of UNO:

- The (Access2Base simple) *Value* property of a control has in UNO as equivalents, depending on the control type or its location in a form, a grid control or a dialog: *CurrentValue*, *Date*, *EffectiveValue*, *HiddenValue*, *ProgressValue*, *RefValue*, *ScrollValue*, *SpinValue*, *State*, *StringItemList*, *Text*, *Time*, *ValueItemList* or … *Value*.
- To get the N first records of a table or a query into a Basic array, one method is simply to use the *GetRows(N)* method on a *Recordset* object. Compare with the *getString*, *getNull*, *getDouble*, *getLong*, … methods in UNO that you should apply on fields depending on their type and the used database system.

There are two main categories of objects handled by Access2Base, targeting either:

- The User Interface. Typical such object classes are: Form, SubForm, Dialog, Control, CommandBar, CommandBarControl and Event. Their methods are invoked usually from a Base application.
- The Database accesses. Typical such object classes are: Database, TableDef, QueryDef, Recordset and Field. Their methods are invoked either from a Base application or from any other LibreOffice application.

Traditionally, the Access2Base API is invoked from the Basic language. As from LibreOffice 6.4, a gateway provides access to the API from Python scripts as well, without any limitation vs. Basic. One may integrate seamlessly Basic and Python scripts in the same application, even by sharing the same object instances.

In the next paragraphs, every example will be given both in Basic and in Python. They are strictly equivalent.

To access the library from a Base application, attach the next procedure to the *OpenDocument* event of your Base file:

(BASIC)

```
Sub DBOpen(Optional oEvent As Object)
   If GlobalScope.BasicLibraries.hasByName("Access2Base") then
      GlobalScope.BasicLibraries.loadLibrary("Access2Base")
   End If
   Call Application.OpenConnection(ThisDatabaseDocument)
End Sub
```

(PYTHON)

```
from access2base import *
def DBOpen(event = None):
        Application.OpenConnection()
g_exportedScripts = (DBOpen, )
```

Alternatively, to gain access to the database from a non-Base application, run:

(BASIC)

```
Function DBOpen() As Object
   If GlobalScope.BasicLibraries.hasByName("Access2Base") then
      GlobalScope.BasicLibraries.loadLibrary("Access2Base")
   End If
   Set myDb = Application.OpenDatabase(" … database file name … ")
End Function
```

(PYTHON)

```
from access2base import *
def DBOpen():
   return Application.OpenDatabase(‘ … database file name … ‘)
```

It is not the intent of this book to replicate the documentation of the above-mentioned website. We will restrict this document to a summary of the main concepts of the API.

The Object Model

In Figure 440, starting from the *Application root object*, is a scheme describing the navigation through the most used objects.

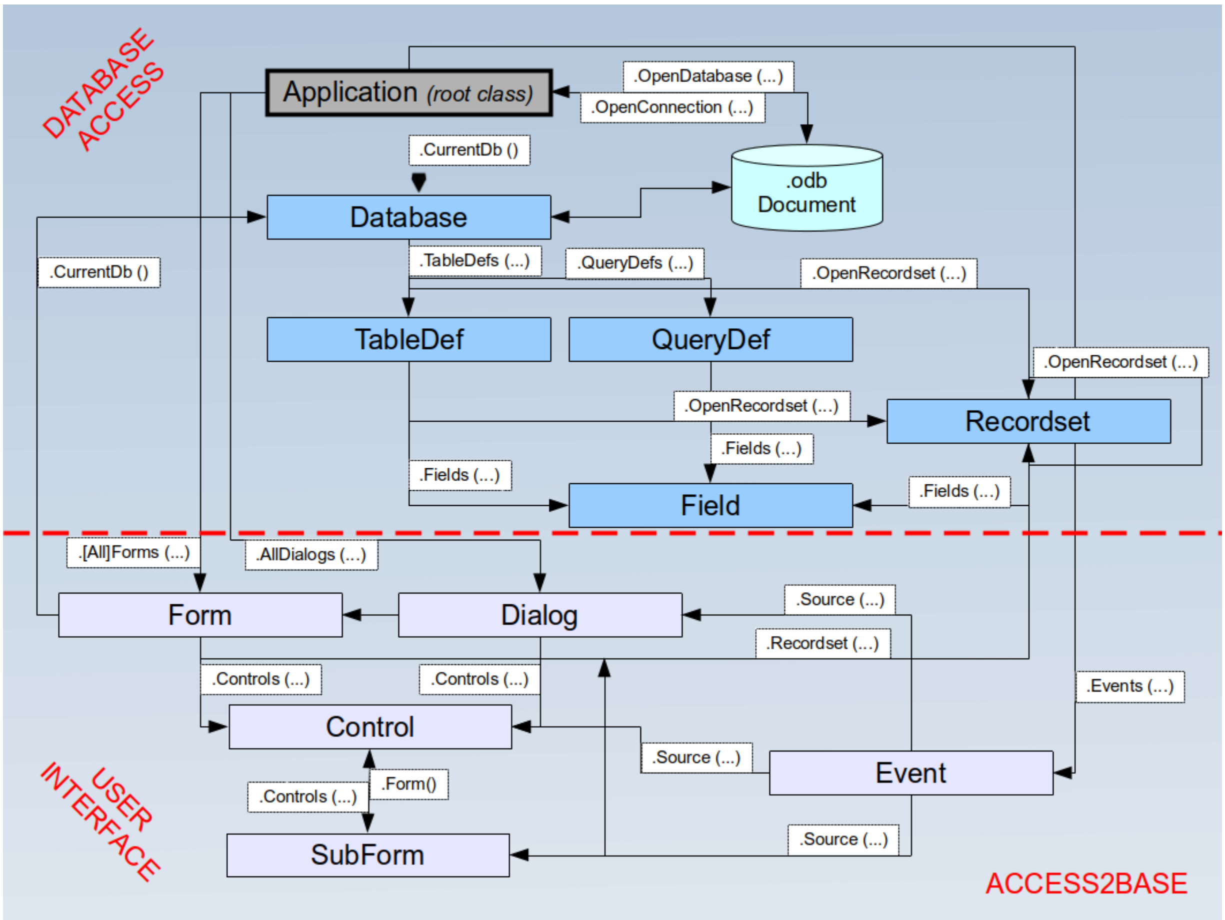


Figure 440: Access2Base Object Model

As an example, to help reading the scheme:

- Application is the principal out of two root objects.
- The *CurrentDb()* and *OpenDatabase()* methods deliver a **Database** object.
- The *Database.TableDefs()* collection lists all tables stored in the database. Each table is represented by a **TableDef** object instance.
- The *TableDef.Fields()* collection lists all fields present in the database table. Each field is represented by a **Field** object instance.
- Fields can also be the children of stored queries (**QueryDefs**) or from data sets (**Recordsets**), with the same attributes.

A few examples

Print a list of table and field names

(BASIC)

```
Sub ScanTables()
Dim oDatabase As Object, oTable As Object, oField As Object
Dim i As Integer, j As Integer
   Set oDatabase = Application.CurrentDb()
   With odatabase
      For i = 0 To .TableDefs.Count - 1
```

```
         Set oTable = .TableDefs(i)    '   Get each individual table
definition
         DebugPrint oTable.Name
         For j = 0 To oTable.Fields.Count - 1
            Set oField = oTable.Fields(j)    '   Get each individual field
            DebugPrint "", oField.Name, oField.TypeName
         Next j
      Next i
   End With
End Sub
```

(PYTHON)

```
def ScanTables():
   oDatabase = Application.OpenDatabase("/home/somedir/TT NorthWind.odb")
   for oTable in oDatabase.TableDefs():
      DebugPrint(oTable.Name)
      for oField in oTable.Fields():
         DebugPrint(“”, oField.Name, oField.TypeName)
```

Store the data produced by a query into a Basic array or a Python tuple

(BASIC)

```
Sub LoadQuery()
Dim oRecords As Object, vData As Variant
   Set oRecords = Application.CurrentDb().OpenRecordset("myQuery")
   vData = oRecords.GetRows(1000)
   orecords.mClose()
End Sub
```

(PYTHON)

```
def LoadQuery():
   oRecords= Application.CurrentDb().Openrecordset(“myQuery”)
   vData = orecords.GetRow(1000)
   oRecords.Close()
```

Set default values in form entries

To specify that after each record entry some control is prefilled with the last value set, assign the next routine to the *After Record Change* event of the form:

(BASIC)

```
Sub SetDefaultNewRec(poEvent As Object)
Dim oForm As Object, oControl As Object
   Set oForm = Application.Events(poEvent).Source    '   Get the current
form
   Set oControl = oForm.Controls("txtCountry")
   oControl.DefaultValue = oControl.Value
End Sub
```

(PYTHON)

```
def SetDefaultNewRec(poEvent):
   oForm = Application.Events(poEvent).Source
   oControl = oForm.Controls(“txtCountry”)
   oControl.DefaultValue = oControl.Value
```

Database functions

A collection of functions is provided to shorten to one single line the access to database values: *DLookup*, *DMax*, *DMin*, *Dsum*. They all accept the same arguments: a field name or an expression based on field names, a table or query name, and a SQL-where clause without the *WHERE* keyword. For example:

(BASIC)

```
Function Lookup(psField As String, psSearchField As String,
psSearchValue As String) As Variant
   Lookup = Application.DLookup(psField, "myTable", _
               psSearchField & "='" & psSearchValue & "'")
End Function
```

(PYTHON)

```
def Lookup(psField, psSearchField, psSearchValue):
   return Application.Dlookup(psField, “myTable”
      , psSearchField + “=’” + psSearchValue + “’”)
```

Special commands

The *DoCmd* (2nd root class) proposes a set of convenient functions allowing to execute in one Basic statement complex although frequent and practical actions. To name a few:

Table 50: Access2Base DoCmd class, special member functions

Function	***Description***
CopyObject	Copy a table or a query within the same database or between two databases.
OpenSQL	Execute a given SQL SELECT statement and display the result in a datasheet.
OutputTo	Store the data from a table or a query in an HTML file. Store the actual content of a form into a PDF file.
SelectObject	Activate the given window (form, report, …)
SendObject	Send by mail with the given form in attachment.

The “Basic” object in Python

From Python an additional object has been introduced in the Access2Base gateway, simply called “Basic”, to let execute by the Basic run-time a number of well-known Basic built-in function. They are invoked exactly with the same arguments and behave strictly in the same manner in both environments. Among them:

- System functions: Basic.ConvertToUrl and Basic.ConvertFromUrl, Basic.GlobalScope, Basic.CreateUnoService.
- User interface functions: Basic.MsgBox, Basic.InputBox
- Date functions: Basic.DateAdd, Basic.DateDiff
- Introspection of UNO objects: Basic.Xray

The ScriptForge Library

Macro programmers frequently need to perform tasks such as creating and opening files, accessing form controls, reading data from databases embedded in Base documents, and so forth. The objective of the ScriptForge library is to make it easier to execute such commands without having to learn the required LibreOffice APIs (Application Programming Interfaces) and commands, which may be difficult for casual programmers.

The ScriptForge library is organized into a set of services, each of which provides methods and properties related to a specific topic. For example, the Dialog service provides access to dialogs available in script modules and the Database service allows to execute SQL commands in Base documents. Many of the Methods are common across different services.

Since version 7.2 the English version of the ScriptForge libraries documentation is fully integrated in the LibreOffice help pages (https://help.libreoffice.org/7.3/en-US/text/sbasic/shared/03/lib_ScriptForge.html?DbPAR=BASIC).

Table 51: Services provided by the ScriptForge library

Category	***Services***	
LibreOffice Basic	Array Dictionary Exception	FileSystem String TextStream
Document Content	Base Calc Chart	Database Document
User Interface	Dialog DialogControl Form	FormControl PopupMenu UI
Utilities	Basic L10N Platform	Services Session Timer

Demonstration of ScriptForge with Base using Python

This section is a brief demonstration of some of the main services of interest with Base using the Python language. This demonstration closely follows the help examples which can be referred to for the basic language. A number of services are required to work with Base documents including: UI (User Interface), FileSystem, Base, Database, Form and FormControl. Other services will be used to support the demonstration.

Creating Python Scripts with ScriptForge

You need a working Python environment so use the help to work through Creating Python Scripts with ScriptForge.

The first section explains a number of differences between running ScriptForge under Basic and Python. Python has no built-in IDE in LibreOffice so it needs to be installed and the My Macros location identified. The key section is Running Scripts from inside the LibreOffice process:

1) Using the APSO extension
2) Creating Python script files

If the increment_cell macro ran you have successfully established a working Python environment with ScriptForge. If not then go back through the steps until it works.

Initialize the Demonstration

The same process can be used to develop the demonstration module. To run scripts from inside LibreOffice we need to open a document, a new untitled spreadsheet is fine as it doesn't need to be saved. Next, create the SF_Base_Demo.py Module file directly in My Macros.

It's more efficient to put common declarations at the start of the module. All the macros use the ScriptForge library so start the module by importing the CreateScriptService method so the library services can be accessed. Many of the macros display a message box and use LibreOffice application windows so define those too:

```
# ScriptForge Base Demo Module
from scriptforge import CreateScriptService
bas = CreateScriptService("Basic")
ui = CreateScriptService("UI")
```

The demonstration needs a working folder but different operating systems using different file systems. Fortunately, SourceForge takes care of that complexity with the FileSystem service. Just initiate the service and add a variable for the workpath, more details are given in the documentation.

```
fs = CreateScriptService("FileSystem")
fs.FileNaming = "SYS"
workpath = fs.BuildPath(fs.HomeFolder, "demo")
```

The first macro is a quick test that SourceForge is working properly so add the sf_HomeFolder macro below and run it to display the home folder on your system. This is the last reminder to update g_exportedScripts.

```
def sf_HomeFolder(args=None):
    bas.MsgBox("Home Folder: " + str(fs.HomeFolder))


g_exportedScripts = (sf_HomeFolder, )
```

Create, Open and Use Database

The only way to create a Base file is with the UI (User Interface) service CreateBaseDocument method . Check the SourceForge documentation for the options but note if the path does not exist then it will be created with the file and any existing file will be overwritten without warning. Run the following example:

```
def CreateBaseDocumentUI(workpath = workpath):
    doc = ui.CreateBaseDocument(fs.BuildPath(workpath, "myDB.odb"))
    bas.MsgBox("Created now closing: " + fs.BuildPath(workpath,
"myDB.odb"))
    doc.CloseDocument()
```

Once a Base file exists the UI service OpenBaseDocument method can be used to open a Base file. Run this macro:

```
def OpenBaseDocumentUI(workpath = workpath):
    ui.OpenBaseDocument(fs.BuildPath(workpath, "myDB.odb"), \
        macroexecution = ui.MACROEXECALWAYS)
```

An open base document can be activated with the UI service GetDocument method. myDB.odb should still be open so minimise it but don't close it. If it's closed just reopen it and minimise it. Run the following macro to activate it:

```
def GetDocumentUI(workpath = workpath):
    myDoc = ui.GetDocument(fs.BuildPath(workpath, "myDB.odb"))
    bas.MsgBox("Bye!")
    myDoc.CloseDocument(True)
```

The Database service also allows direct access to a database without the Base document. This gives access to tables and queries, and the data they contain. There is no access to the forms and reports in the Base document which must be accessed through Base services. Run the macro below to return data directly from the database:

```
def database(workpath = workpath):
    myDatabase = CreateScriptService("Database", fs.BuildPath(workpath,
"myDB.odb"))
    # Run queries, SQL statements,
    bas.MsgBox("Closing: " + fs.BuildPath(workpath, "myDB.odb"))
    myDatabase.CloseDatabase()
```

You won't need the myDB.odb file in the demo folder under your home folder anymore so you can delete it.

Employees Demonstration Database

Now, on with the demonstration. To use forms in the demonstration without setting them up just download employees.odb to a demo folder under your home folder. It’s time to open the database so create another macro and run it enabling macros:

```
def open_database(workpath = workpath):
    doc = ui.OpenBaseDocument(fs.BuildPath(workpath, "Employees.odb"))
    # User and password are supplied below, if needed
    myDatabase = doc.GetDatabase()
```

You might recognize employees.odb from the Getting Started with Base Guide as the Automobile.odb database with an extra form, sf_demo to run the macros. You can use it instead of APSO from here on.

Enter and run the following macro to load data into the database. This demo is not intended to teach SQL, just demonstrate that it can be run using the database service with the RunSql method. You can experiment with the data. Every time this macro is run the Drop Table command will delete the table if it exists before creating the table and loading the data.

```
# SQL Demo - Create Table with test data
def create_table_with_values(args=None):
    # Prep SQL - Drop; Create; Add data
    mysql = """
```

```
Drop Table "EmployeeData" IF EXISTS;

CREATE TABLE "EmployeeData" (
  "ID" INT  IDENTITY,
  "FirstName" VARCHAR(50),
  "LastName" VARCHAR(50),
  "Position" VARCHAR(50),
  "Salary" DECIMAL(10, 2),
  "City" VARCHAR (50)
);

-- If no Field_name specified, all fields must be completed in right order
INSERT INTO "EmployeeData"
VALUES
  (NULL, 'Paul', 'Mejia', 'Family Therapist', 50000, 'Hogworth');

INSERT INTO "EmployeeData"
VALUES
  (NULL, 'Jordin', 'Wise', 'Sales', 83954, 'Springstead');

INSERT INTO "EmployeeData"
VALUES
  (NULL, 'Kaitlin', 'Palmer', 'Manager', 83250, 'Chicago');

INSERT INTO "EmployeeData"
VALUES
  (NULL, 'Paul', 'Smith', 'Mechanic', 74600, 'Goldview');

INSERT INTO "EmployeeData"
VALUES
  (NULL, 'Conrad', 'Palmer', 'Manager', 115000, 'North Dodgestead');

INSERT INTO "EmployeeData"
VALUES
  (NULL, 'Kate', 'Wordsmith', 'Sales', 75000, 'Chicago');
 """
    # get base document
    # doc = CreateScriptService("Document", bas.ThisDatabaseDocument)
    doc = ui.GetDocument(bas.ThisDatabaseDocument)
    # Creates Database service with current database
    myDatabase = doc.GetDatabase()
    # Runs the SQL
    myDatabase.RunSql(mysql, directsql=True)
    # does not work
    doc.RunCommand("DBRefreshTables")
    myDatabase.CloseDatabase()
    # Display message
    bas.MsgBox("Table Created")
```

Statistics are commonly needed from databases so let's use a few of the Database methods on the Employees table. First get access to the data just loaded, use **View > Refresh Tables**.

```
# Calculate employee stats
def table_stats(args=None):
    # Creates Database service with current database
    doc = ui.GetDocument(bas.ThisDatabaseDocument)
    Employees = doc.GetDatabase()
    # For alternative string display
    item_a = "Employee Count"
    # Method Dcount
    item_b = str(Employees.DCount("[ID]", "EmployeeData"))
    bas.MsgBox("Employee Count" + "\n" + item_b.center(len(item_a), "_"))
    # Method DSum – Selective
    bas.MsgBox("Total Managers Salary - " + str(Employees.Dsum( \
        "[Salary]", "EmployeeData", "[Position] = 'Manager'")))
```

Details on a single record can be returned with the Dlookup service. Note it only returns the first record if multiple records are returned in the query.

```
def get_first_match(args=None):
    # Creates Database service with current database
    myDB = CreateScriptService("Database", bas.ThisDatabaseDocument.URL, 
"", False)
    # Method Dlookup
    bas.MsgBox(
        "Select First Occurance FirstName with" + "\n"
        + "Descending Seq of FirstName" + "\n"
        + "and LastName like Smith" + "\n\n"
        + (myDB.DLookup("[FirstName]", "EmployeeData",      \
          Criteria = "[LastName] LIKE LOWER ('%Smith%')", \
          OrderClause = "[FirstName] DESC")))
```

Finally, the Database service allows the whole table to be read into an array with the GetRows method.

```
# Get employee data records
def get_employee_records(args=None):
    # Creates Database service with current database
    doc = ui.GetDocument(bas.ThisDatabaseDocument)
    Employees = doc.GetDatabase()
    # queryResults is a returned Array
    # header is column titles
    queryResults = Employees.GetRows("EmployeeData", header=True)
    bas.MsgBox(
        "Retrieve table data; Show:" + "\n"
        + "FirstName of first record" + "\n\n"
        + (queryResults[1][1])
    )
    # queryResults is a returned Array
    queryResults = Employees.GetRows(
        "SELECT * FROM EmployeeData ORDER BY [FirstName]", maxrows=5
    )
    bas.MsgBox(
        "Retrieve table data; Show:" + "\n"
        + "FirstName of first record" + "\n"
        + "Select statement ordered by FirstName" + "\n\n"
        + (queryResults[0][1])
    )
```

Most of the SourceForge Base services relate to forms. Forms are used when direct entry into a table is inconvenient, to pick up errors in data entry promptly, or when too many tables make direct management of data impossible. The next macro demonstrates using a form with a subform:

```
# get_form and sub form - display data from each
def get_form(args=None):
    # Creates DB service using current document
    doc = CreateScriptService("Document", bas.ThisDatabaseDocument)
    # Statement below only necessary if form hasn't been opened yet
    form_doc = doc.OpenFormDocument("Fuel")
    # Forms - In document `Fuel` access `Mainform`
    form = doc.Forms("Fuel", "MainForm")
    # get control on `MainForm`
    form_control = form.Controls("fmtOdometer")
    # Display value of field
    bas.MsgBox("Odometer - " + str(form_control.Value))
    # Get sub form using main form
    subform = form.Subforms("SubForm")
    # Get table grid control
    subform_control = subform.Controls("SubForm_Grid")
    # Get column in grid control
    grid_control = subform_control.Controls("FuelCost")
    # Display value of field in selected record
    bas.MsgBox("First FuelCost in Grid - " + str(grid_control.Value))
    # Close form
    form.CloseFormDocument()

def get_form2(args=None):
    bas = CreateScriptService("Basic")
    # Creates DB service using current document
    doc = CreateScriptService("Document", bas.ThisDatabaseDocument)
    # Statement below only necessary if form hasn't been opened yet
    form_doc = doc.OpenFormDocument("Fuel")
    # Forms - In document `Fuel` access `Mainform`
    form = doc.Forms(thisFormDocument, "MainForm")
    # get control on `MainForm`
    form_control = form.Controls("fmtOdometer")
```

The FormEvent service can can be used to access to the FormControl details that triggered the event. The next macro is assigned to a form button and run with an execute action. Note the form document is not directly available in ScriptForge but the name is, and it is used to find the form document.

```
# close form
def close_form(evt): # assign form button execute action event to macro
    control = CreateScriptService("FormEvent", evt) # the button
    form = control.Parent
    fdoc_name = form.BaseForm if CreateScriptService("Platform" \
    ).OfficeVersion.split()[1] >= "7.3.2" else "sf_demo"
    doc = CreateScriptService("Document", bas.ThisDatabaseDocument)
    fdoc = doc.Forms(fdoc_name, form.Name)
    # Close form
    fdoc.CloseFormDocument()
```

The document can be closed directly:

```
# close current document
def close_doc(args=None):
    # Creates DB service using current document
    doc = CreateScriptService("Document", bas.ThisDatabaseDocument)
    # Close current document
    doc.CloseDocument(True)
```

Further Resources

Refer to the ScriptForge documentation for details of the services and more examples or use LibreOffice help resources. You can download a more complete version of this demonstration.

Base Guide 7.3

Chapter 10
Database Maintenance

General remarks on maintaining databases

Dynamic databases—especially in those that frequently deletes and changes data—have two negative effects of concern. First, the database grows steadily even though it may not actually contain more data. Second, the automatically created primary key continues to increment regardless of what value of the next needed key is. This chapter describes important database maintenance tasks to consider for a good peformance and management.

Compacting a database

The behavior of HSQLDB databases is to keep storage space even for deleted records. Databases that are filled with test data, especially images, retain the same size even if all these records are subsequently deleted. The reason is a property of each table’s primary keys. The database document file contains the last value used for each primary key. When a record is created within a table, the next key value is assigned.

To free up this storage space, the database records must be rewritten (tables, table descriptions, etc). This can be done by opening each table and deleting all of its records. Care must be taken when dealing with linked tables.

Use **Tools > Relationships** to determine which table should have its data deleted. Look at the two tables. The one with its primary key being part of the relationship is the table whose data needs to be deleted. Close the Relationships dialog. Select the Tables icon in the main database window. Then double-click the table to show its data. Delete its data. Save the table and then the database. After doing this, these changes need to be written to the database document file. To do this, close LibreOffice. This will also compact the database files.

Close LibreOffice and reopen it if you are going to use it again.

Resetting autovalues

When creating a database, and then test all possible functions with examples, and made corrections until everything works, you will likely have the primary keys with values well above the initial value before the database is put into production. Often, primary keys are actually set to auto-increment. If the tables are emptied in preparation for normal usage or prior to handing the database on to another person, the primary key continues to increment from its current position instead of resetting itself to zero.

The following SQL command, entered using **Tools > SQL**, lets you reset the initial value:

```
ALTER TABLE "Table_name" ALTER COLUMN "ID" RESTART WITH New value
```

This assumes that the primary key field has the name ID and has been defined as an autovalue field. The new value should be the one that you want to be automatically created for the next new record. So, for example, if current records go up to 4, the new value should be 5 without altering the ID field. The first ID value will be the *New value* in the SQL statement above.

Querying database properties

All information on the tables of the database is stored in table form in a separate part of HSQLDB. This separate area can be reached using the name INFORMATION_SCHEMA.

The following query can be used to find out field names, field types, column sizes, and default values. Here is an example for a table named Searchtable.

```
SELECT "COLUMN_NAME", "TYPE_NAME", "COLUMN_SIZE", "COLUMN_DEF" AS "Default
Value" FROM "INFORMATION_SCHEMA"."SYSTEM_COLUMNS" WHERE "TABLE_NAME" =
'Searchtable' ORDER BY "ORDINAL_POSITION"
```

All special tables in HSQLDB are described in Appendix A of this book. Information on the content of these tables is most easily obtained by direct queries:

```
SELECT * FROM "INFORMATION_SCHEMA"."SYSTEM_PRIMARYKEYS"
```

The asterisk ensures that all available columns of the table are shown. The table searched for above gives essential information on the primary keys of the various tables.

This information is useful above all for macros. Instead of having to provide detailed information on each freshly created table or database, procedures are written to fetch this information directly out of the database and are therefore universally applicable. The example database shows this, among other things, in one of the maintenance modules, where foreign keys are determined.

Exporting data

There is a much simpler method of exporting data other than the standard method of exporting data by opening the *.odb file. Directly at the Base interface, you can use **Tools > SQL** to enter a simple command that, in server databases, is reserved for the system administrator.

```
SCRIPT 'my_exported_database_file'
```

This creates a complete SQL extraction of the database with all table definitions, relationships between tables, and records. Queries and forms are not extracted since they were created in the user interface and are not stored in the internal database. However all views are included.

Note

This procedure can be used to update an embedded database for connecting to the database with HSQLDB 2.50. Again, queries and forms have to be replaced.

By default, the exported file is a normal text file with filename 'my_exported_database_file'. The file can also be provided in binary or compressed (zipped form), which is useful for large databases. However, this makes re-importing it back into LibreOffice Base somewhat more complicated, see below.

The format of the exported file can be changed using one of the option values of the SCRIPTFORMAT setting:

```
SET SCRIPTFORMAT {TEXT | BINARY | COMPRESSED};
```

To export the file requires using this SQL code one line at a time:

```
SCRIPT ‘my_exported_database_file’;
SET SCRIPTFORMAT TEXT:
SHUTDOWN SCRIPT;
CHECKPOINT;
```

This exports the text file *my_exported_database_file* in the home folder with the database information.

Note

Ensure '*my_exported_database_file*' does not exist in the folder or an error message will appear.

The file can be imported using **Tools > SQL** and creating a new database with the same data. In the case of a LibreOffice internal database, the following lines must be removed from '*my_exported_database_file*' before import:

```
CREATE SCHEMA PUBLIC AUTHORIZATION DBA
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 60
SET SCHEMA PUBLIC
```

These entries deal with the user profile and other default settings, which are already set for LibreOffice internal databases. As a result, an error message appears if any of these lines are present. They are found directly before the contents that will be inserted into the tables using the `INSERT` command.

To import this file, the contents needs to be split into multiple text files created by a simple text editing program. The first file should contain all of the Create Tables and Views. Copy all the lines from the beginning with `CREATE TABLE` and stopping one line above the line containing `INSERT INTO`. Paste this into the first file. Copy and paste the remaining lines into the second file.

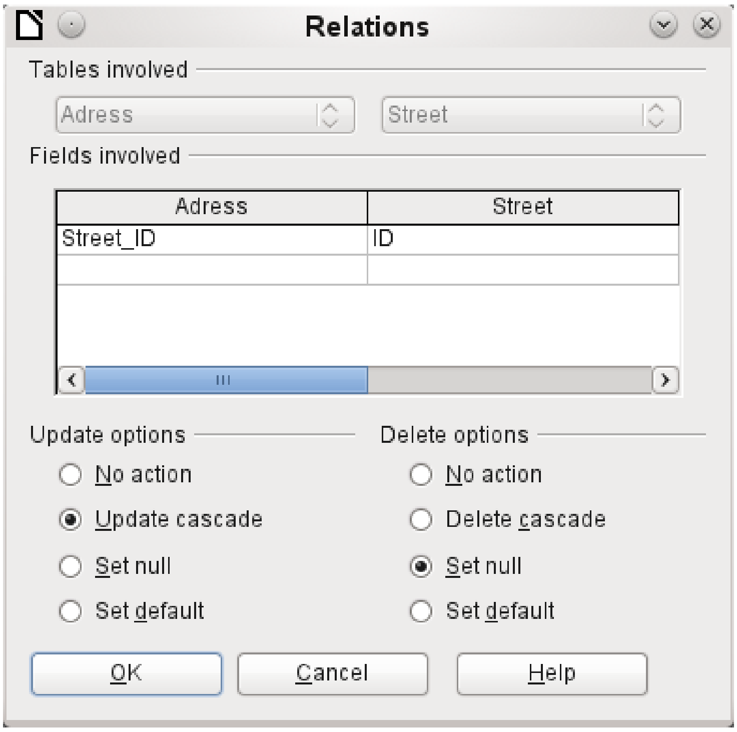
However, there is a limit to the size of second file: it must be smaller than 65KB. If it is larger than that, it too should be split into smaller text files by cutting and pasting. Just make sure that the top line of each of these new files begins with `INSERT INTO`. One way to do this is to cut from the bottom up to such a line.

Testing tables for unnecessary entries

A database consists of one or more main tables, which contain foreign keys from other tables. In the example database, these are the Media and Address tables. In the Address table the primary key of the postcode occurs as a foreign key. If a person moves to a new home, the address gets changed. The result may be that no foreign key Postcode_ID corresponding to this postcode exists any longer. In principle therefore, the postcode itself could be deleted. However, it is not apparent during normal usage that the record is no longer needed. There are various ways to prevent this sort of problem arising.

Testing entries using the relationship definition

The integrity of the data can be ensured while defining relationships. In other words, you can prevent the deletion or change of keys from leading to errors in the database. The following dialog is accessible through **Tools > Relationships**, followed by a right-click on the connector between two tables.



Relations

Tables involved

Adress — Street

Fields involved

Adress	Street
Street_ID	ID

Update options

○ No action
● Update cascade
○ Set null
○ Set default

Delete options

○ No action
○ Delete cascade
● Set null
○ Set default

Figure 441: Setting update and delete options on the Relations dialog

Here the tables Address and Street are considered. *All specified actions apply to the Address table*, which contains the foreign key Street_ID. Update options refer to an update of the ID field in the Street table. If the numeric key in the "Street"."ID" field is altered, **No action** means that the database resists this change if a "Street"."ID" with that key number occurs as a foreign key in the Address table.

Update cascade means that the key number is simply carried over. If the street 'Burgring' in the Street table has the ID ‘3' and is also represented in "Address"."Street_ID", the ID can be safely altered. For example, if it is changed to '67', the corresponding "Address"."Street_ID" values will be automatically be changed to '67'.

If **Set null** is chosen, altering the ID makes "Address"."Street_ID" an empty field.

The Delete options are handled similarly.

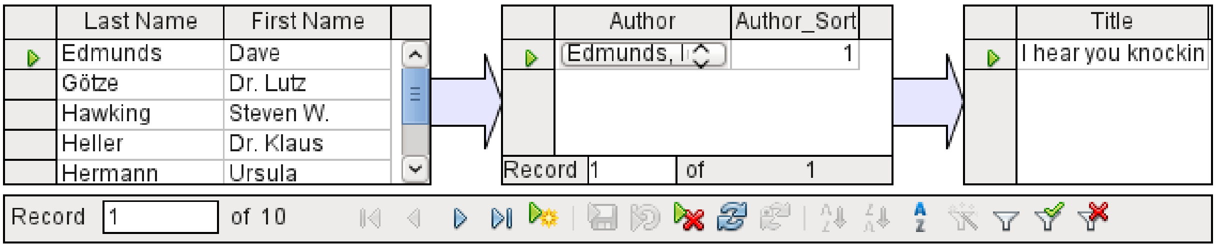
For both options, the GUI currently does not allow the possibility **Set default**, as the GUI default settings are different from those of the database. See Chapter 3, Tables.

Defining relationships helps keep the relationships themselves clean, but it does not remove unnecessary records that provide their primary key as a foreign key in the relationship. There may be any number of streets without corresponding addresses.

Editing entries using forms and subforms

In principle the whole interrelationship between tables can be displayed within forms. This is easiest of course when a table is related to only one other table. Thus in the following example, the author’s primary key becomes the foreign key in the table rel_Media_Author. rel_Media_Author also contains a foreign key from Media, so that the following arrangement shows an n:m relationship with three forms. Each is presented through a table.

Figure 442 shows that the title *I hear you knocking* belongs to the author *Dave Edmunds*. Therefore *Dave Edmunds* must not be deleted – otherwise information required for the media *I hear you knocking will be missing*. However the listbox allows you to choose a different record instead of *Dave Edmunds.*



Last Name	First Name
Edmunds	Dave
Götze	Dr. Lutz
Hawking	Steven W.
Heller	Dr. Klaus
Hermann	Ursula

Author	Author_Sort
Edmunds, I	1

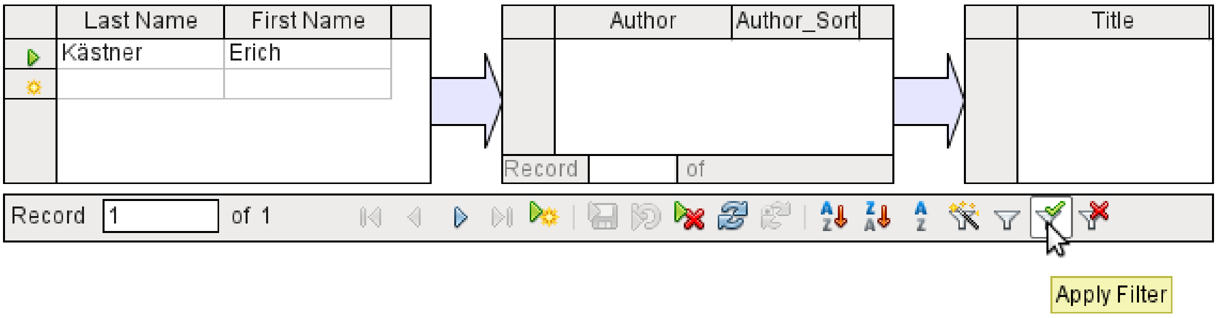
Record 1 of 1

Title
I hear you knockin

Record 1 of 10

Figure 442: Entry for Dave Edmunds should not be deleted from Author table

In the form there is a built-in filter whose activation can tell you which categories are not needed in the Media table. In the case just described, almost all of the sample authors are in use. Only the Erich Kästner record can be deleted without any consequences for any other record in Media.



Last Name	First Name
Kästner	Erich

Author	Author_Sort

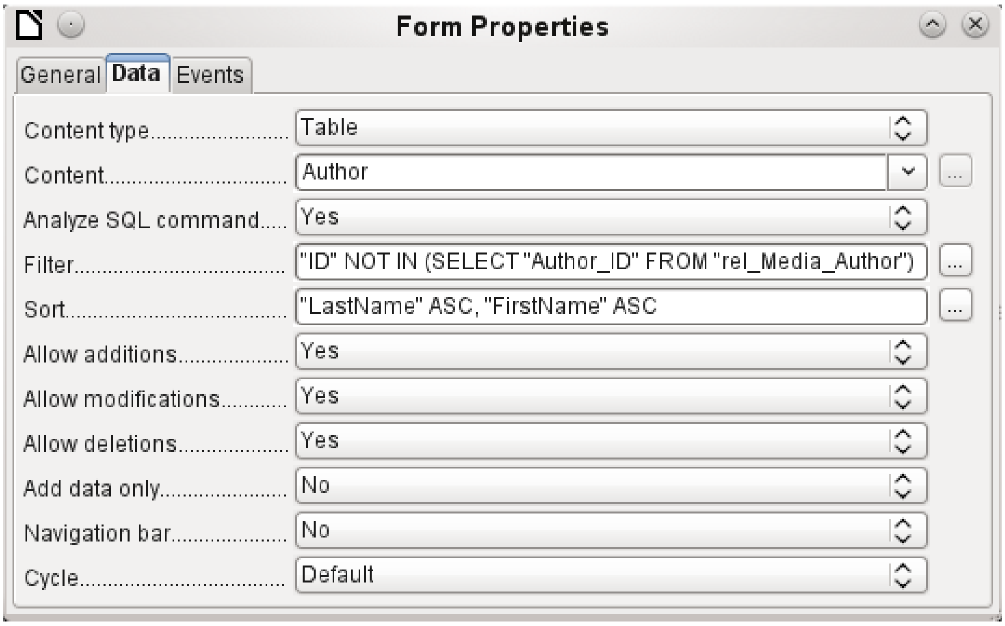
Record of

Title

Record 1 of 1

Figure 443: Entry for Erich Kästner could be deleted from Author table

The filter is hard-coded in this case. It is found in the form properties. Such a filter is activated automatically when the form is launched. It can be switched off and on. If it is deleted, it can be accessed again by a complete reload of the form. This means more than just updating the data; the whole form document must be closed and then reopened.



Form Properties

Content type	Table
Content	Author
Analyze SQL command	Yes
Filter	"ID" NOT IN (SELECT "Author_ID" FROM "rel_Media_Author")
Sort	"LastName" ASC, "FirstName" ASC
Allow additions	Yes
Allow modifications	Yes
Allow deletions	Yes
Add data only	No
Navigation bar	No
Cycle	Default

Figure 444: Filter field on Data tab of Form Properties dialog

Queries for finding orphan entries

The filter shown in Figure 444 is part of a query which can be used to find orphaned entries.

```
SELECT "Surname", "Firstname" FROM "Author" WHERE "ID" NOT IN (SELECT
"Author_ID" FROM "rel_Media_Author")
```

If a table contains foreign keys from several other tables, the query needs to be extended accordingly. This affects, for example, the Town table, which has foreign keys in both the Media table and the Postcode table. Therefore, records in the Town table which are to be deleted should not be referenced in either of these tables. This is determined by the following query:

```
SELECT "Town" FROM "Town" WHERE "ID" NOT IN (SELECT "Town_ID" FROM
"Media") AND "ID" NOT IN (SELECT "Town_ID" FROM "Postcode")
```

Orphaned entries can then be deleted by selecting all the entries that pass the set filter, and using the Delete option in the context menu of the record pointer, called up by right-clicking.

Database search speed

Effect of queries

It is just these queries, used in the previous section to filter data, that prove unsatisfactory in regard to the maximum speed of searching a database. The problem is that in large databases, the subquery retrieves a correspondingly large amount of data with which each single displayable record must be compared. Only comparisons with the relationship IN make it possible to compare a single value with a set of values. The query

```
… WHERE "ID" NOT IN (SELECT "Author_ID" FROM "rel_Media_Author")
```

can contain a large number of possible foreign keys from the rel_Media_Author table, which must first be compared with the primary keys in the Authors table for each record in that table. Such a query is therefore not suitable for daily use but may be required for database maintenance. For daily use, search functions need to be constructed differently so that the search for data is not excessively long and does not damage day-to-day work with the database.

Effect of listboxes and comboboxes

The more listboxes that are built into a form, and the more data they contain, the longer the form takes to load, since these listboxes must be fetched and created.

The better a Base program sets up the graphical interface and initially reads the listbox contents only partially, the faster it will run

Listboxes are created using queries, and these queries must be run when the form is loaded for each listbox.

When the same query structure is used for several listboxes, it is better to use a common View, rather than repeatedly creating fields with the same syntax using the stored SQL commands in the listboxes. Views are above all preferable for external database systems, as here the server runs significantly faster than a query which has to be put together by the GUI and freshly put to the server. The server treats Views as completed local queries.

Influence of the database system used

The internal HSQLDB database is set up to ensure that Base and Java work well together. When Base is using an embedded database management system, such as the HSQLDB database engine, the size of, and responsiveness of your database is limited when compared to use of an external database engine. Particularly when that database server is run on a separate computer. Should your database functions begin to slow down, first follow the steps in this guide for cleaning away empty space, deleted or temporary data and check that your are using indexes where they make sense. If responsiveness does not pay, consider moving your data from the Base ODB data to an external database server.

External databases run significantly faster. Direct connections to MySQL or PostgreSQL, and connections using ODBC, run at practically the same speed. JDBC also depends on cooperation with Java, but still works faster than an internal connection using HSQLDB.

Base Guide 7.3

Appendix A
Common Database Tasks

Introduction

As your database grows and becomes more integrated in your business, it will be necessary to deal with the administrative tasks to keep your database in a sane and safe conditions and to let you get the best of it. This chapter presents some under-the-hood information about the databases data types, functions and stored procedures, Base UNO commands, administrative and repair procedures.

Barcodes

To be able to use the barcode print function, the font `ean13.ttf` must be installed. This font is freely available. For example, as in the following address returned by a search engine: https://graphicore.github.io/librebarcode/documentation/ean13.html

EAN13 barcodes can be created using `ean13.ttf` as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number	Upper case, A=0 B=1 etc.						*	Lower case, a=0 b=1 etc.						+

See also the query `Barcode_EAN13_ttf_command` in the example database Media_without_Macros.

Data types for the table editor

Table 52: Integer data types

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Tiny Integer	TINYINT	TINYINT	2^{8} = 256 | – 128 to + 127	1 Byte
Small Integer	SMALLINT	SMALLINT	2^{16} = 65536 | – 32768 to + 32767	2 Bytes
Integer	INTEGER	INTEGER | INT	2^{32} = 4294967296 | – 2147483648 to + 2147483647	4 Bytes
BigInt	BIGINT	BIGINT	2^{64}	8 Bytes

Table 53: Floating-point data types

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Decimal	DECIMAL	DECIMAL	Unlimited, up to 50 places in the GUI, fixed decimal point, perfect accuracy	variable
Number	NUMERIC	NUMERIC	Unlimited, up to 50 places in the GUI, fixed decimal point, perfect accuracy	variable

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Float	FLOAT	(DOUBLE used instead)		
Real	REAL	REAL		
Double	DOUBLE	DOUBLE [PRECISION] | FLOAT	Adjustable, not exact, 15 decimal places maximum	8 Bytes

Table 54: Text data types

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Text	VARCHAR	VARCHAR	Adjustable	variable
Text	VARCHAR_ IGNORECASE	VARCHAR_ IGNORECASE	Adjustable, range affects sorting	variable
Text (fix)	CHAR	CHAR | CHARACTER	Adjustable, rest of actual text replaced with spaces	fixed
Memo	LONGVARCHAR	LONGVARCHAR		variable

Table 55: Date and time data types

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Date	DATE	DATE		4 Bytes
Time	TIME	TIME		4 Bytes
Date/ Time	TIMESTAMP	TIMESTAMP | DATETIME	Adjustable (0.6 – 6 means with milliseconds)	8 Bytes

Table 56: Other data types

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Yes/No	BOOLEAN	BOOLEAN | BIT		
Binaryfield (fix)	BINARY	BINARY	Like Integer	fixed
Binary field	VARBINARY	VARBINARY	Like Integer	variable

Type	***Option***	***HSQLDB***	***Range***	***Storage space***
Image	LONGVARBINARY	LONGVARBINARY	Like Integer	variable, intended for larger images
OTHER	OTHER	OTHER | OBJECT		

In the table definitions, and when data types are changed in queries using the “convert” or “cast” functions, some data types expect information about the number of characters (a), the precision (g, corresponding to the total number of characters) and the number of decimal places (d). The types are CHAR(a), VARCHAR(a), DOUBLE(g), NUMERIC(g, d), DECIMAL(g, d) and TIMESTAMP(g).

TIMESTAMP(g) can have only two values: ‘0’ and ‘6’. ‘0’ means that no seconds will be stored in the decimal part (tenths, hundredths...). The precision of timestamps can be given only *directly using SQL commands*. So if you are storing timings from some kind of sport, you must set TIMESTAMP(6) using **Tools > SQL** in advance.

Data types in LibreOffice Basic

Table 57: Numeric data types

Type	***Corresponds to HSQLDB***	***Initial value***	***Remarks***	***Storage requirements***
Integer	SMALLINT	0	2^{16} = – 32768 to + 32767	2 Bytes
Long	INTEGER	0	2^{32} = – 2147483648 to + 2147483647	4 Bytes
Single		0.0	Decimal: .	4 Bytes
Double	DOUBLE	0.0	Decimal: .	8 Bytes
Currency	Resembles DECIMAL, NUMERIC	0.0000	4 fixed decimal places	8 Bytes

Table 58: Other data types

Type	***Corresponds to HSQLDB***	***Initial value***	***Remarks***	***Storage requirements***
Boolean	BOOLEAN	False	1 = yes, everything else: no.	1 Byte
Date	TIMESTAMP	00:00:00	Date and time	8 Bytes
String	VARCHAR	Empty String	Up to 65536 characters	variable
Object	OTHER	Null		variable

Type	*Corresponds to HSQLDB*	*Initial value*	*Remarks*	*Storage requirements*
Variant		Empty	Can accept any (other) data type	variable

There are great risks in data conversion, especially with numeric values. For example, primary keys in databases are most commonly of the type INTEGER. If these are read out by a macro, the variable in which they are stored must be of the type Long, as this corresponds in size to the INTEGER type in Base. The corresponding read instruction is `getLong`.

Built-in functions and stored procedures

The following functions are available in the built-in HSQLDB. However, one or two functions can only be used when **Run SQL command directly** is chosen. This will then prevent these queries from being edited.

Functions that work with the graphical user interface are marked [Works in the GUI]. Functions that work only in direct SQL commands are marked [Direct SQL – does not work in the GUI].

Numeric functions

Note

As we are dealing here with floating point numbers, be sure to take care with the settings of the fields in queries. Mostly the display of decimal places is restricted, so that in some cases there may be unexpected results. For example, column 1 might show 0.00 but actually contain 0.001, and column 2, 1000. If column 3 is set to show Column 1 * Column 2, it would actually show 1.

Table 59: Numeric functions

Function	*Description*
ABS(d)	Returns the absolute value of a number. [Works in the GUI]
ACOS(d)	Returns the arccosine. [Works in the GUI]
ASIN(d)	Returns the arcsine. [Works in the GUI]
ATAN(d)	Returns the arctangent. [Works in the GUI]
ATAN2(a,b)	Returns the arctangent using coordinates, where a is the value of the x-axis, b the value of the y-axis. [Works in the GUI]
BITAND(a,b)	Both the binary form of a and the binary form of b must have 1 at the same position to yield 1 in the result. BITAND(3,5) yields 1; 0011 AND 0101 = 0001 [Works in the GUI]
BITOR(a,b)	Either the binary form of a or the binary form of b must have 1 at the same position to yield 1 in the result. BITOR(3,5) yields 7; 0011 OR 0101 = 0111 [Works in the GUI]

CEILING(d)	Returns the smallest whole number that is not smaller than d. [Works in the GUI]
COS(d)	Returns the cosine. [Works in the GUI]
COT(d)	Returns the cotangent. [Works in the GUI]
DEGREES(d)	Converts radians to degrees. [Works in the GUI]
EXP(d)	Returns e^d (e: (2.718...)). [Works in the GUI]
FLOOR(d)	Returns the largest whole number that is not greater than d. [Works in the GUI]
LOG(d)	Returns the natural logarithm to base e. [Works in the GUI]
LOG10(d)	Returns the logarithm to base 10. [Works in the GUI]
MOD(a,b)	Returns the remainder as a whole number, in the division of 2 whole numbers. MOD(11,3) returns 2, because 3*3+2=11 [Works in the GUI]
PI()	Returns π (3.1415...). [Works in the GUI]
POWER(a,b)	a^b , POWER(2,3) = 8, since 2^3 = 8 [Works in the GUI]
RADIANS(d)	Converts degrees to radians. [Works in the GUI]
RAND()	Returns a random number greater than or equal to 0.0 and less than 1.0. [Works in the GUI]
ROUND(a,b)	Rounds a to b decimal places. [Works in the GUI]
ROUNDMAGIC(d)	Solves rounding problems that arise from using floating point numbers. 3.11-3.1-0.01 is not exactly 0, but is shown as 0 in the GUI. ROUNDMAGIC makes it an actual zero value. [Works in the GUI]
SIGN(d)	Returns –1 if d is less than 0, 0 if d is equal to 0, and 1 if d is greater than 0. [Works in the GUI]
SIN(A)	Returns the sine of an angle in radians. [Works in the GUI]
SQRT(d)	Returns the square root. [Works in the GUI]
TAN(A)	Returns the tangent of an angle in radians. [Works in the GUI]

TRUNCATE(a,b)	Truncates a to b decimal places. TRUNCATE(2.37456,2) = 2.37 [Works in the GUI]

Text Functions

Table 60: Text functions

Function	***Description***
ASCII(s)	Returns the ASCII code of the first letter of the string. [Works in the GUI]
BIT_LENGTH(str)	Returns the length of the text string str in bits. [Works in the GUI]
CHAR(c)	Returns the letter corresponding to the ASCII code c. [Works in the GUI]
CHAR_LENGTH(str)	Returns the length of the string str in characters. [Works in the GUI]
CONCAT(str1,str2)	Concatenates str1 and str2. [Works in the GUI]
'str1'||'str2'||'str3' or 'str1'+'str2'+'str3'	Concatenates str1, str2, and str3. A simpler alternative to CONCAT. [Works in the GUI]
DIFFERENCE(s1,s2)	Returns the sound difference between s1 and s2. Only a whole number is output. 0 means they sound the same. For example, 'for' and 'four' yield 0, ‘king’ and ‘wing’ yield 1, ‘see’ and ‘sea’ yield 0. [Works in the GUI]
HEXTORAW(s1)	Translates hexadecimal code to other characters. [Works in the GUI]
INSERT(s,start,len,s2)	Returns a text string, with part of the text replaced. Beginning with start, a length len is cut out of the text s and replaced by the text s2. INSERT(“Bundesbahn”, 3, 4, mmel) converts Bundesbahn into Bummelbahn, where the length of the inserted text can be greater than that of the deleted text without causing any problems. So INSERT(“Bundesbahn”, 3, 5, s und B) yields 'Bus und Bahn'. [Works in the GUI]
LCASE(s)	Converts a string to lower case. [Works in the GUI]
LEFT(s,count)	Returns the number of characters specified by count from the beginning of the string s. [Works in the GUI]
LENGTH(s)	Returns the length of string s in characters. [Works in the GUI]
LOCATE(search,s,[start])	Returns the first match for the term search in the string s. The match is given as an offset number: (1=left, 0=not found). Setting a starting point within the text string is optional. [Works in the GUI]

Function	***Description***
LTRIM(s)	Removes leading spaces and non-printing characters from the beginning of a text string. [Works in the GUI]
OCTET_LENGTH(str)	Returns the length of a text string in bytes. This corresponds to twice the length of the string. [Works in the GUI]
RAWTOHEX(s1)	Converts to hexadecimals, reverse of HEXTORAW(). [Works in the GUI]
REPEAT(s,count)	Repeats the text string s count times. [Works in the GUI]
REPLACE(s,replace,s2)	Replaces all existing occurrences of replace in the text string s by the string s2. [Works in the GUI]
RIGHT(s,count)	Opposite of LEFT; returns the last count characters at the end of a text string. [Works in the GUI]
RTRIM(s)	Removes all spaces and non-printing characters from the end of a text string. [Works in the GUI]
SOUNDEX(s)	Returns a 4-character code, corresponding to the sound of s. Matches the function DIFFERENCE(). [Works in the GUI]
SPACE(count)	Returns count spaces. [Works in the GUI]
SUBSTR(s,start[,len])	Abbreviation for SUBSTRING. [Works in the GUI]
SUBSTRING(s,start[,len])	Returns the text s from the start position (1=left). If length len is left out, the whole string is returned. [Works in the GUI]
UCASE(s)	Converts a string to upper case. [Works in the GUI]
LOWER(s)	As LCASE(s). [Works in the GUI]
UPPER(s)	As UCASE(s). [Works in the GUI]

Date and time functions

Table 61: Date/time functions

Function	***Description***
CURDATE()	Returns the current date. [Works in the GUI]
CURTIME()	Returns the current time. [Works in the GUI]

Function	***Description***
DATEDIFF(string, datetime1, datetime2)	Date difference between two dates - compares date/time values. The entry in string determines the units in which the difference is returned: ms=millisecond, ss=second, mi=minute, hh=hour, dd=day, mm=month, yy = year. Both the long and the short forms can be used for string. [Works in the GUI]
DAY(date)	Returns the day of the month (1-31). [Works in the GUI]
DAYNAME(date)	Returns the English name of the day. [Works in the GUI]
DAYOFMONTH(date)	Returns the day of the month (1-31). Synonym for DAY(). [Works in the GUI]
DAYOFWEEK(date)	Returns the weekday as a number (1 represents Sunday). [Works in the GUI]
DAYOFYEAR(date)	Returns the day of the year (1-366). [Works in the GUI]
HOUR(time)	Returns the hour (0-23). [Works in the GUI]
MINUTE(time)	Returns the minute (0-59). [Works in the GUI]
MONTH(date)	Returns the month (1-12). [Works in the GUI]
MONTHNAME(date)	Returns the English name of the month. [Works in the GUI]
NOW()	Returns the current date and the current time together as a timestamp. Alternatively CURRENT_TIMESTAMP can be used. [Works in the GUI]
QUARTER(date)	Returns the quarter of the year (1-4). [Works in the GUI]
SECOND(time)	Returns the seconds part of the time (0-59). [Works in the GUI]
WEEK(date)	Returns the week of the year (1-53). [Works in the GUI]
YEAR(date)	Returns the year part of a date entry. [Works in the GUI]
CURRENT_DATE	Synonym for CURDATE(), SQL-Standard. [Works in the GUI]
CURRENT_TIME	Synonym for CURTIME(), SQL-Standard. [Works in the GUI]
CURRENT_TIMESTAMP	Synonym for NOW(), SQL-Standard. [Works in the GUI]

Database connection functions

Note

Except for IDENTITY(), which has no meaning in Base, all of these can be carried out using Direct SQL Command.

Table 62: Database connection functions

Function	***Description***
DATABASE()	Returns the name of the database to which this connection belongs. [Works in the GUI]
USER()	Returns the username of this connection. [Direct SQL – does not work with the GUI]
CURRENT_USER	SQL standard function, synonym for USER(). [Works in the GUI]
IDENTITY()	Returns the last value for an autovalue field, which was created in the current connection. This is used in macro coding to transfer a primary key in one table to become a foreign key for another table. [Works in the GUI]

System Functions

Table 63: System functions

Function	***Description***
IFNULL(exp,value)	If exp is NULL, value is returned, otherwise exp is returned. Alternatively as an extension COALESCE() can be used. Exp and value must have the same data type. [Works in the GUI]
CASEWHEN(exp,v1,v2)	If exp is true, v1 is returned, otherwise v2. Alternatively CASE WHEN can be used. CASE WHEN works better with the GUI. [Works in the GUI]
CONVERT(term,type)	Converts term into another data type. [Works in the GUI]
CAST(term AS type)	Synonym for CONVERT(). [Works in the GUI]
COALESCE(expr1,expr2, expr3,...)	If expr1 is not NULL, returns expr1, otherwise expr2 is checked, then expr3 and so on. [Works in the GUI]
NULLIF(v1,v2)	If v1 is equal to v2, NULL is returned, otherwise v1 is returned. [Works in the GUI]
CASE v1 WHEN v2 THEN v3 [ELSE v4] END	If v1 is equal to v2, v3 is returned. Otherwise v4 is returned or NULL, if there is no ELSE condition. [Direct SQL – does not work with the GUI]

Function	*Description*
CASE WHEN expr1 THEN v1[WHEN expr2 THEN v2] [ELSE v4] END	If expr1 is true, v1 is returned [optionally further conditions can be set]. Otherwise v4 is returned or NULL if there is no ELSE condition. [Works in the GUI]
EXTRACT ({YEAR | MONTH | DAY | HOUR | MINUTE | SECOND} FROM <date or time>)	Can replace many of the date and time functions. Returns the year, month, day, etc. from a date or date/time value. [Works in the GUI]
POSITION(<string expression> IN <string expression>)	If the first string is contained in the second one, the offset of the first string is given, otherwise 0 is returned. [Works in the GUI]
SUBSTRING(<string expression> FROM <numeric expression> [FOR <numeric expression>])	Yields part of a text string from the position specified in FROM, optionally up to the length given in FOR. [Works in the GUI]
TRIM([{LEADING | TRAILING | BOTH}] FROM <string expression>)	Non-printing special characters and spaces are removed. [Works in the GUI]

Control characters for use in queries

Fields can be linked together in queries. Two fields in

```
SELECT "First name", "Surname" FROM "Table"
```

become one field by using:

```
SELECT "First name"||' '||"Surname" FROM "Table"
```

Here an additional space is inserted. It could be any character; as long as it is enclosed in '', it will be interpreted as text. Sometimes, however, it is necessary to insert non-printing characters such as new lines, for example in preparing reports. Here is a short list of control characters. More information on these is available at https://en.wikipedia.org/wiki/Control_character.

Table 64: Control characters for use in queries

Code	*Name*	*Notes*
CHAR(9)	Horizontal Tab	
CHAR(10)	Line feed	In mail merge letters and Report Builder, creates a line break (Linux, Unix, Mac).
CHAR(13)	Carriage return	Line break when combined with Carriage return in Windows `CHAR(13)||CHAR(10)`. Can also be used in Linux and Mac, hence the universal variant.

Some UNO commands for use with a button

A button can have various UNO commands directly bound to it. For this purpose you need to choose **Properties: Button > Action > Open document/web page** and then for example the **URL > .uno:RecSearch** to open the search function. Often you will need to choose **Take Focus on Click > No** if the action accesses another control directly in a way that requires it to be in focus, for example **.uno:Paste**, which can insert the contents of the clipboard.

The following list contains only a few commands. All the commands from the navigation toolbar are already usable in the button, but they can also be created using UNO commands. Many commands can be discovered by using the macro recorder, which often uses a dispatcher to access them.

Table 65: Sample UNO commands

UNO command	***Used for ...***
.uno:RecSearch	Opens the search function in a form.
.uno:Paste	Paste from clipboard. Only works for **Take Focus on Click > No**
.uno:Copy	Copies the selected content onto the clipboard. Only works for **Take Focus on Click > No**
.uno:Print	Opens the print dialog for the form.
.uno:PrintDefault	Prints with the default printer without showing a dialog.

Information tables for HSQLDB

Inside a database, information on all table properties and their connections to one another are stored in the *INFORMATION_SCHEMA* area. This information allows Base macros to be created that require very few arguments for their procedures. An application is given in the example database in the Maintenance module—the Table_purge procedure for the control of dialogs.

In a query, individual pieces of information and all the fields that belong can be provided in the following way:

```
SELECT * FROM "INFORMATION_SCHEMA"."SYSTEM_ALIASES"
```

In contrast to a normal table, it is necessary here to use *INFORMATION_SCHEMA* as a prefix to the appropriate name from the following list:

```
SYSTEM_ALIASES
SYSTEM_ALLTYPEINFO
SYSTEM_BESTROWIDENTIFIER
SYSTEM_CACHEINFO
SYSTEM_CATALOGS
SYSTEM_CHECK_COLUMN_USAGE
SYSTEM_CHECK_CONSTRAINTS
SYSTEM_CHECK_ROUTINE_USAGE
SYSTEM_CHECK_TABLE_USAGE
SYSTEM_CLASSPRIVILEGES
SYSTEM_COLUMNPRIVILEGES
SYSTEM_COLUMNS
SYSTEM_CROSSREFERENCE
SYSTEM_INDEXINFO
SYSTEM_PRIMARYKEYS
SYSTEM_PROCEDURECOLUMNS
SYSTEM_PROCEDURES
```

```
SYSTEM_PROPERTIES
SYSTEM_SCHEMAS
SYSTEM_SEQUENCES
SYSTEM_SESSIONINFO
SYSTEM_SESSIONS
SYSTEM_SUPERTABLES
SYSTEM_SUPERTYPES
SYSTEM_TABLEPRIVILEGES
SYSTEM_TABLES
SYSTEM_TABLETYPES
SYSTEM_TABLE_CONSTRAINTS
SYSTEM_TEXTTABLES
SYSTEM_TRIGGERCOLUMNS
SYSTEM_TRIGGERS
SYSTEM_TYPEINFO
SYSTEM_UDTATTRIBUTES
SYSTEM_UDTS
SYSTEM_USAGE_PRIVILEGES
SYSTEM_USERS
SYSTEM_VERSIONCOLUMNS
SYSTEM_VIEWS
SYSTEM_VIEW_COLUMN_USAGE
SYSTEM_VIEW_ROUTINE_USAGE
SYSTEM_VIEW_TABLE_USAGE
```

The following query gives a complete overview of all tables in the database with field types, primary keys and foreign keys:

```
SELECT
"A"."TABLE_NAME",
"A"."COLUMN_NAME",
"A"."TYPE_NAME",
"A"."NULLABLE",
"B"."KEY_SEQ" AS "PRIMARYKEY",
"C"."PKTABLE_NAME" || '.' || "C"."PKCOLUMN_NAME" AS "FOREIGNKEY FOR"
FROM "INFORMATION_SCHEMA"."SYSTEM_COLUMNS" AS "A"
   LEFT JOIN "INFORMATION_SCHEMA"."SYSTEM_PRIMARYKEYS" AS "B"
   ON ( "B"."TABLE_NAME" = "A"."TABLE_NAME" AND "B"."COLUMN_NAME" =
   "A"."COLUMN_NAME" )
   LEFT JOIN "INFORMATION_SCHEMA"."SYSTEM_CROSSREFERENCE" AS "C"
   ON ( "C"."FKTABLE_NAME" = "A"."TABLE_NAME" AND
   "C"."FKCOLUMN_NAME"    = "A"."COLUMN_NAME" )
WHERE "A"."TABLE_SCHEM" = 'PUBLIC'
```

Database repair for *.odb files

Regular data backups should be standard practice when using a PC. Backup copies are the simplest way to return to an even halfway current state for your data. However, in practice this is often lacking.

Forms, queries, and reports can always be copied using the clipboard into a new database, providing that a previous version of the database has been saved. But if, for any reason, the current database can no longer be opened, the main problem becomes access to the data.

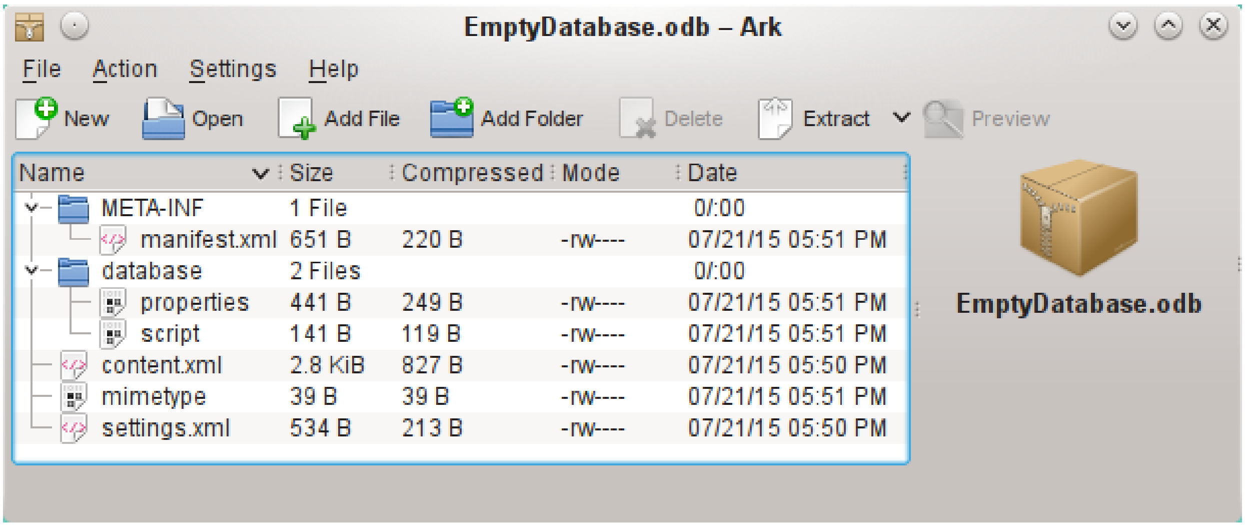
In the case of sudden PC crashes, it can happen that open databases (internal HSQLDB databases) can no longer be opened in LibreOffice. Instead, when you attempt to open the database, you are asked for a filter corresponding to the format.

The problem here is that part of the data in an open database is contained in working memory and is only temporarily copied to intermediate storage. Only when the file is closed is the whole database written back into the file and repacked.

Recovery of the database archive file

To get access again to your data, you may find the following procedure helpful:

1) Create a copy of your database for the steps that follow.
2) Try to open the copy with an archiving program. In the case of *.odb files, we are dealing with a compressed format, a Zip archive. If the file cannot be opened directly, try renaming it from *.odb to *.zip. If that does not open it, your database is past saving.
3) The following folders will always be seen after opening a database file in an archiving program:



Name	Size	Compressed	Mode	Date
META-INF	1 File			0/:00
manifest.xml	651 B	220 B	-rw----	07/21/15 05:51 PM
database	2 Files			0/:00
properties	441 B	249 B	-rw----	07/21/15 05:51 PM
script	141 B	119 B	-rw----	07/21/15 05:51 PM
content.xml	2.8 KiB	827 B	-rw----	07/21/15 05:50 PM
mimetype	39 B	39 B	-rw----	07/21/15 05:51 PM
settings.xml	534 B	213 B	-rw----	07/21/15 05:50 PM

Figure 446: Basic structure of files compressed into ODB file for an "empty" database

4) The database file must be decompressed. The most important information, as far as the data is concerned, is in the subfolder database in the files `data` and `script`.
5) It may be necessary to look at the `script` file and test it for contradictions. This step can, however, be left for the testing stage. The `script` file contains above all the description of the table structure.
6) Create a new empty database file and open this file with the archiving program.
7) Replace the files `data` and `script` in the new database file with the files unpacked in step 4.

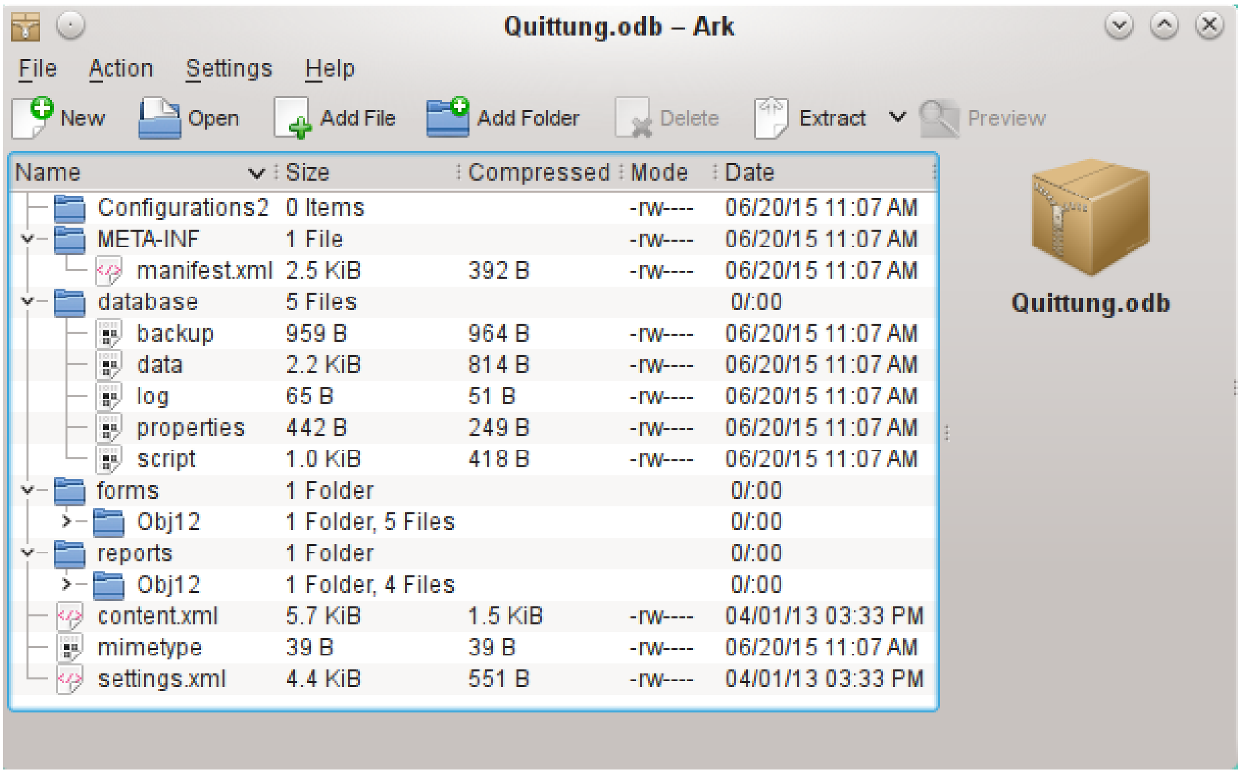
8) Close the archiving program. If it was necessary to rename the file to *.zip before opening it in the archiving program (this depends on your operating system), now rename it again to *.odb.
9) Open the database file in LibreOffice. You should be able to access your tables again.
10) How much of your queries, forms, and reports can be recovered in a similar way must be the subject of further testing.

See also: http://forum.openoffice.org/en/forum/viewtopic.php?f=83&t=17125

Further information on database archive files

In practice, a database archive file contains not only the basic folder for the database, and the folder META-INF which is specified for the OpenDocument format, but also additional folders for storing forms and reports. A description of the basic structure of the OpenDocument format can be found at https://en.wikipedia.org/wiki/OpenDocument_technical_specification.

The following view shows a database containing tables, a form and a report. It is not apparent that the database also contains a query. Queries are not stored in separate folders but in the ‘content.xml’ file. The information necessary to run a query is a simple piece of SQL code.



Name	Size	Compressed	Mode	Date
Configurations2	0 Items		-rw----	06/20/15 11:07 AM
META-INF	1 File		-rw----	06/20/15 11:07 AM
manifest.xml	2.5 KiB	392 B	-rw----	06/20/15 11:07 AM
database	5 Files			0/:00
backup	959 B	964 B	-rw----	06/20/15 11:07 AM
data	2.2 KiB	814 B	-rw----	06/20/15 11:07 AM
log	65 B	51 B	-rw----	06/20/15 11:07 AM
properties	442 B	249 B	-rw----	06/20/15 11:07 AM
script	1.0 KiB	418 B	-rw----	06/20/15 11:07 AM
forms	1 Folder			0/:00
Obj12	1 Folder, 5 Files			0/:00
reports	1 Folder			0/:00
Obj12	1 Folder, 4 Files			0/:00
content.xml	5.7 KiB	1.5 KiB	-rw----	04/01/13 03:33 PM
mimetype	39 B	39 B	-rw----	06/20/15 11:07 AM
settings.xml	4.4 KiB	551 B	-rw----	04/01/13 03:33 PM

Figure 447: Database file containing stored information for a form and a report in addition tables and queries

Here is an overview of one of the database archive files.

mimetype

```
application/vnd.oasis.opendocument.base
```

eine

This little text file contains only the notice that this archive file is a database file in OpenDocument format.

content.xml for a database without content

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-content
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0"
xmlns:math="http://www.w3.org/1998/Math/MathML"
xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:ooow="http://openoffice.org/2004/writer"
xmlns:oooc="http://openoffice.org/2004/calc"
xmlns:dom="http://www.w3.org/2001/xml-events"
xmlns:db="urn:oasis:names:tc:opendocument:xmlns:database:1.0"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rpt="http://openoffice.org/2005/report"
xmlns:of="urn:oasis:names:tc:opendocument:xmlns:of:1.2"
xmlns:xhtml="http://www.w3.org/1999/xhtml"
xmlns:grddl="http://www.w3.org/2003/g/data-view#"
xmlns:tableooo="http://openoffice.org/2009/table"
xmlns:drawooo="http://openoffice.org/2010/draw"
xmlns:calcext="urn:org:documentfoundation:names:experimental:calc:xmlns:calce
xt:1.0" xmlns:field="urn:openoffice:names:experimental:ooo-ms-
interop:xmlns:field:1.0"
xmlns:formx="urn:openoffice:names:experimental:ooxml-odf-
interop:xmlns:form:1.0" xmlns:css3t="http://www.w3.org/TR/css3-text/"
office:version="1.2">
   <office:scripts/>
   <office:font-face-decls/>
   <office:automatic-styles/>
   <office:body>
      <office:database>
         <db:data-source>
            <db:connection-data>
               <db:connection-resource xlink:href="sdbc:embedded:hsqldb"/>
               <db:login db:is-password-required="false"/>
            </db:connection-data>
            <db:driver-settings
                  db:system-driver-settings=""
                  db:base-dn=""
                  db:parameter-name-substitution="false"/>
            <db:application-connection-settings
                  db:is-table-name-length-limited="false"
                  db:append-table-alias-name="false"
                  db:max-row-count="100">
               <db:table-filter>
                  <db:table-include-filter>
                     <db:table-filter-pattern>%</db:table-filter-pattern>
                  </db:table-include-filter>
               </db:table-filter>
            </db:application-connection-settings>
         </db:data-source>
      </office:database>
   </office:body>
</office:document-content>
```

It begins with the XML version and the character set used. Everything that follows is actually a single unwrapped line. The view prepared above should make things clearer. Elements that belong together are bracketed by tags.

The initial definitions beginning with `xmlns:` (XML namespace) give the namespaces that can be accessed from inside the file. Then somewhat more concrete terms are considered. Here it becomes clear that we are dealing with an internal HSQLDB database, and that a password is not required for access.

content.xml for a database with contents

The following content is only an excerpt from the content.xml file, to clarify its structure.

```
<office:scripts/>
<office:font-face-decls>
   <style:font-face style:name="F" svg:font-family=""/>
</office:font-face-decls>
<office:automatic-styles>
   <style:style
      style:name="co1"
      style:family="table-column"
      style:data-style-name="N0"/>
   <style:style
       style:name="co2"
      style:family="table-column"
      style:data-style-name="N107"/>
   <style:style style:name="ce1" style:family="table-cell">
      <style:paragraph-properties fo:text-align="start"/>
   </style:style>
    <number:number-style style:name="N0" number:language="de"
number:country="DE">
      <number:number number:min-integer-digits="1"/>
   </number:number-style>
   <number:currency-style
      style:name="N107P0"
      style:volatile="true"
      number:language="de"
      number:country="DE">
      <number:number
         number:decimal-places="2"
         number:min-integer-digits="1"
         number:grouping="true"/>
      <number:text> </number:text>
      <number:currency-symbol
         number:language="de"
         number:country="DE">€
       </number:currency-symbol>
   </number:currency-style>
```

Here a field is defined as a currency field. The number of decimal places is given, the separation between the numbers and the currency symbol, and the currency symbol itself. In the example below, currency and locale refers to Germany.

```
   <number:currency-style
      style:name="N107"
     number:language="de"
     number:country="DE">
     <style:text-properties fo:color="#ff0000"/>
     <number:text>-</number:text>
     <number:number
       number:decimal-places="2"
       number:min-integer-digits="1"
       number:grouping="true"/>
     <number:text> </number:text>
     <number:currency-symbol
       number:language="de"
       number:country="DE">€
      </number:currency-symbol>
     <style:map style:condition="value()&gt;=0" style:apply-style-
name="N107P0"/>
   </number:currency-style>
```

The second extract states that up to a particular value, currency should appear in red (“ff0000”).

```
</office:automatic-styles>
<office:body>
  <office:database>
    <db:data-source>
```

This entry from the above content.xml file, with all its subentries, corresponds to an empty database archive file.

```
      </db:data-source>
     <db:forms>
       <db:component
         db:name="Receipts"
         xlink:href="forms/Obj12"
         db:as-template="false"/>
     </db:forms>
```

The database archive file contains a subsection in which details of a form are stored. The form is designated in the user interface as *Receipts*.

```
      <db:reports>
       <db:component
         db: name="Receipts"
         xlink:href="reports/Obj12"
         db:as-template="false"/>
     </db:reports>
```

The database archive file also contains a subsection in which details of a report are stored. The report is also designated in the user interface as *Receipts*.

```
      <db:queries>
       <db:query
         db:name="Sales_calc"
         db:command="SELECT &quot;a&quot;.*, ( SELECT &quot;Price&quot; *
           &quot;a&quot;.&quot;Total&quot; FROM &quot;Stock&quot; WHERE
           &quot;ID&quot; = &quot;a&quot;.&quot;Stock_ID&quot; ) AS
```

```
               &quot;Total*Price&quot; FROM &quot;Sales&quot; AS
&quot;a&quot;"/>
     </db:queries>
```

All the queries are stored directly in content.xml. &quot; stands for double quotes. The query above in this example is actually quite complicated with many correlated subqueries. It is reproduced here in an abbreviated form.

```
      <db:table-representations>
       <db:table-representation db:name="Receipts"/>
       <db:table-representation db:name="Sales"/>
       <db:table-representation db:name="Stock">
          <db:columns>
            <db:column
               db:name="ID"
               db:style-name="co1"
               db:default-cell-style-name="ce1"/>
            <db:column
               db:name="MWSt"
               db:style-name="co1"
               db:gefault-cell-style-name="ce1"/>
            <db:column
               db:name="Price"
               db:style-name="co2"
               db:default-cell-style-name="ce1"/>
            <db:column
               db:name="Stock"
               db:style-name="co1"
               db:default-cell-style-name="ce1"/>
          </db:columns>
       </db:table-representation>
     </db:table-representations>
```

This shows how various tables are to be displayed. Here the display properties of particular columns are stored: in this example, settings for the Stock table with its fields – ID, MWSt and so on – are stored. Apparently something has been directly entered here, changing the columns of the table a bit.

```
   </office:database>
</office:body>
```

Basically, content.xml stores directly the contents of queries and information about the visual appearance of tables. In addition there is a definition of the database connection. Finally comes information about forms and reports.

settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-settings
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
xmlns:svg="http://www.w3.org/2000/svg"
xmlns:config="urn:oasis:names:tc:opendocument:xmlns:config:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:db="urn:oasis:names:tc:opendocument:xmlns:database:1.0"
office:version="1.2"/>
```

For a database without further content, only basic definitions are stored here. With content, various settings are also stored. After the start of the above definition, the following settings from the example database are stored.

```
<office:settings>
  <config:config-item-set config:name="ooo:view-settings">
    <config:config-item-set config:name="Queries">
      <config:config-item-set config:name="Calculate_sales">
        <config:config-item-set config:name="Tables">
          <config:config-item-set config:name="Table1">
            <config:config-item config:name="WindowName"
              config:type="string">Verkauf</config:config-item>
            <config:config-item config:name="WindowLeft"
              config:type="int">153</config:config-item>
            <config:config-item config:name="ShowAll"
              config:type="boolean">true</config:config-item>
            <config:config-item config:name="WindowTop"
              config:type="int">17</config:config-item>
            <config:config-item config:name="WindowWidth"
              config:type="int">120</config:config-item>
            <config:config-item config:name="WindowHeight"
              config:type="int">120</config:config-item>
            <config:config-item config:name="ComposedName"
              config:type="string">Verkauf</config:config-item>
            <config:config-item config:name="TableName"
              config:type="string">Verkauf</config:config-item>
          </config:config-item-set>
        </config:config-item-set>
        <config:config-item config:name="SplitterPosition"
          config:type="int">105</config:config-item>
        <config:config-item config:name="VisibleRows"
          config:type="int">1024</config:config-item>
      </config:config-item-set>
    </config:config-item-set>
  </config:config-item-set>
  <config:config-item-set config:name="ooo:configuration-settings">
    <config:config-item-set config:name="layout-settings">
      <config:config-item-set config:name="Tables">
        <config:config-item-set config:name="Table1">
          <config:config-item config:name="WindowName"
            config:type="string">Verkauf</config:config-item>
          <config:config-item config:name="WindowLeft"
            config:type="int">186</config:config-item>
          <config:config-item config:name="ShowAll"
            config:type="boolean">false</config:config-item>
          <config:config-item config:name="WindowTop"
            config:type="int">17</config:config-item>
          <config:config-item config:name="WindowWidth"
            config:type="int">120</config:config-item>
          <config:config-item config:name="WindowHeight"
            config:type="int">120</config:config-item>
          <config:config-item config:name="ComposedName"
            config:type="string">Verkauf</config:config-item>
          <config:config-item config:name="TableName"
            config:type="string">Sales</config:config-item>
        </config:config-item-set>
```

```
          <config:config-item-set config:name="Table2">
            ... (identical config:type-Points as "Table1"
              <config:config-item config:name="TableName"
              config:type="string">Ware</config:config-item>
          </config:config-item-set>
          <config:config-item-set config:name="Table3">
            ... (identical config:type-Points as "Table1"
              <config:config-item config:name="TableName"
              config:type="string">Receipts</config:config-item>
          </config:config-item-set>
        </config:config-item-set>
     </config:config-item-set>
  </config:config-item-set>
</office:settings>
```

The whole overview relates to different views of windows for the query Calculate_sales and the tables Sales, Stock, and Receipts. The last two are shown here in an abbreviated form. If these settings were absent in a defective *.odb file, it would not matter. They would be recreated when the corresponding window was next opened.

META-INF/manifest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest:manifest
   xmlns:manifest="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0">
  <manifest:file-entry
      manifest:full-path="/"
      manifest:media-type="application/vnd.oasis.opendocument.base"/>
  <manifest:file-entry
      manifest:full-path="database/script"
      manifest:media-type=""/>
  <manifest:file-entry
      manifest:full-path="database/properties"
      manifest:media-type=""/>
  <manifest:file-entry
      manifest:full-path="settings.xml"
      manifest:media-type="text/xml"/>
  <manifest:file-entry
      manifest:full-path="content.xml"
      manifest:media-type="text/xml"/>
</manifest:manifest>
```

This file in the META-INF folder gives the contents folder for the whole database archive. As this file deals with an empty database, there are only five file entries. A database archive that contains forms and reports will have a much more complicated META-INF file.

database/properties

```
#HSQL Database Engine 1.8.0.10
#Sun Jul 14 18:02:08 CEST 2013
hsqldb.script_format=0
runtime.gc_interval=0
sql.enforce_strict_size=true
hsqldb.cache_size_scale=8
readonly=false
hsqldb.nio_data_file=false
hsqldb.cache_scale=13
version=1.8.0
hsqldb.default_table_type=cached
hsqldb.cache_file_scale=1
hsqldb.lock_file=true
hsqldb.log_size=10
modified=no
hsqldb.cache_version=1.7.0
hsqldb.original_version=1.8.0
hsqldb.compatible_version=1.8.0
```

The properties file contains the basic settings for the internal HSQLDB database.

database/script

```
SET DATABASE COLLATION "German"
CREATE SCHEMA PUBLIC AUTHORIZATION DBA
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 60
```

The script file contains default settings for connection to the database, the language setting, etc. The user SA, to be described later, appears here.

In a database with contents, this file contains the basic table definitions.

```
SET DATABASE COLLATION "German"
CREATE SCHEMA PUBLIC AUTHORIZATION DBA
```

The tables are defined before the database user is defined. First the tables are created in the cache with their fields.

```
CREATE CACHED TABLE "Stock"
   ("ID" INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL

   PRIMARY KEY,"Stock" VARCHAR(50),"Price" DECIMAL(8,2),"MWSt" TINYINT)
CREATE CACHED TABLE "Sales"
   ("ID" INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL

   PRIMARY KEY,"Total" TINYINT,"Stock_ID" INTEGER,"Receipt_ID" INTEGER,
   CONSTRAINT SYS_FK_59 FOREIGN KEY("Stock_ID") REFERENCES "Stock"("ID"))
CREATE CACHED TABLE "Receipts"
   ("ID" INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL

   PRIMARY KEY,"Date" DATE)
```

Then changes in the table are made to ensure that the relationships (REFERENCES) are consistent.

```
ALTER TABLE "Sales" ADD CONSTRAINT SYS_FK_76 FOREIGN KEY("Receipt_ID")
   REFERENCES "Receipts"("ID")
SET TABLE "Stock" INDEX'608 20'
SET TABLE "Sales" INDEX'1872 1656 1872 12'
SET TABLE "Receipts" INDEX'2232 1'
```

After setting the position of the index in the data file (it appears here only in the script file but is never actually entered directly in SQL), the automatically incrementing fields in the tables (AutoValues) are set up so that they will provide the next value on entry of a new record. Suppose the last entered value in the ID field of the Stock table is 19. Auto-incrementing then starts at 20.

```
ALTER TABLE "Stock" ALTER COLUMN "ID" RESTART WITH 20
ALTER TABLE "Sales" ALTER COLUMN "ID" RESTART WITH 12
ALTER TABLE "Receipts" ALTER COLUMN "ID" RESTART WITH 1
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 60
```

Managing the internal Firebird database

The internal Firebird database is at the moment only available as an experimental function. To create such a database, or to edit one that has been created, you must select **Tools > Options > LibreOffice > Advanced > Optional (Unstable) Options > Enable experimental features**.

The following link allows significant bugs in the internal Firebird database to be reported and addressed jointly with the LibreOffice team: Reporting bugs for Firebird in Base.

Users will notice the following differences from HSQLDB:

1) If a field is given the type Integer and then declared as the primary key, it appears to be possible to give it an auto-incrementing value. However on saving, this setting disappears without notice.
2) When new records are entered, they are not automatically saved in the database. The Save button has to be used for each entry. In the built-in HSQLDB, the explicit saving of records is not necessary.
3) Aliases are completely ignored in queries. An alias can be created but it will not appear in the table heading of the query.
4) It is not possible to create conditions, although external Firebird databases support them.
5) The decimal and numeric data types are faulty at present. These are the only types that ensure precise values, especially when there are decimal places. They are therefore the preferred fields for currency values. At present, only values with at most one decimal place can be entered.

Making AutoValues available

The following code, entered using **Tools > SQL**, can help with the problem of auto-values not being provided.

```
CREATE TABLE "Table1" ( "ID" INTEGER NOT NULL PRIMARY KEY, "Name" 
VARCHAR(20) NOT NULL );

CREATE GENERATOR GEN_T1_ID;

SET GENERATOR GEN_T1_ID TO 0;
```

After this, the SQL entry window should be closed and **View > Refresh Tables** selected. Only when the table appears, and in some cases only after an (unsuccessful) attempt to create an entry, can the following Trigger be created.

```
CREATE TRIGGER T1_BI FOR "Table1" ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
```

```
IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(GEN_T1_ID, 1);
END;
```

Even after this, many entries in Name can be made in the table without creating an entry in ID. The ID field just shows 0. Only when **Update** is pressed are the actual assigned values displayed. The trigger provides values that begin with 1.

Base Guide 7.3

Appendix B Comparison of HSQLDB and Firebird

Data Types and Functions

Data types and functions in HSQLDB and Firebird

The tables in this Appendix are taken from the manuals for HSQLDB and Firebird.

- http://hsqldb.org/doc/guide/
- https://www.firebirdsql.org/en/documentation/

The information for internal HSQLDB is the same as in Appendix A of this book.

The additional internal database Firebird is classified as experimental.

The tables first provide a comparison of the functions, especially the functions that are popular in forums, such as:

- Add a certain number of days to a date (DATEADD)
- Values from multiple data lines grouped together in one data line (LIST)

are currently only available in the external Firebird database, but not in the internal version.

Built-in functions and stored procedures

The following functions are available in the built-in databases. One or two functions can only be used when **Run SQL command directly** is chosen. While in that mode, queries cannot be edited.

Functions that work with the graphical user interface are marked [works in the GUI]. Functions that work only in direct SQL commands are marked [does not work in the GUI].

Numeric Functions

Caution

As we are dealing here with floating point numbers, be sure to take care with the settings of the fields in queries. Mostly the display of decimal places is restricted, which can result in unexpected behavior in some cases. For example, column 1 might show 0.00 but actually contain 0.001, and column 2, 1000. If column 3 is set to show Column 1 * Column 2, it would actually show 1.

Table 66: Numeric functions

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
`ABS(d)`	Returns the absolute value of a number. [works in the GUI]	`ABS(d)`	
`ACOS(d)`	Returns the arccosine. [works in the GUI]	`ACOS(d)`	
`ASIN(d)`	Returns the arcsine. [works in the GUI]	`ASIN(d)`	
`ATAN(d)`	Returns the arctangent. [works in the GUI]	`ATAN(d)`	
`ATAN2(a,b)`	Returns the arctangent via coordinates. 'a' is the value of the x-axis, 'b' is the value of the y-axis. [works in the GUI]	`ATAN2(x,y)`	
`BITAND(a,b)`	Both the binary notation of 'a' and the binary notation of 'b.' Must have a '1' in the same position to yield '1' in the result. BITAND (3,5) returns 1; 0011 AND 0101 = 0001 [works in the GUI]	`BIN_AND(x,y [,z...])`	
`BITOR(a,b)`	Either the binary notation of 'a' or the binary notation of 'b.' Must have a '1' in the same position to yield '1' in the result. BITOR (3,5) returns 7; 0011 OR 0101 = 0111 [works in the GUI]	`BIN_OR(x,y [,z...])`	
		`BIN_SHL(n,exp)`	n · 2 exp [works in the GUI]
		`BIN_SHR(n,exp)`	n / 2 exp The result is shown as a rounded integer. [works in the GUI]

HSQLDB		*Firebird*	
Syntax	***Description***	***Syntax***	***Description***
		`BIN_XOR(x,y [,z...])`	Either the binary notation of 'a' or the binary notation of 'b.' Must have a '1' in the same position to yield '1' in the result. `BIN_XOR(3,5)` returns 6; 0011 X OR 0101 = 011 0 [works in the GUI]

`CEILING(d)`	Specifies the smallest integer that is not less than d. [works in the GUI]	`CEIL(d)`
		`CEILING(d)`	

`COS(d)`	Returns the cosine. [works in the GUI]	`COS(radians)`	The radians can also be represented using the angle (here for the unit circle): radians = (2 * PI () * angles / 360)
		`COSH(d)`	Returns the hyperbolic cosine.
`COT(d)`	Returns the cotangent. [works in the GUI]	`COT(d)`	
`DEGREES(d)`	Converts radians to degrees. [works in the GUI]		
`EXP(d)`	Returns e ^{d} (e: (2.718 ...)). [works in the GUI]	`EXP(d)`	
`FLOOR(d)`	Returns the largest integer that is not greater than d. [works in the GUI]	`FLOOR(d)`	
`LOG(d)`	Returns the natural logarithm of base 'e'. [works in the GUI]	`LN(d)`	
`LOG10(d)`	Returns the base 10 logarithm. [works in the GUI]	`LOG10(d)`	
		`LOG(base,d)`	Returns the log at any base.
`MOD(a,b)`	Returns the remainder as an integer that results from dividing 2 integers. MOD (11,3) returns 2 because 3 * 3 + 2 = 11 [works in the GUI]	`MOD(a,b)`	
`PI()`	Returns π (3.1415...) [works in the GUI]	`PI()`	

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
`POWER(a,b)`	a^b , POWER (2,3) = 8, because 2^3 = 8 [works in the GUI]	`POWER(x,y)`	
`RADIANS(d)`	Converts degrees to radians. [works in the GUI]		
`EDGE()`	Returns a random number x greater than or equal to 0.0 and less than 1.0. [works in the GUI]	`EDGE( )`	
`ROUND(a,b)`	Rounds a to b digits after the decimal point. [works in the GUI]	`ROUND(d [, places])`	Rounds after the specified number of digits from the decimal point. ROUND (123.45, 1) returns 123.50 ROUND (123.45, –2) returns 100.00 [works in the GUI]
`ROUNDMAGIC (d)`	Solves rounding problems caused by floating point numbers. 3.11–3.1–0.01 may not be exactly 0, but is displayed as 0 in the GUI. ROUNDMAGIC turns it into an actual 0 value. [works in the GUI]		
`SIGN(d)`	Returns –1 if 'd' is less than 0, 0 if 'd' is equal to 0, and 1 if 'd' is greater than 0. [works in the GUI]	`SIGN(d)`	
`SIN(A)`	Returns the sine of an angle in radians. [works in the GUI]	`SIN(radians)`	
		`SINH(d)`	Returns the hyperbolic sine.
`SQRT(d)`	Returns the square root. [works in the GUI]	`SQRT(d)`	
`TAN(A)`	Returns the tangent of an angle in radians. [works in the GUI]	`TAN(radians)`	
		`TANH(d)`	Returns the hyperbolic tangent.

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
TRUNCATE (a,b)	Cuts 'a' to 'b' characters after the decimal point. TRUNCATE(2.37456.2) = 2.37 [works in the GUI]	TRUNC(d[,jobs])	Sets to 0 after the specified number of digits from the decimal point . TRUNC (123.45, 1) returns 123. 4 0 ROUND (123.45, –2) returns 100.00 [works in the GUI]

Text Functions

Table 67: Text functions

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
ASCII(s)	Returns the ASCII code of the first letter of the string. [works in the GUI]	ASCII_VAL ('s')	Returns the numerical value that matches the character entered. [works in the GUI]
BIT_LENGTH (str)	Returns the length of the text str in bits. [works in the GUI]	BIT_LENGTH ('s')	
CHAR(c)	Returns the letter that belongs to the ASCII code. It is not just about letters, but also about control characters. CHAR (13) creates a line break in a query, which is visible in multi-line fields of a form or in reports. [works in the GUI]	ASCII_CHAR (n)	Returns the letter that belongs to the ASCII code. It is not just about letters, but also about control characters. [works in the GUI]
CHAR_LENGTH (str)	Returns the length of the text in letters. [works in the GUI]	CHAR_LENGTH ('s') CHARACTER_LENGTH('s')	
		CHAR_TO_UUID	Converts a 36-character Universally Unique Identifier (UUID) to a 16-byte UUID (outputs unreadable characters).
CONCAT (STR1,STR2)	Joins str1 + str2. [works in the GUI]		

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
DIFFERENCE (s1,s2)	Shows the sound difference between s1 and s2. Only an integer is output. 0 means the same sound. So 'for' and 'four' with 0 appear equal, shortening and seasoning is set to 1, mouth and moon back to 0. [works in the GUI]		
HEXTORAW (s1)	Translates hexadecimal code into other characters. [works in the GUI]		
INSERT(s, start, len,s2)	Returns a text string, with part of the text replaced. Beginning with ‘start,’ a length ‘len’ is cut out of the text ‘s’ and replaced by the text ‘s2.’ INSERT(‘Bundesbahn’, 3, 4, ‘mmel’) converts Bundesbahn into Bummelbahn, where the length of the inserted text can be greater than that of the deleted text without causing any problems. So INSERT(‘Bundesbahn’, 3, 5, ‘s und B’) yields 'Bus und Bahn'. [works in the GUI]	OVERLAY ('s' PLACING 's2' FROM pos [FOR length])	If the start position is set so that it is greater than or equal to the actual text 's', then 's2' is directly appended to 's'. OVERLAY 'Bundesbahn' PLACING 'mmel' FROM 3 FOR 4) turns 'Bundesbahn' into 'Bummelbahn', where the length of the inserted text can be longer than that of the cut text. So OVERLAY ('Bundesbahn' PLACING ' s and B ' FROM 3 FOR 5) results in 'Bus und Bahn'. [does not work in the GUI]
LCASE(s)	Converts the string to lowercase. [works in the GUI]		
LEFT(s, count)	Returns the number of characters specified with count from the beginning of the string s. [works in the GUI]	LEFT('s', length)	
LENGTH(s)	Returns the length of a string. [works in the GUI]		

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
LOCATE (search,s [,start])	Returns the first match for the term from search in the string s. The match is given numerically: (1 = left, 0 = not found). Specifying a starting point within the string is optional. [works in the GUI]	POSITION ('s2' IN 's') POSITION ('s2', 's' [, S tartpos ition])	
POSITION (<string expression> IN <string expression>)	If the first string is contained in the second, the position of the first string will be returned, otherwise 0 will be displayed. This could be used instead of a search option with LIKE. [works in the GUI]		
LTRIM(s)	Removes leading spaces and non-printable characters from text. [works in the GUI]		
OCTET_LENGTH (str)	Returns the length of a text string in bytes. In principle, this corresponds to twice the length in characters. [works in the GUI]	OCTET_LENGTH (str)	Returns the actual number of characters, taking spaces into account. The number depends on the character set. UTF-8 needs two bytes for special characters.
RAWTOHEX(s1)	Converts to hexadecimal notation; reverse of HEXTORAW(). [works in the GUI]		
REPEAT(s, count)	Repeats the text string ‘s’ count times. [works in the GUI]		
REPLACE(s, replace,s2)	Replaces all existing occurrences of ‘replace’ in the string ‘s’ with the text ‘s2’. [works in the GUI]	REPLACE('s', 's2', replacement)	REPLACE ('Schule', 'ul', 'eib') converts 'Schule' to 'Schieibe’. If 's2' does not appear in 's', nothing will be replaced. If N2 appears in 's2' or replacement, the result is NULL.

HSQLDB		*Firebird*	
Syntax	***Description***	***Syntax***	***Description***
		LPAD('s', total length [,characters])	LPAD ('Hello', 8 , '+') = +++ Hello Places any characters in front of a string until the total length is reached. If the total length is less than the length of the string, the string is cut off on the right. If the third parameter remains empty, spaces are placed in front of it.
		RPAD('s', total length [,characters])	RPAD ('Hello', 8 , '+') = Hello +++ Places any characters behind a string until the total length is reached. If the total length is less than the length of the string, the string is cut off on the right. If the third parameter remains empty, spaces are placed behind it.
		REVERSE('s')	Inverts the string completely. This can be useful, for example, if you want to sort by character endings (e.g. domain endings).
RIGHT(s, count)	Reverse to LEFT; returns the number of characters specified with count from the end of the string. [works in the GUI]	RIGHT('s', length)	
RTRIM(s)	Removes all spaces and non-printable characters at the end of the string. [works in the GUI]		
SOUNDEX(s)	Returns a code of 4 characters, which should correspond to the sound of s - fits the function DIFFERENCE (). [works in the GUI]		
SPACE(count)	Returns the number of spaces specified in count. [works in the GUI]		
SUBSTR(s, start [,len])	Abbreviation for SUBSTRING. [works in the GUI]		

HSQLDB		*Firebird*	
Syntax	***Description***	***Syntax***	***Description***
SUBSTRING (s,start [,len])	Returns the string s from the start position. (1 = left). If the length is omitted, the entire string is returned. [works in the GUI]		
SUBSTRING (<string expression> FROM <numeric expression> [FOR <numeric expression>])	Returns the part of a string from the start position specified in FROM, optionally in the length specified in FOR. If, for example, "Roberta" appears in the "Name" field, SUBSTRING ("Name" FROM 3 FOR 3) results in the substring 'bert'. [works in the GUI]	SUBSTRING ('s' FROM start position [FOR length])	
TRIM ([{LEADING | TRAILING | BOTH}] FROM <string expression>)	Special characters and spaces that cannot be printed are removed. [works in the GUI]	TRIM ([Where 's2' FROM] 's')	Where: BOTH | LEADING | TRAILING standard here is BOTH for both sides of 's'. s2: The character to be removed. By default, these are spaces (' '), but can also be other characters. TRIM (TRAILING '!' FROM 'Hallo!') returns 'Hallo'
UCASE(s)	Converts the string to uppercase. [works in the GUI]		
LOWER(s)	As LCASE(s) [works in the GUI]	LOWER('s')	
UPPER(s)	As UCASE(s). [works in the GUI]	UPPER('s')	
		UUID_TO_CHAR ('s')	Converts a 16-byte UUID to a 36-character ASCII format.

Date/Time Functions

Table 68: Date/time functions

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
		DATEADD (n DAY TO date) DATEADD (DAY, n, date)	n is an integer and can also be negative for subtraction. YEAR | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND | MILLISECOND are to be used as terms for the time interval. Use either a date / date field, a time / time field, or a timestamp as the date term.
DATEDIFF (string, datetime1, datetime2)	Date difference between two dates or times. The entry in string decides in which unit the difference is shown: 'ms' = 'millisecond', 'ss' = 'second', 'mi' = 'minute', 'hh' = 'hour', 'dd' = 'day', 'mm' = 'month', 'yy' = 'year'. Both the long version and the short version of the string can be used. [works in the GUI]	DATEDIFF (DAY FROM date TO date) DATEDIFF (DAY, date, date)	See DATEADD.
EXTRACT ({YEAR| MONTH |DAY| HOUR| MINUTE| SECOND} FROM <date or time value>)	Can replace many of the date and time functions. Returns the year, month, day, etc. from a date or time of day value. EXTRACT (DAY FROM "date") shows the day of the month. [works in the GUI]	EXTRACT ({YEAR| MONTH | WEEK | DAY | WEEKDAY | YEARDAY | HOUR | MINUTE | SECOND | MILLISECOND } FROM <date or time value>)	WEEKDAY Sunday = 0 YEARDAY January 1st = 0 WEEK 1st week: min. 4 days a year
DAY(date)	Returns the day of the month (1-31). [works in the GUI]		
DAYNAME (date)	Returns the English name of the day. [works in the GUI]		
DAYOFMONTH (date)	Returns the day of the month (1-31), synonym for DAY(). [works in the GUI]		
DAYOFWEEK (date)	Returns the day of the week as a number (1 means Sunday.) [works in the GUI]		

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
DAYOFYEAR (date)	Returns the day of the year (1-366). [works in the GUI]		
HOUR(time)	Returns the hour (0-23). [works in the GUI]		
MINUTE(time)	Returns the minute (0-59). [works in the GUI]		
MONTH(date)	Returns the month (1-12). [works in the GUI]		
MONTHNAME (date)	Returns the English name of the month. [works in the GUI]		
QUARTER (date)	Returns the quarter of the year (1-4). [works in the GUI]		
SECOND(time)	Returns the seconds of a time (0-59). [works in the GUI]		
WEEK(date)	Returns the week of the year (1-53). [works in the GUI]		
YEAR(date)	Returns the year from a date input. [works in the GUI]		

Database connection functions

Table 69: Database connection functions

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
`DATABASE()`	Returns the path and name of the database belonging to this connection. [works in the GUI]		
		`CURRENT_TRANSACTION`	SELECT CURRENT_T RANSACTION FROM RDB $ DATABASE returns the unique identifier of the transaction as an integer.

HSQLDB		***Firebird***	
Syntax	***Description***	***Syntax***	***Description***
		`CURRENT_CONNECTION`	SELECT CURRENT_CONNECTION FROM RDB $ DATABASE returns an integer value for the current connection.
		`CURRENT_ROLE`	SELECT CURRENT_ROLE FROM RDB $ DATABASE reflects the role of the current user. If no role is defined, the result is NONE.
		`RDB $ SET_CONTEXT ('<namespace>', '<variable name>', value | NULL )`	Namespace: USER_SESSION | USER_TRANSACTION The variable name can have a maximum of 80 characters, and the value can have a maximum of 255 characters.
`CURRENT_USER`	SQL standard function, synonym for USER(). It should be noted that there are no parentheses here. [works in the GUI]	`CURRENT_USER`	
`USER()`	Returns the username of this connection. The username is important if the database is to be converted into an external database. [SQL direct - does not work with the GUI]	`USER`	
`IDENTITY()`	Returns the last value for an autovalue field that was created in the current connection. This is used in macro programming to create a foreign key for another table from a primary key created for one table. [works in the GUI]	`GEN_ID (generator name, <step>)`	Autovalues are created with a generator. The step size should be given here as 1. In principle, any integer value is possible. `new.rec_id = gen_id (gen_recnum, 1);`

System Functions

Table 70: System functions

<table>
<tr><th colspan="2">HSQLDB</th><th colspan="2">Firebird</th></tr>
<tr><th>Syntax</th><th>Description</th><th>Syntax</th><th>Description</th></tr>
<tr><td><code>IFNULL (exp, value)</code></td><td>If exp is NULL, value is returned, otherwise exp. Instead, COALESCE () can also be used as an extension. Exp and value must have the same data type.
IFNULL is an important function when fields are linked with each other by invoice or CONCAT. The content of the result would be NULL, even if only one value is NULL.
"Last name" || ',' || "first name" would result in an empty field for people who, for example, lack the entry for "first name", ie NULL.
"Last Name" || IFNULL (',' || "First Name", '') would just print "Last Name" instead.
[works in the GUI]</td><td></td><td></td></tr>
<tr><td><code>CASE WHEN (exp, v1, v2)</code></td><td>If exp is true v1 is returned, otherwise v2. CASE WHEN can also be used instead.
CASEWHEN ("a" > 10, 'goal reached', 'still practice') returns 'goal reached' if the content of the field "a" is greater than 10.
[works in the GUI]</td><td><code>IIF (&lt;condition>, v1, v2)</code></td><td></td></tr>
<tr><td><code>CONVERT (term, type)</code></td><td>Converts term to another data type.
CONVERT ("a", DECIMAL (5,2)) makes the field "a" a field with 5 digits, including 2 decimal places . If the number is too large, an error is output.
[works in the GUI]</td><td></td><td></td></tr>
<tr><td><code>CAST (term AS type)</code></td><td>Synonym to CONVERT ()
[works in the GUI]</td><td><code>CAST (term AS type)</code></td><td><table><tr><th>From</th><th>To</th></tr><tr><td>Numeric types</td><td>Numeric types [VAR]
CHAR
BLOB</td></tr></table></td></tr>
</table>

HSQLDB		*Firebird*		
Syntax	*Description*	*Syntax*	*Description*	
			[VAR] CHAR BLOB	[VAR] CHAR BLOB Numeric types DATE TIME TIMESTAMP
			DATE TIME	[VAR] CHAR BLOB TIMESTAMP
			TIMESTAMP	[VAR] CHAR BLOB DATE TIME
`COALESCE (expr1, expr2, expr3, ...)`	If expr1 is not NULL, expr1 is displayed, otherwise expr2 is checked, then expr3, etc. All expressions must have at least a similar data type. This is the alternative representation of integers and floating point numbers, but not also of a date or time value. [works in the GUI]	`COALESCE (expr1, expr2 [, expr3 ...]`		
`NULLIF (v1, v2)`	If v1 is equal to v2, null is returned, otherwise value of v1 is returned. The data must be comparable in type. [works in the GUI]	`NULLIF (v1, v2)`		

HSQLDB		*Firebird*	
Syntax	***Description***	***Syntax***	***Description***
`CASE v1 WHEN v2 THEN v3 [ELSE v4] END`	If v1 is equal to v2, v3 is executed. Otherwise v4 is executed or NULL if no ELSE is formulated. [SQL direct - does not work with the GUI]	`DECODE ( test expression , expression , result [, expression2 , Earnings2 ...] [, default expression ])`	DECODE (UPPER ("gender"), 'M', 'male', 'F', 'female', 'unknown')
`CASE WHEN expr1 THEN v1 [WHEN expr2 THEN v2] [ELSE v4] END`	If expr1 is true, v1 is returned [Optionally, further cases can be specified]. Otherwise v4 is reproduced or NULL if no ELSE is formulated. CASE WHEN DAYOFWEEK ("date") = 1 THEN 'Sunday' WHEN DAYOFWEEK ("date") = 2 THEN 'Monday'… END could output the day of the week via SQL, which is otherwise only available in English in the function. [works in the GUI]		DECODE (EXTRACT (WEEK DAY FROM "date"), 0 , ' Sunday ', 1 , ' Monday ', ' etc. ')
		`GEN_UUID ()`	Returns a unique ID as a 16-byte character set.
		`HASH (s)`	Returns the hash value of an arbitrarily long string. Hash values of the same character strings must be the same.
		`MAXVALUE (expr [, expr ...])`	Returns the maximum value of a list of values. Works with strings, numeric values, date or time values.
		`MINVALUE (expr [, expr ...])`	Returns the minimum value of a list of values. Works with strings, numeric values, date or time values.

Aggregate Functions

Table 71: Aggregate functions (especially with GROUP BY)

HSQLDB		**Firebird**	
Syntax	***Description***	***Syntax***	***Description***
		MAX (expr)	Maximum value of a field in a table.
		MIN (expr)	Minimum value of a field in a table.
		LIST ([ALL | DISTINCT] 's' , [' s2'])	Connects fields of several data records to one field with the corresponding connection term 's2'. [works in the GUI]

Variables

Table 72: Variables (depending on the computer)

HSQLDB		**Firebird**	
Variable	***Description***	***Variable***	***Description***
CURRENT_TIME	Synonym for CURTIME (), SQL standard. [works in the GUI]	CURRENT_TIME	Time in hours, minutes and seconds. CURRENT_TIME (3) also specifies milliseconds.
CURTIME ()	Returns the current time. [works in the GUI]		
CURRENT_TIMESTAMP	Synonym for NOW (), SQL standard. [works in the GUI]	CURRENT_TIMESTAMP [(accuracy)]	Time specification with date and milliseconds. The accuracy can be set with [0 | 1 | 2 | 3]. The standard is 3 decimal places. SELECT CURRENT_TIMESTAMP (2) FROM RDB $ DATABASE returns the time stamp with tenths and hundredths of a second (2 decimal places).
NOW ()	Returns the current date and time together as a timestamp. CURRENT_TIMESTAMP can also be used instead. [works in the GUI]	CAST ('NOW' AS DATE | TIME | TIMESTAMP) or DATE 'NOW'	'NOW', written alone, is understood as a string. With the appropriate conversion, it becomes a date, a time, or a time stamp (each with 1/1000 s). The short form does not work in the GUI.

HSQLDB		***Firebird***	
Variable	***Description***	***Variable***	***Description***
`CURRENT_DATE`	Synonym for CURDATE (), SQL standard. [works in the GUI]	`CURRENT_DATE`	
`CURDATE ()`	Returns the current date. [works in the GUI]		

Operators and statements

Table 73: Operators and statements

HSQLDB		***Firebird***	
Operator or statement	***Description***	***Operator or statement***	***Description***
`'str1' || 'str2' || 'str3' or 'str1' + 'str2' + 'str3'`	Connects str1 + str2 + str3; simpler alternative to CONCAT. [works in the GUI]	`'s1' || ' s 2 ' [ ||' s3 '... ]`	Connects s1, s2 etc. to a new string [works in the GUI]
		`ALL`	
		`ANY / SOME`	
		`IN ( )`	
		`IS [NOT] DISTINCT FROM`	Result is 'yes' or 'no'.
		`NEXT VALUE FOR sequence name`	See GEN_ID (), but does not allow any steps other than 1.
		`SOME`	

Data types for the table editor

Integer data types

Table 74: Integer data types

Type	*Option*	*HSQLDB*	*Firebird*	*Range*	*Storage space*
Tiny integer	TINYINT	TINYINT		2^8 = 256 | – 128 to + 127	1 byte
Small integer	SMALLINT	SMALLINT	SMALLINT	2^{16} = 65536 | – 32768 to + 32767	2 bytes
integer	INTEGER	INTEGER | INT	INTEGER	2^{32} = 4294967296 | – 2147483648 to + 2147483647	4 bytes
BigInt	BIGINT	BIGINT	BIGINT	2^{64} (-2^{63} to $+2^{63}$)	8 bytes

Floating point data types

Table 75: Floating-point data types

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Decimal	DECIMAL	DECIMAL	DECIMAL (n, m)	Unlimited, through GUI to 50 digits, adjustable, fixed decimal places, exact accuracy	2, 4 or 8 Byte
Number	NUMERIC	NUMERIC	NUMERIC (n, m)	Unlimited, through GUI to 50 digits, adjustable, fixed decimal places, exact accuracy	2, 4 or 8 Byte
Float	FLOAT	(DOUBLE is used instead)	FLOAT	$3.4 * 10^{-38}$ to $3.4 * 10^{38}$ adjustable, not exact, 7 decimal places maximum	4 Byte
Real	REAL	REAL			
Double	DOUBLE	DOUBLE [PRECISION] | FLOAT	DOUBLE PRECISION	$1 , 7 * 10^{-308}$ to $1 , 7 * 10^{308}$ adjustable, not exact, 15 decimal places maximum	8 bytes

Text data types

Table 76: Text data types

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Text	VARCHAR	VARCHAR	VARCHAR (n)	Adjustable	Variable Firebird: 1 to 32767 bytes
Text	VARCHAR_IGNORECASE	VARCHAR_IGNORECASE		Adjustable, affect sorting, ignores differences between upper and lower case	variable
Text (fixed)	CHAR	CHAR | CHARACTER		Adjustable, rest of the actual text is filled with spaces	fixed
Memo	LONGVARCHAR	LONGVARCHAR	BLOB (BLOB SUB_TYPE 1)		variable Firebird: <32 GB

Date/Time data types

Table 77: Date/time data types

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Date	DATE	DATE	DATE		4 bytes
Time	TIME	TIME	TIME	Firebird: 0:00 to 23:59,9999	4 bytes
Date/Time	TIME STAMP	TIMESTAMP | DATE TIME	TIME STAMP	Adjustable (HSQLDB: 0, 6 - 6 means with milliseconds)	8 bytes

Other data types

Table 78: Other data types

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Yes No	BOOLEAN	BOOLEAN | BIT			

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Binary field (fixed)	BINARY	BINARY		Like integer	fixed
Binary field	VARBINARY	VARBINARY		Like integer	variable
Image	LONGVARBINARY	LONGVARBINARY	BLOB SUB_TYPE 0 BLOB SUB_TYPE binary	Like integer	variable, intended for larger images Firebird: <32 GB
OTHER	OTHER	OTHER | OBJECT			

LibreOffice Community

LibreOffice Documentation Team

Base Guide

Managing your data

About this book:

This book is for beginners to advanced users of LibreOffice Base. If you have never used Base before, or you want an introduction to all of LibreOffice's components, you might like to read Getting Started with LibreOffice first.

This book introduces you to the most common functions of LibreOffice Base:

Creating a database

Input and output of data

Working with tables, forms, queries, and reports

Using macros

Maintaining a database

And much more

About the authors:

This book was written by volunteers from the LibreOffice community. Profits from sales of the printed edition will be used to benefit the community.

A PDF version of this book can be downloaded free from:
https://documentation.libreoffice.org

About LibreOffice:

LibreOffice is the free, libre, and open source personal productivity suite from The Document Foundation. It runs on Windows, macOS, and GNU/Linux. Support and documentation is free from our large, dedicated community of users, contributors and developers.

You too can get involved with volunteer work in many areas: development, quality assurance, documentation, translation, user support, and more.

You can download LibreOffice Community from:

https://libreoffice.org/download/

For commercial/enterprise use: The LibreOffice Enterprise class of applications may look and behave differently from the Community edition described in this book. For guides to Enterprise editions of LibreOffice, please refer to the professional partners that produced them.

7.3

Fantastic Community - Fun Project - Free Office Suite