LibreOffice
The Document Foundation

Base

Kapitel 10
Wartung von Datenbanken

Copyright

Mitwirkende/Autoren

Robert Großkopf Jost Lange Jochen Schiffers

Michael Niedermair

Rückmeldung (Feedback)

Kommentare oder Vorschläge zu diesem Dokument können Sie in deutscher Sprache an die Adresse *discuss@de.libreoffice.org* senden.

Vorsicht 	Alles, was an eine Mailingliste geschickt wird, inklusive der E-Mail-Adresse und anderer persönlicher Daten, die die E-Mail enthält, wird öffentlich archiviert und kann nicht gelöscht werden. Also, schreiben Sie mit Bedacht!

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 03.08.2020. Basierend auf der LibreOffice Version 7.0.

Inhalt

Allgemeines zur Wartung von Datenbanken

Werden in einer Datenbank die Datenbestände häufig geändert, vor allem auch gelöscht, so macht sich das an zwei Stellen besonders bemerkbar. Zum einen wird die Datenbank immer größer, obwohl sie ja eigentlich gar nicht mehr Daten enthält. Zum anderen wird der automatisch erstellte Primärschlüssel einfach weiter hoch geschrieben ohne Rücksicht auf die tatsächlich erforderliche nächste Schlüsselzahl.

Datenbank komprimieren

Die HSQLDB hat die Eigenart, auch für bereits gelöschte Daten weiterhin Speicherplatz bereitzustellen. Datenbanken, die zum Test einmal mit Daten, vor allem auch Bildern gefüllt waren, weisen auch dann noch die gleiche Größe auf, wenn all diese Daten gelöscht wurden.

Um den Speicherplatz wieder frei zu geben, müssen die Datenbankdateien neu geschrieben werden (Tabellen, Beschreibungen zu diesen Tabellen usw.).

Direkt auf der Oberfläche von Base kann über **Extras → SQL** ein einfaches Kommando direkt eingegeben werden, dass bei Serverdatenbanken nur dem Systemadministrator vorbehalten ist:

```
SHUTDOWN COMPACT
```

Die Datenbank wird heruntergefahren und dabei von allen Altlasten befreit. Anschließend muss allerdings Base neu gestartet werden, wenn wieder auf die Datenbank zugegriffen werden soll.

Seit der Version LO 3.6 wird die Komprimierung standardmäßig spätestens beim Schließen der Datenbank durchgeführt.

Autowerte neu einstellen

Eine Datenbank wird erstellt, alle möglichen Funktionen mit Beispieldaten ausprobiert, Korrekturen vorgenommen, bis alles klappt. Dann ist mittlerweile der Wert für manch einen Primärschlüssel über 100 gestiegen. Pech nur, wenn der Primärschlüssel, wie meistens üblich, als Autowert-Schlüssel angelegt wurde. Werden die Tabellen zum Normalbetrieb oder zur Weitergabe der Datenbank an andere Personen geleert, so zählt anschließend der Primärschlüssel trotzdem munter weiter und startet nicht neu ab 0.

Mit dem folgenden SQL-Kommando, wieder eingegeben über **Extras → SQL**, lässt sich der Startwert neu festlegen:

```
ALTER TABLE "Tabellenname" ALTER COLUMN "ID" RESTART WITH NeuerWert
```

Hier wird davon ausgegangen, dass das Primärschlüsselfeld die Bezeichnung "ID" hat und außerdem als Autowert definiert wird. Der neue Wert sollte der sein, der als nächster automatisch eingefügt wird. Existieren z.B. noch Einträge bis zum Wert 4, so ist als neuer Wert 5 (ohne einfache Anführungsstriche) anzugeben.

Datenbankeigenschaften abfragen

In einem gesonderten Bereich der HSQLDB sind alle Informationen zu den Tabellen der Datenbank in Tabellenform abgelagert. Dieser besondere Bereich ist über den Zusatz "INFORMATION_SCHEMA" zu erreichen.

Mit der folgenden Abfrage können Feldnamen, Feldtypen, Spaltengrößen und Standardwerte ermittelt werden, hier am Beispiel der Tabelle mit dem Namen 'Suchtabelle'.

```
SELECT "COLUMN_NAME",
   "TYPE_NAME",
   "COLUMN_SIZE",
   "COLUMN_DEF" AS "Default Value"
FROM "INFORMATION_SCHEMA"."SYSTEM_COLUMNS"
WHERE "TABLE_NAME" = 'Suchtabelle'
ORDER BY "ORDINAL_POSITION"
```

Im Anhang sind alle Spezialtabellen der HSQLDB aufgeführt. Informationen über den Inhalt der jeweiligen Tabelle sind am einfachsten über direkte Abfragen zu erreichen:

```
SELECT * FROM "INFORMATION_SCHEMA"."SYSTEM_PRIMARYKEYS"
```

Über das Sternchen werden alle verfügbaren Spalten der Tabelle angezeigt. Die obige Tabelle gibt dabei in der Hauptsache Auskunft über die Primärschlüssel der verschiedenen Tabellen.

Diese Informationen lassen sich besonders mittels Makros gut nutzen. So können statt detaillierter Informationen zu einer gerade erstellten Tabelle oder Datenbank gleich Prozeduren geschrieben werden, die diese Informationen direkt aus der Datenbank holen und somit universeller nutzbar sind. Die Beispieldatenbank zeigt dies unter anderem an der Ermittlung von Fremdschlüsseln bei einem entsprechenden Wartungsmodul.

Die Firebird-Datenbank hat entsprechende Informationen, nur leider etwas mehr in ihren Systemdatenbanken verstreut:

```
SELECT "a".RDB$RELATION_NAME AS "Tables",
   "a".RDB$FIELD_NAME AS "Fields",
   "c".RDB$TYPE_NAME AS "Types",
   "a".RDB$Field_POSITION AS "Fieldposition",
   "a".RDB$NULL_FLAG AS "Nullflag",
   "b".RDB$FIELD_LENGTH AS "Fieldlength"
FROM RDB$RELATION_FIELDS AS "a", RDB$FIELDS AS "b", RDB$TYPES AS "c"
WHERE "a".RDB$FIELD_SOURCE = "b".RDB$FIELD_NAME
   AND "b".RDB$FIELD_TYPE = "c".RDB$TYPE
   AND "c".RDB$FIELD_NAME = 'RDB$FIELD_TYPE'
   AND "a".RDB$RELATION_NAME = 'Suchtabelle'
ORDER BY "Tables", "Fieldposition"
```

Weitere Informationen hierzu auch im Anhang.

Daten exportieren

Neben der Möglichkeit, Daten durch Öffnen der gepackten *.odb-Datei zu exportieren, existiert auch eine wesentlich einfachere Möglichkeit. Direkt auf der Oberfläche von Base kann über **Extras → SQL** ein einfaches Kommando in die HSQLDB direkt eingegeben werden, das bei Serverdatenbanken nur dem Systemadministrator vorbehalten ist:

```
SCRIPT 'Datenbank'
```

Dies erzeugt einen kompletten SQL-Auszug der Datenbank mit allen Definitionen der Tabellen, Beziehungen der Tabellen untereinander und Daten. Die daraus erstellte Datei wird auf dem Desktop abgelegt. Davon sind allerdings die Abfragen nicht berührt, da diese in der grafischen Benutzeroberfläche erstellt und nicht in der eigentlichen internen Datenbank gespeichert sind. Sehr wohl sind aber alle Ansichten (*Views*) enthalten.

Die Datei wird standardmäßig als normale Textdatei erzeugt. Sie kann allerdings vor allem für große Formate in binärem oder gepacktem Format ausgegeben werden. Nur ist dann der Transport zurück in LO etwas komplizierter.

Das Format der exportierten Datei lässt sich über

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

andern.

Die entsprechende Datei kann über **Extras → SQL** eingelesen werden und erzeugt so eine Datenbank mit den entsprechenden Daten. Dabei müssen für eine interne Datenbank allerdings die folgenden Zeilen entfernt werden:

```
CREATE SCHEMA PUBLIC AUTHORIZATION DBA
```

Das Schema existiert schon. Es wird also ein ungültiger Schema-Name bemängelt.

```
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 60
SET SCHEMA PUBLIC
```

Diese Eingaben haben etwas mit dem Benutzerprofil und anderen Standardeinstellungen zu tun. Die Einstellungen werden bei den internen Datenbanken von LO standardmäßig schon gesetzt. Diese Einträge befinden sich direkt vor den Inhalten, die in die Tabellen über "INSERT" eingefügt werden.

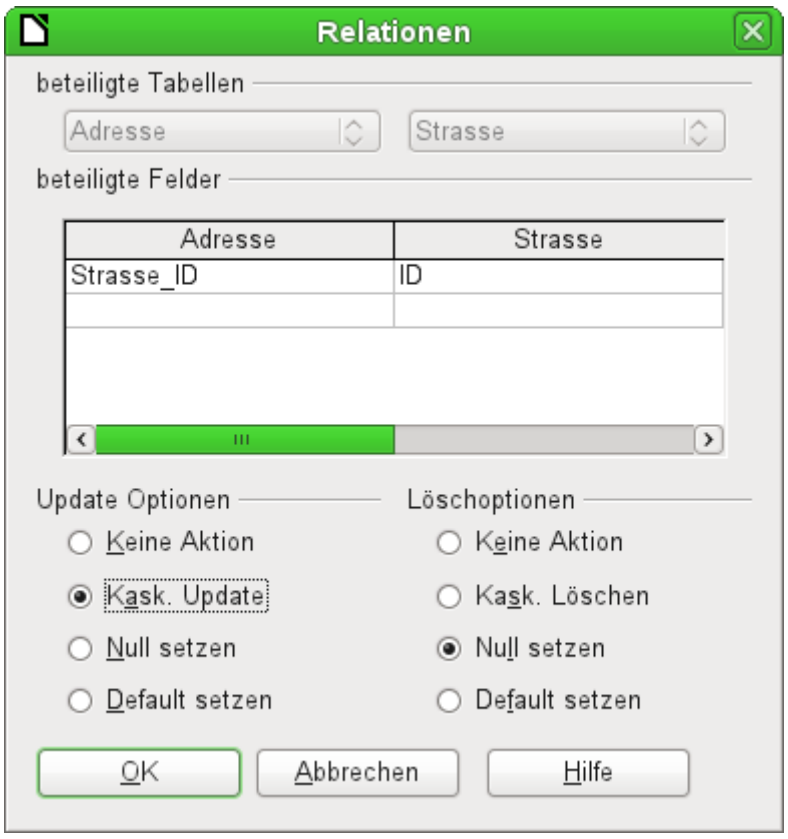
Tabellen auf unnötige Einträge überprüfen

Eine Datenbank besteht aus einer oder mehreren Haupttabellen, die Fremdschlüssel aus anderen Tabellen aufnehmen. In der Beispieldatenbank sind das besonders die Tabellen "Medien" und "Adresse". In der Tabelle "Adresse" wird der Primärschlüssel der Postleitzahl als Fremdschlüssel geführt. Zieht eine Person um, so wird die Adresse geändert. Dabei kann es vorkommen, dass zu der Postleitzahl anschließend überhaupt kein Fremdschlüssel "Postleitzahl_ID" mehr existiert. Die Postleitzahl könnte also gelöscht werden. Nur fällt im Normalbetrieb nicht auf, dass sie gar nicht mehr benötigt wird. Um so etwas zu unterbinden, gibt es verschiedene Methoden.

Einträge durch Beziehungsdefinition kontrollieren

Über die Definition von Beziehungen kann die Integrität der Daten abgesichert werden. Das heißt, dass verhindert wird, dass die Löschung oder Änderung von Schlüsseln zu Fehlermeldungen der Datenbank führt.

Das folgende Fenster steht in **Extras → Beziehungen** nach einem Rechtsklick auf die Verbindungslinie zwischen zwei Tabellen zur Verfügung.



Hier sind die Tabelle "Adresse" und "Strasse" betroffen. **Alle aufgeführten Aktionen gelten für die Tabelle "Adresse"**, die den Fremdschlüssel "Strasse_ID" enthält. Update-Optionen beziehen sich auf ein Update des Feldes "ID" aus der Tabelle "Strasse". Wenn also in dem Feld "Strasse"."ID" die Schlüsselnummer geändert wird, so bedeutet «Keine Aktion», dass sich die Datenbank gegen diese Änderung wehrt, wenn "Strasse"."ID" mit der entsprechenden Schlüsselnummer als Fremdschlüssel in der Tabelle "Adresse" verwandt wird.

Mit «Kask. Update» wird die Schlüsselnummer einfach mitgeführt. Wenn die Straße 'Burgring' in der Tabelle "Strasse" die "ID" '3' hat und damit auch in "Adresse"."Strasse_ID" verzeichnet ist, kann die "ID" gefahrlos auf z.B. '67' geändert werden – die "Adresse"."Strasse_ID" wird dabei automatisch auch auf '67' geändert.

Wird «Null setzen» gewählt, so erzeugt die Änderung der "ID" in "Adresse"."Strasse_ID" ein leeres Feld.

Entsprechend werden die Löschoptionen gehandhabt.

Für beide Optionen steht über die GUI die Möglichkeit «Default setzen» zur Zeit nicht zur Verfügung, da die GUI-Standardeinstellungen sich von denen der Datenbank unterscheiden. Siehe hierzu den Abschnitt «Default setzen» aus dem Kapitel Tabellen.

Die Beziehungsdefinition hilft also nur, die Beziehung selbst sauber zu halten, nicht aber unnötige Datensätze in der Tabelle zu entfernen, die ihren Primärschlüssel als Fremdschlüssel in der Beziehung zur Verfügung stellt. Es können also beliebig viele Straßen ohne Adresse existieren.

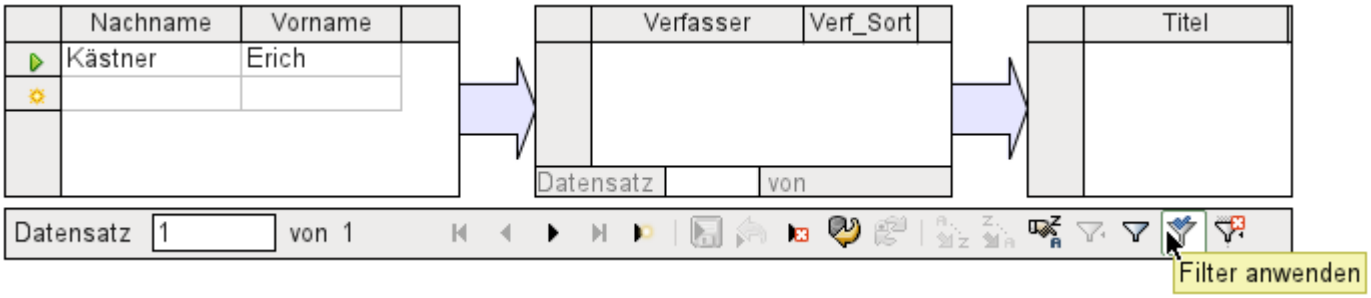
Einträge durch Formular und Unterformular bearbeiten

Vom Prinzip her kann der gesamte Zusammenhang zwischen Tabellen innerhalb von Formularen dargestellt werden. Am einfachsten ist dies natürlich, wenn eine Tabelle nur zu einer anderen Tabelle in Beziehung steht. So gibt z.B. in dem folgenden Beispiel der Verfasser seinen Primärschlüssel als Fremdschlüssel an die Tabelle "rel_Medien_Verfasser" weiter; "rel_Medien_Verfasser"

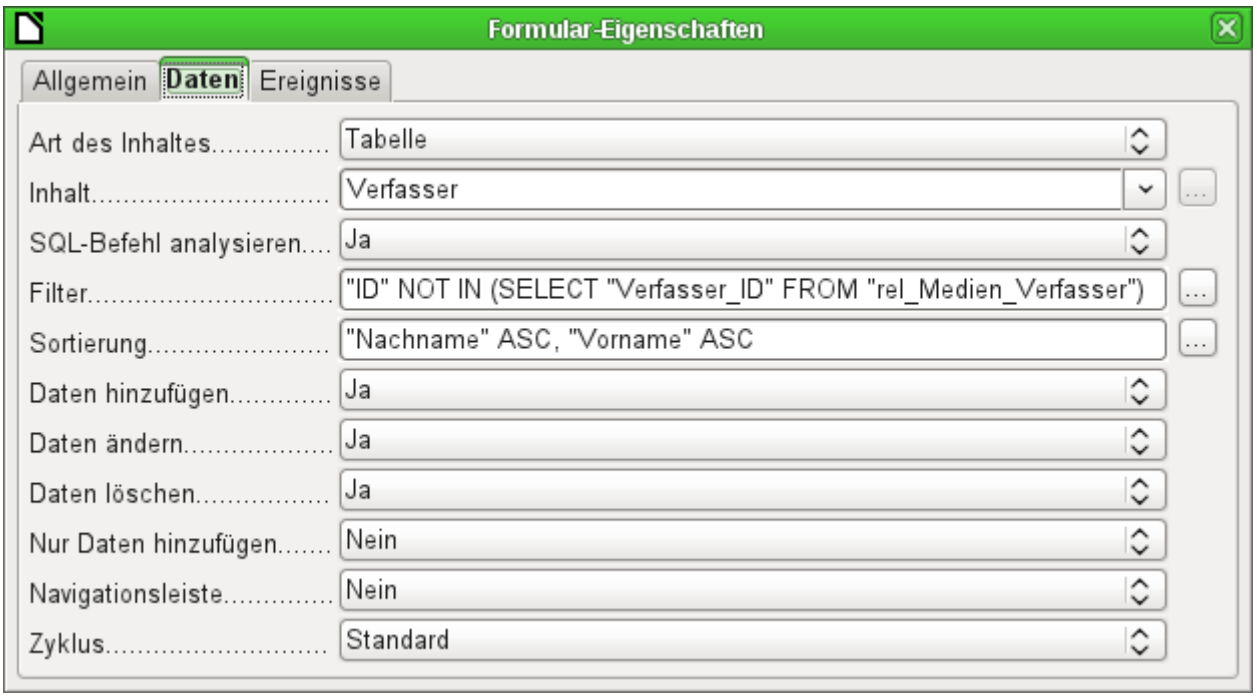
enthält außerdem einen Fremdschlüssel von "Medien", so dass die folgende Anordnung eine n:m-Beziehung mit drei Formularen aufzeigt. Jede wird durch eine Tabelle präsentiert.

Die erste Abbildung zeigt, dass der Titel 'I hear you knocking' dem Verfasser 'Dave Edmunds' zugeordnet wurde. 'Dave Edmunds' darf also nicht gelöscht werden – sonst fehlt eine Information zu dem Medium 'I hear you knocking'. Es kann aber durch das Listenfeld statt 'Dave Edmunds' ein anderer Datensatz ausgewählt werden.

In dem Formular ist ein Filter eingebaut, so dass bei Betätigung des Filters zu erkennen ist, welche Kategorien denn überhaupt nicht in der Tabelle Medien benötigt werden. In dem gerade abgebildeten Fall sind fast alle Beispielverfasser in Benutzung. Es kann also nur der Datensatz 'Erich Kästner' gelöscht werden, ohne dass dies eine Konsequenz für einen anderen Datensatz in "Medien" hätte.



Der Filter ist in diesem Fall fest vorgegeben. Er befindet sich in den Formular-Eigenschaften. Ein solcher Filter tritt direkt beim Start des Formulars automatisch in Kraft. Er kann ausgeschaltet und angewandt werden. Wurde er aber einmal gelöscht, so ist er nur dann wieder erreichbar, wenn das Formular komplett neu gestartet wurde. Das bedeutet, dass nicht nur die Daten zu aktualisieren sind, sondern das ganze Formulardokument erst geschlossen und dann wieder geöffnet werden muss.



Verwaiste Einträge durch Abfrage ermitteln

Der obige Filter ist Teil einer Abfrage, nach der die verwaisten Einträge ermittelt werden können.

```
SELECT "Nachname", "Vorname" FROM "Verfasser" WHERE "ID" NOT IN
(SELECT "Verfasser_ID“ FROM "rel_Medien_Verfasser")
```

Bezieht sich eine Tabelle mit Fremdschlüsseln auf mehrere andere Tabellen, so ist die Abfrage entsprechend zu erweitern. Dies trifft z.B. auf die Tabelle "Ort" zu, die sowohl in der Tabelle "Medien" als auch in der Tabelle "Postleitzahl" Fremdschlüssel liegen hat. Daten aus der Tabelle "Ort", die gelöscht werden, sollten also möglichst in keiner dieser Tabellen noch benötigt werden. Dies zeigt die folgende Abfrage auf:

```
SELECT "Ort" FROM "Ort" WHERE "ID" NOT IN (SELECT "Ort_ID“ FROM
"Medien") AND "ID" NOT IN (SELECT "Ort_ID“ FROM "Postleitzahl")
```

Verwaiste bzw. nicht benötigte Einträge können so durch die Markierung sämtlicher Einträge bei gesetztem Filter und Betätigung der rechten Maustaste über das Kontextmenü des Datensatzanzeigers gelöscht werden.

Datenbankgeschwindigkeit

Einfluss von Abfragen

Gerade Abfragen, die im vorherigen Abschnitt zur Filterung der Daten verwandt wurden, sollten allerdings im Hinblick auf die Geschwindigkeit einer Datenbank möglichst unterbleiben. Hier handelt es sich um Unterabfragen, die bei größeren Datenbanken eine entsprechend große Datenmenge für jeden einzelnen anzuzeigenden Datensatz zum Vergleich bereitstellen. Nur Vergleiche mit der Beziehung «IN» ermöglichen es überhaupt, einen Wert mit einer Menge von Werten zu vergleichen. In sofern kann die folgende Unterabfrage

```
… WHERE "ID" NOT IN (SELECT "Verfasser_ID“ FROM "rel_Medien_Verfas-
ser")
```

eine große Menge an vorhandenen Fremdschlüsseln der Tabelle "rel_Medien_Verfasser" ergeben, die erst mit dem Primärschlüssel der Tabelle "Verfasser" für jeden Datensatz der Tabelle "Verfas-

ser" verglichen werden muss. So eine Abfrage sollte also nicht für den täglichen Gebrauch gedacht sein, sondern nur vorübergehend wie hier zum Beispiel der Wartung von Tabellen. Suchfunktionen sind anders aufzubauen, damit die Suche von Daten nicht endlos dauert und die Arbeit mit der Datenbank im Alltagsbetrieb verleidet.

Einfluss von Listenfeldern und Kombinationsfeldern

Je mehr Listenfelder in einem Formular eingebaut sind und je größer der Inhalt ist, den sie bereitstellen müssen, desto länger braucht das Formular zum Aufbau.

Je besser das Programm Base zuerst einmal die grafische Oberfläche bereitstellt und erst einmal Listenfelder nur teilweise einliest, desto weniger fällt eine entsprechende Last auf.

Listenfelder werden durch Abfragen erstellt, und diese Abfragen finden beim Formularstart für jedes Listenfeld statt.

Gleiche Abfragestrukturen für z.B. mehrere Listenfelder können besser in eine gemeinsame Tabellenansicht (*View*) ausgelagert werden, statt mehrmals hintereinander mit gleicher Syntax über in den Listenfeldern abgespeicherte SQL-Kommandos entsprechende Felder zu erstellen. Ansichten sind vor allem bei externen Datenbanksystemen vorzuziehen, da hier der Server deutlich schneller läuft als eine Abfrage, die erst von der GUI zusammengestellt und an den Server neu gestellt wird. Ansichten (*Views*) hält der Server als fertige Abfragen schließlich vorrätig.

Einfluss des verwendeten Datenbanksystems

Die interne HSQLDB ist auf eine gut funktionierende Zusammenarbeit von Base mit Java ausgelegt. Leider hat Base seit der LO-Version 3.5 mit Geschwindigkeitsproblemen gerade in dieser Kombination zu kämpfen. Dies macht sich vor allem bei großen Tabellen mit mehreren tausend Datensätzen bemerkbar. Diese Geschwindigkeitsprobleme hatten unterschiedliche Ursachen und sind erst mit der LO-Version 4.1 verschwunden.

Externe Datenbanken laufen hier deutlich schneller. Von der Geschwindigkeit sind die direkten Verbindungen zu MySQL oder PostgreSQL sowie die Verbindungen über ODBC nahezu gleichwertig. JDBC ist ebenfalls auf das Zusammenspiel mit Java angewiesen, funktioniert aber deutlich schneller als eine interne Verbindung zur HSQLDB.