Base Guide 7.2

Chapter 10
Database Maintenance

Copyright

Contributors

To this edition

Pulkit Krishna

To previous editions

Robert Großkopf
Dan Lewis
Pulkit Krishna
Hazel Russman
Jost Lange
Jean Hollis Weber

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 December 2021. Based on LibreOffice 7.2 Community.
Other versions of LibreOffice may differ in appearance and functionality.

Contents

General remarks on maintaining databases

Frequently altering the data in a database—especially frequently deleting data—has two effects. 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. Important maintenance is described in this chapter.

Compacting a database

The behavior of HSQLDB is to preserve storage space for deleted records. Databases that are filled with test data, especially images, retain the same size even if all these records are subsequently deleted. This is because of a property of each table's primary keys. The database document file contains the last value used for each primary key. When a new record is made within a table, it is assigned the next value.

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

A database is created, all possible functions tested with examples, and corrections made until everything works. As a result, before a database is even ready to use, it is possible for primary key values to have risen to over 100. Often, primary keys are 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 'database name'
```

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. 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.

The format of the exported file can be changed using:

```
SET SCRIPTFORMAT {TEXT | BINARY | COMPRESSED};
```

To export the file requires using this SQL code one line at a time:

```
SCRIPT ‘database name’;
SET SCRIPTFORMAT {TEXT | BINARY | COMPRESSED}:
SHUTDOWN SCRIPT;
CHECKPOINT;
```

This exports the text file *database name* in the home folder with the database information.

The file can be imported using **Tools > SQL** and creating a new database with the same data. In the case of an internal database, the following lines must be removed 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 of it needs to be divided 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.

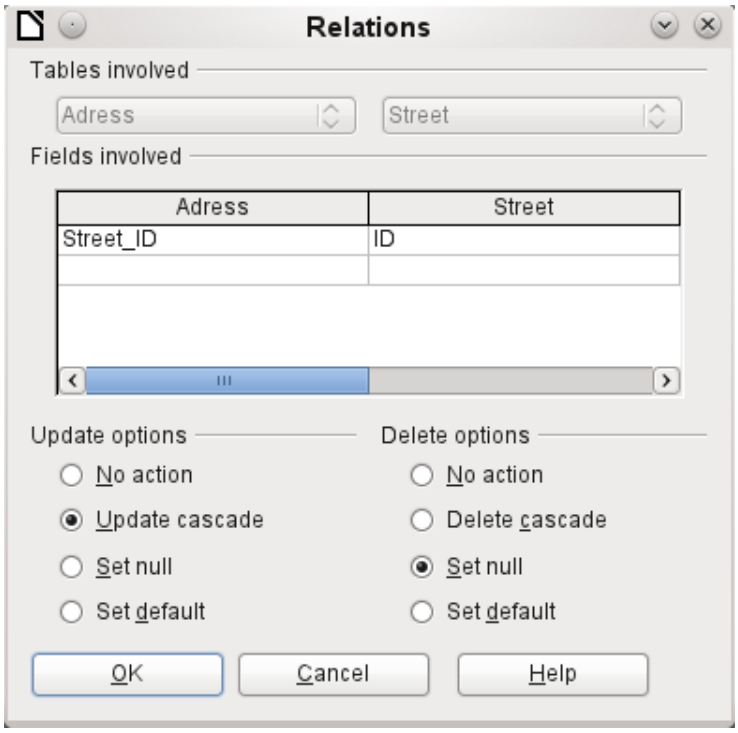
There is a limit to the size of second file: it needs to be smaller than 65KB. If it is larger than that, it too should be divided into smaller text files by cutting and pasting. Just make sure that the top line of each of these new files begins with `INSERT INTO`. One way to do this is to cut from the 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 alteration 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.



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.

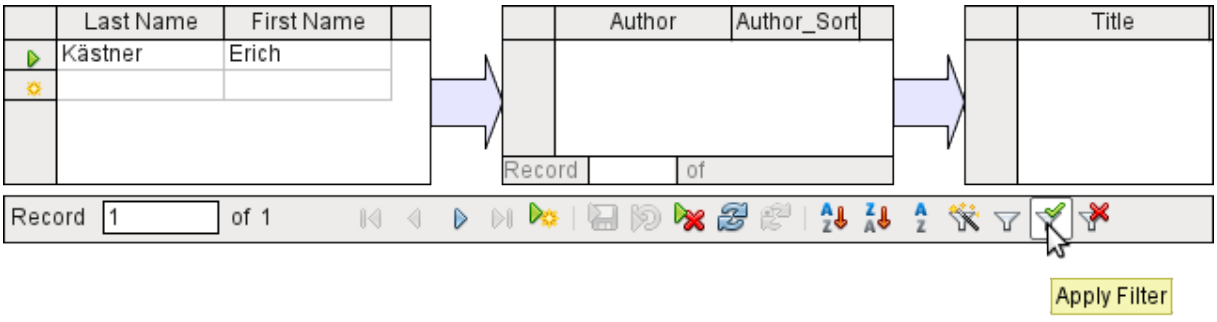
Defining relationships helps keep the relationships themselves clean, but it does not remove unnecessary records that provide their primary key as a foreign key in the relationship. There may be any number of streets without corresponding addresses.

Editing entries using forms and subforms

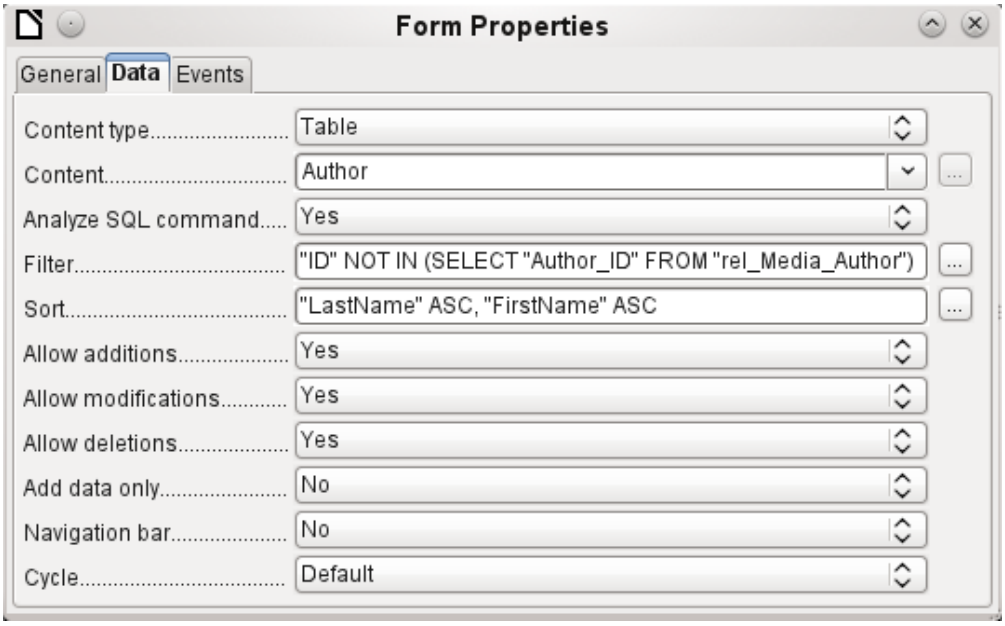
In principle the whole interrelationship between tables can be displayed within forms. This is easiest of course when a table is related to only one other table. Thus in the following example, the author’s primary key becomes the foreign key in the table rel_Media_Author. rel_Media_Author also contains a foreign key from Media, so that the following arrangement shows an n:m relationship with three forms. Each is presented through a table.

The first figure 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.*

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 sampleauthors are in use. Only the Erich Kästner record can be deleted without any consequences for any other record in Media.



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.



Queries for finding orphan entries

The above filter 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 they contain, the longer the form takes to load, since these listboxes must be created.

The better the Base program sets up the graphical interface and initially reads the listbox contents only partially, the less of a delay there will be.

Listboxes are created using queries, and these queries must be run when the form is loaded for each listbox.

The same query structure for more listboxes is better done using a common View, instead of 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 complete 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 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 return look into moving your data from the Base ODB file 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.