LibreOffice
The Document Foundation

Base

Kapitel 3
Tabellen

Copyright

Mitwirkende/Autoren

Robert Großkopf Jost Lange Michael Niedermair

Jochen Schiffers

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 1.3.2013. Basierend auf der LibreOffice Version 4.0.

Anmerkung für Macintosh Nutzer

Einige Tastenbelegungen (Tastenkürzel) und Menüeinträge unterscheiden sich zwischen der Macintosh Version und denen für Windows- und Linux-Rechnern. Die unten stehende Tabelle gibt Ihnen einige grundlegende Hinweise dazu. Eine ausführlichere Aufstellung dazu finden Sie in der Hilfedatei des jeweiligen Moduls.

Windows/Linux	***entspricht am Mac***	***Effekt***
Menü-Auswahl **Extras → Optionen**	**LibreOffice → Einstellungen**	Zugriff auf die Programmoptionen
Rechts-Klick	*Control*+Klick	Öffnen eines Kontextmenüs
Ctrl (Control) oder *Strg* (Steuerung)	⌘ *(Command)*	Tastenkürzel in Verbindung mit anderen Tasten
F5	*Shift*+⌘+*F5*	öffnet den Dokumentnavigator Dialog
F11	⌘+*T*	öffnet den Formatvorlagen Dialog

Inhalt

Allgemeines zu Tabellen

Daten werden in Datenbanken innerhalb von Tabellen gespeichert. Wesentlicher Unterschied zu Tabellen innerhalb einer einfachen Tabellenkalkulation ist, dass die Felder, in die geschrieben wird, klar vordefiniert werden müssen. Eine Datenbank erwartet innerhalb einer Textspalte keine Zahleneingaben, mit denen sie rechnen kann. Sie stellt die Zahlen dann zwar dar, geht aber von einem Wert «0» für diese Zahlen aus. Auch Bilder lassen sich nicht in jeder Feldform ablegen.

Welche Datentypen es im einzelnen gibt, kann bei der grafischen Benutzeroberfläche dem Tabelleneditor entnommen werden. Details dafür im Anhang dieses Handbuches.

Einfache Datenbanken beruhen lediglich auf einer Tabelle. Hier werden alle Daten unabhängig davon eingegeben, ob eventuell mehrfache Eingaben des gleichen Inhaltes gemacht werden müssen. Eine einfache Adressensammlung für den Privatgebrauch lässt sich so erstellen. Die Adressensammlung einer Schule oder eines Sportvereins würde aber so viele Wiederholungen bei Postleitzahl und Ort aufweisen, dass diese Tabellenfelder in eine oder sogar 2 separate Tabellen ausgelagert würden. Die Auslagerung von Informationen in andere Tabellen hilft:

- laufend wiederkehrende Eingaben gleichen Inhaltes zu reduzieren
- Schreibfehler bei diesen laufenden Eingaben zu vermeiden
- Daten besser nach den ausgelagerten Tabellen zu filtern

Bei der Erstellung der Tabellen sollte also immer wieder überlegt werden, ob eventuell viele Wiederholungen vor allem von Texten oder Bildern (hier stecken die Speicherfresser) in den Tabellen vorkommen. Dann empfiehlt sich eine Auslagerung der Tabelle. Wie dies prinzipiell geht ist im Handbuch «Erste Schritte Handbuch» «*Einführung in Base*» bereits beschrieben.

Hinweis	In einer Datenbank, in der mehrere Tabellen in Beziehung zueinander stehen ("relationale Datenbank"), wird angestrebt, möglichst wenige Daten in einer Tabelle doppelt einzugeben. Es sollen «Redundanzen» vermieden werden. Dies kann erreicht werden, • indem Tabellenfelder nicht zu viel Inhalt auf einmal speichern (z.B. nicht eine komplette Adresse mit Straße, Hausnummer, Postleitzahl und Ort), sondern Straße, Hausnummer, Postleitzahl und Ort getrennt, • doppelte Angaben in einem Feld vermieden werden (z.B. Postleitzahl und Ort aus einer Tabelle in eine andere auslagern) Dieses Vorgehen wird als Normalisierung von Datenbanken bezeichnet.

Beziehungen von Tabellen

Anhand der Beispieldatenbanken «Medien_ohne_Makros» bzw. «Medien_mit_Makros» werden in diesem Handbuch viele Schritte möglichst detailliert erklärt. Bereits die Tabellenkonstruktion dieser Datenbank ist sehr umfangreich, da sie neben der Aufnahme von Medien in eine Mediothek auch die Ausleihe von Medien abdeckt.

Beziehungen zwischen Tabellen allgemein

Tabellen in der internen Datenbank HSQLDB haben immer ein unverwechselbares, einzigartiges Feld, den Primärschlüssel. Dieses Feld muss definiert sein, bevor überhaupt Daten in die Tabelle geschrieben werden können. Anhand dieses Feldes können bestimmte Datensätze einer Tabelle ermittelt werden.

Nur in Ausnahmefällen wird ein Primärschlüssel auch aus mehreren Feldern zusammen gebildet. Dann müssen diese Felder zusammen einzigartig sein.

Tabelle 2 kann ein Feld besitzen, das auf die Inhalte von Tabelle 1 hinweist. Hier wird der Primärschlüssel aus Tabelle 1 als Wert in das Feld der Tabelle 2 geschrieben. Tabelle 2 hat jetzt ein Feld, das auf ein fremdes Schlüsselfeld verweist, also einen Fremdschlüssel. Dieser Fremdschlüssel existiert in Tabelle 2 neben dem Primärschlüssel.

Je mehr Tabellen in Beziehung zueinander stehen, desto komplexer kann der Entwurf sein. Das folgende Bild zeigt die gesamte Tabellenstruktur der Beispieldatenbank in einer Übersicht, die von der Größe her die Seite dieses Dokuments sprengt:

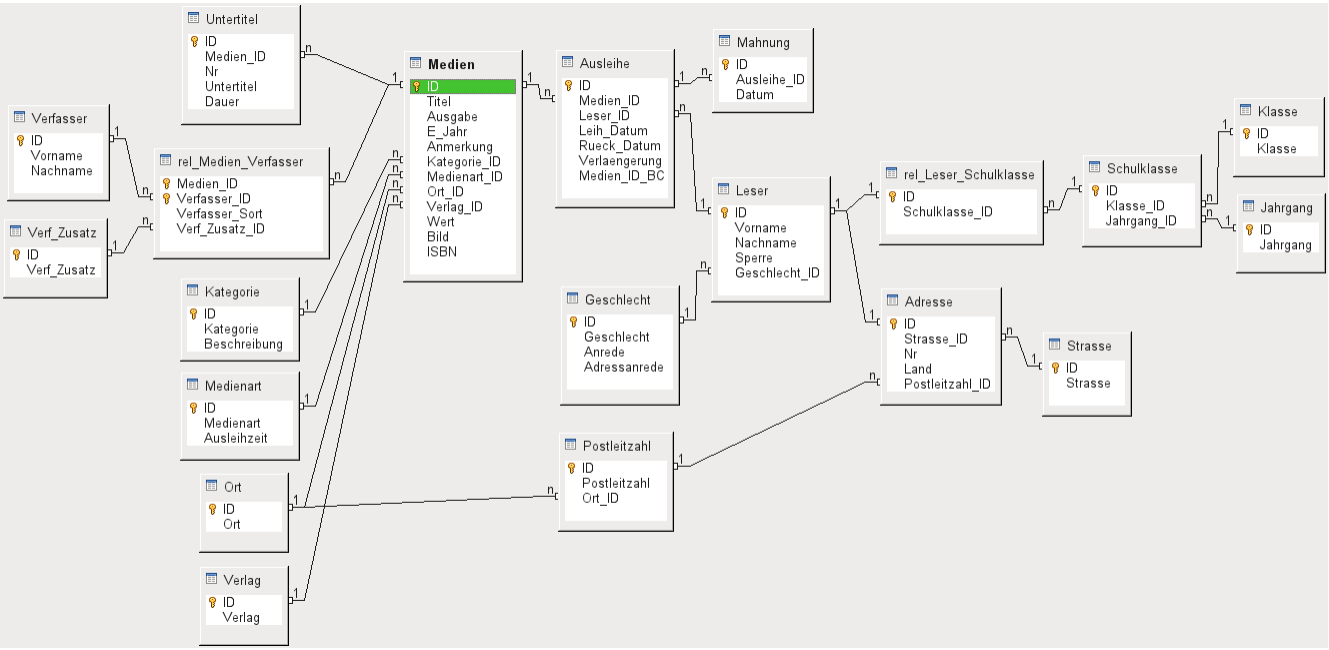


Abbildung 1: Beziehungsentwurf der Beispieldatenbank «Medien_ohne_Makros"

Eins-zu-Viele Beziehungen:

Eine Datenbank für Medien listet in einer Tabelle die Titel der Medien auf. Da es für jeden Titel unterschiedlich viele Untertitel gibt (manchmal auch gar keine) werden in einer gesonderten Tabelle diese Untertitel abgespeichert. Dies ist als eine Eins-zu-viele-Beziehung (1:n) bekannt. Einem Medium werden gegebenenfalls viele Untertitel zugeordnet, z.B. bei dem Medium Musik-CD die vielen Musiktitel auf dieser CD. Der Primärschlüssel der Tabelle "Medien" wird als Fremdschlüssel in der Tabelle "Untertitel" abgespeichert. Die meisten Beziehungen zwischen Tabellen in einer Datenbank sind Eins-zu-viele Beziehungen.

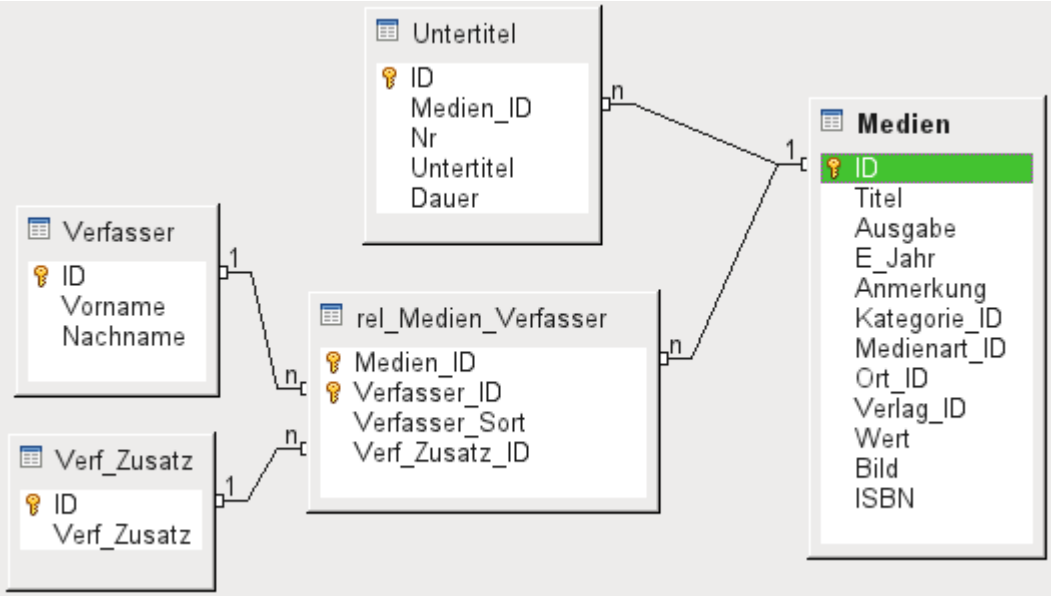


Abbildung 2: Beispiel 1:n-Beziehung; n:m-Beziehung

Viele-zu-Viele Beziehungen:

Eine Datenbank für eine Bibliothek wird eine Tabelle für den Namen der Verfasser und eine Tabelle für die Medien enthalten. Es gibt einen offensichtlichen Zusammenhang zwischen den Verfasser und z.B. Büchern, die sie geschrieben haben. Die Bibliothek kann mehr als ein Buch desselben Verfassers enthalten. Sie kann aber auch Bücher enthalten, die von mehreren Verfassern stammen. Dies ist als eine Viele-zu-viele-Beziehung (n:m) bekannt. Solche Beziehungen werden durch Tabellen gelöst, die als Mittler zwischen den beiden betroffenen Tabellen eingesetzt werden. Dies ist in der obigen Abbildung die Tabelle "rel_Medien_Verfasser".

Praktisch wird also die n:m-Beziehung über zwei 1:n-Beziehungen gelöst. In der Mittlertabelle kann die "Medien_ID" mehrmals erscheinen, ebenso die "Verfasser_ID". Dadurch, dass beide zusammen den Primärschlüssel ergeben ist nur ausgeschlossen, dass zu einem Medium wiederholt der gleiche Verfasser gewählt wird.

Eins-zu-Eins-Beziehung:

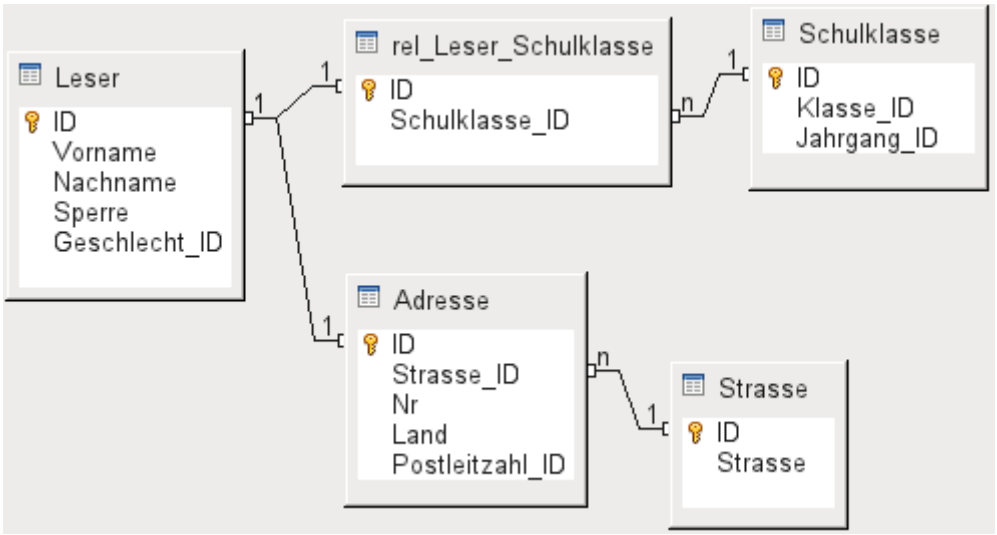


Abbildung 3: Beispiel 1:1-Beziehung

Die bereits angesprochene Bibliotheks-Datenbank enthält eine Tabelle für die Leser. In dieser Tabelle sind erst einmal nur die direkt notwendig erscheinenden Felder vorgesehen. Für eine

Datenbank im Bereich von Schulen würde noch die jeweilige Schulklasse benötigt. Über diese Klasse kann gegebenenfalls auch die Adresse erfahren werden. Eine Aufnahme der Adresse in die Datenbank ist also nicht notwendig. Die Klassenbeziehung des Schülers ist aus der Lesertabelle ausgegliedert, weil nicht in allen Bereichen mit der Zuordnung zu Klassen etwas angefangen werden kann. Dadurch entsteht eine 1:1-Beziehung zwischen Leser und seiner ganz persönlichen Klassenzuweisung.
Handelt es sich um eine Datenbank im öffentlichen Bereich, so wird wohl die Adresse der Leser benötigt. Einem Leser wird hier genau eine Adresse zugeordnet. Würde es mehrere Leser mit der gleichen Adresse geben, so müsste das bei dieser Konstruktion zu einer Neueingabe der Adresse führen, denn der Primärschlüssel aus der Tabelle "Leser" wird direkt als Primärschlüssel in die Tabelle "Adresse" eingetragen. Primärschlüssel und Fremdschlüssel sind in der Tabelle "Adresse" eins. Hier besteht also eine 1:1-Beziehung.
Eine 1:1-Beziehung bedeutet nicht, dass automatisch zu jedem Datensatz der einen Tabelle auch ein Datensatz der anderen Tabelle existiert. Es existiert allerdings **höchstens** ein Datensatz. Durch die 1:1-Beziehung werden also Felder ausgelagert, die vermutlich nur bei einem Teil der Datensätze mit Inhalt gefüllt sein werden.

Tabellen und Beziehungen der Beispieldatenbank

Die Beispieldatenbank muss drei verschiedene Aufgabenbereiche erfüllen. Zuerst einmal müssen Medien in die Datenbank aufgenommen werden. Dies soll so erfolgen, dass auch eine Bibliothek damit arbeiten kann. Für eine einfache Übersicht über die Medien, die zu Hause lagern, reichen dagegen die Beispieldatenbanken aus dem Handbuch «Erste Schritte Base» aus.

Tabellen Medienaufnahme

Zentrale Tabelle der *Medienaufnahme* ist die Tabelle **Medien.** In dieser Tabelle werden alle Felder direkt verwaltet, die vermutlich nicht auch von anderen Medien mit dem gleichen Inhalt belegt werden. Doppelungen sollen also vermieden werden.

Aus diesem Grund sind in der Tabelle z.B. der Titel, die ISBN-Nummer, ein Bild des Umschlags oder das Erscheinungsjahr vorgesehen. Die Liste der Felder kann hier entsprechend erweitert werden. So sehen Bibliotheken z.B. Felder für den Umfang (Seitenanzahl), den Reihentitel und ähnlichem vor.

Die Tabelle **Untertitel** soll dazu dienen, z.B. den Inhalt von CDs im Einzelnen aufzunehmen. Da auf einer CD mehrere Musikstücke (*Untertitel*) vorhanden sind würde eine Aufnahme der Musikstücke in die Haupttabelle dazu führen, dass entweder viele zusätzliche Felder (Untertitel 1, Untertitel 2 usw.) erstellt werden müssten oder das gleiche Medium mehrmals hintereinander eingegeben werden müsste. Die Tabelle *Untertitel* steht also in einer *n:1-Beziehung* zu der Tabelle *Medien*.

Felder der Tabelle *Untertitel* sind neben dem Untertitel selbst die Nummerierung der Reihenfolge der Titel und die Dauer der Untertitel. Das Feld *Dauer* ist erst einmal als ein Zeitfeld vorgesehen. So kann gegebenenfalls die Gesamtdauer der CD in einer Übersicht berechnet und ausgegeben werden.

Die Verfasser haben zu den Medien eine n:m-Beziehung. Ein Medium kann mehrere Verfasser haben, ein Verfasser kann mehrere Medien herausgebracht haben. Dies wird mit der Tabelle **rel_Medien_Verfasser** geregelt. Primärschlüssel dieser Verbindungstabelle sind die Fremdschlüssel, die aus der Tabelle **Verfasser** und **Medien** ausgegeben werden. In der Tabelle *rel_Medien_Verfasser* wird zusätzlich noch eine Sortierung der Verfasser vorgenommen (z.B. nach der Reihenfolge, wie sie im Buch genannt werden). Außerdem wird gegebenenfalls ein Zusatz wie «Herausgeber», «Fotograf» o.ä. dem jeweiligen Verfasser beigefügt.

Kategorie, Medienart, Ort und Verlag haben jeweils eine 1:n-Beziehung.

In der **Kategorie** kann bei kleinen Bibliotheken so etwas stehen wie «Kunst», «Biologie» … Bei größeren Bibliotheken gibt es gängige Systematiken wie z.B. die ASB (allgemeine Systematik für

Bibliotheken). Bei dieser Systematik gibt es Kürzel und ausführlichere Beschreibungen. Daher die beiden Felder für die Kategorie.

Die **Medienart** ist gekoppelt mit der *Ausleihzeit.* So kann es z.B. sein, dass Video-DVDs grundsätzlich nur eine Ausleihzeit von 1 Woche haben, Bücher aber eine von 3 Wochen. Wird die Ausleihzeit an ein anderes Kriterium gekoppelt, so muss entsprechend anders verfahren werden.

Die Tabelle **Ort** dient nicht nur dazu, die Ortsbenennungen aus den Medien aufzunehmen. In ihr werden gleichzeitig die Orte gespeichert, die für die Adressen der Nutzer Bedeutung haben.

Da der **Verlag** vermutlich auch häufiger vorkommt, ist für seine Eingabe ebenfalls eine gesonderte Tabelle vorgesehen.

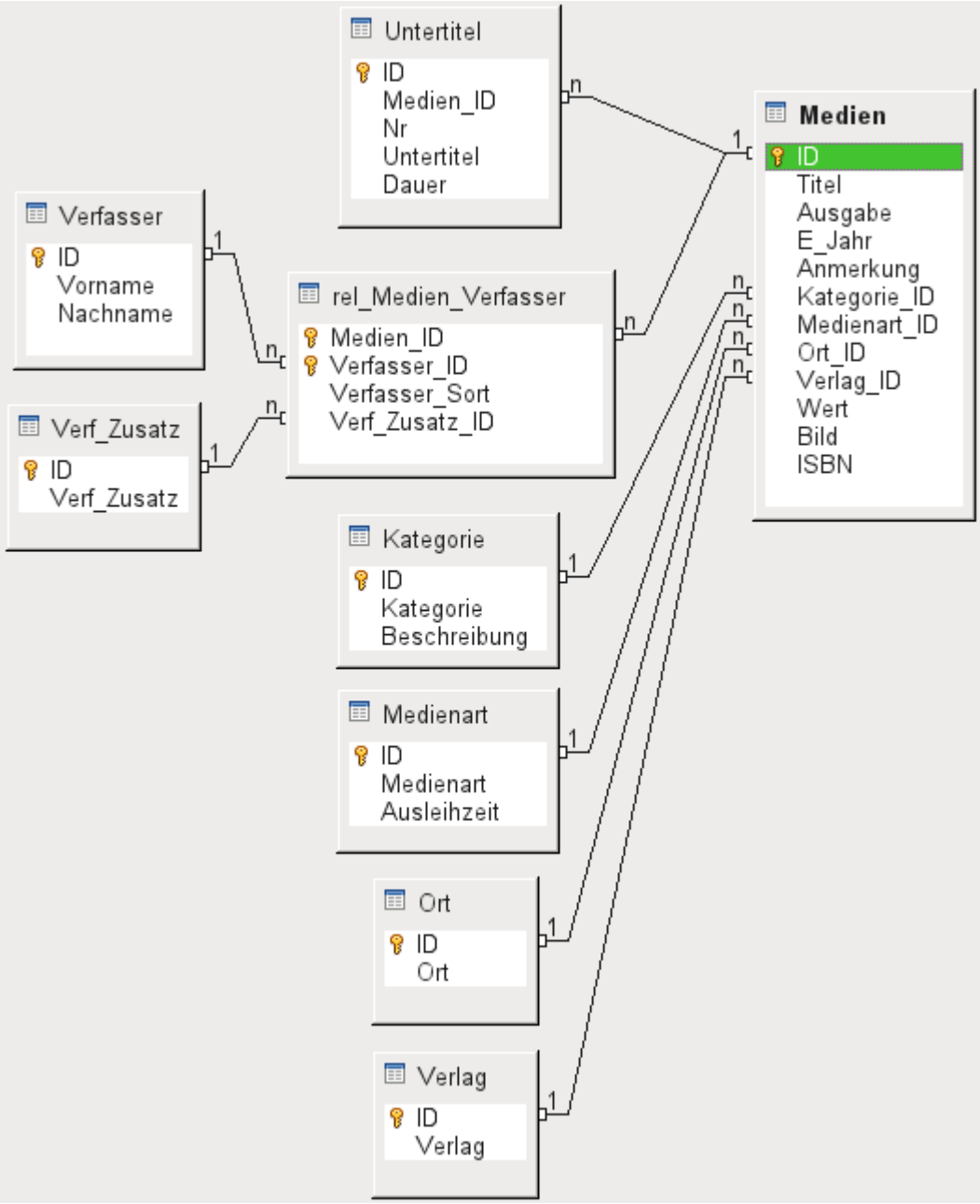


Abbildung 4: Medienaufnahme

Die Tabelle Medien hat so insgesamt vier Fremdschlüssel und einen Primärschlüssel, der für 2 Tabellen der Abbildung *Medienaufnahme* zum Fremdschlüssel wird.

Tabellen Ausleihe

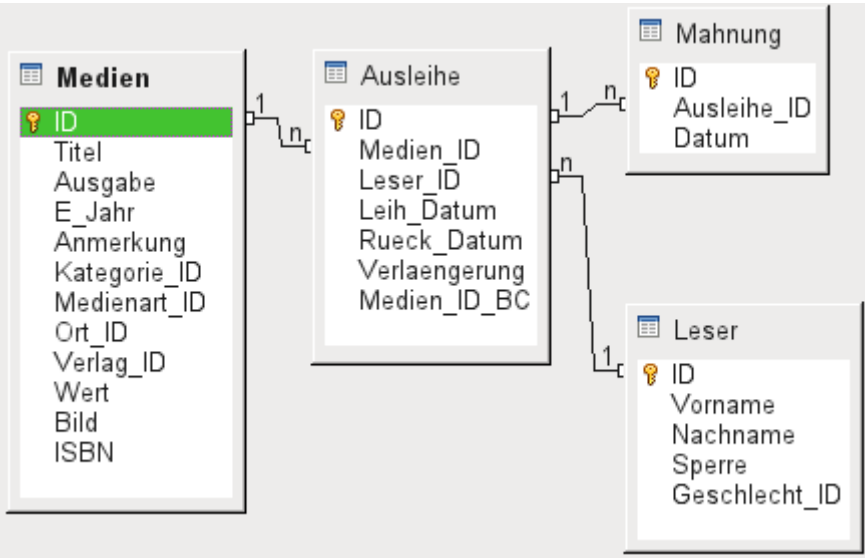


Abbildung 5: Ausleihe

Zentrale Tabelle ist die **Ausleihe**. In ihr werden die Tabellen Medien und Leser verknüpft. Da auch später noch nachvollzogen werden soll, wer ein Buch ausgeliehen hat (falls z.B. jemand beim Ausleihen bemerkt, dass das Buch beschädigt ist, oder falls eine Hitliste der Medien erstellt werden soll) wird der Datensatz in der Ausleihe bei der Rückgabe nicht einfach gelöscht. Vielmehr wird ein Rückgabedatum (*Rueck_Datum*) vermerkt.

Ebenso in die Ausleihe integriert ist das Mahnverfahren. Um die Anzahl der Mahnungen zu erfassen wird jede Mahnung separat in der Tabelle **Mahnung** eingetragen.

Neben der Verlängerung um einzelne Wochen steht noch ein gesondertes Feld in der Ausleihe, das es ermöglicht, Medien über einen Barcodescanner zu entleihen (*Medien_ID_BC*). Barcodes enthalten neben der eigentlichen "Medien_ID" auch eine Prüfziffer, mit der das Gerät feststellen kann, ob der eingelesene Wert korrekt ist. Dieses Barcodefeld ist hier nur testweise eingebaut. Besser wäre es, wenn der Primärschlüssel der Tabelle Medien direkt in Barcode-Form eingegeben würde oder per Makro aus der eingelesenen Barcodeziffer einfach die Prüfziffer vor dem Abspeichern entfernt wird.

Schließlich ist noch der **Leser** mit der Ausleihe in Verbindung zu bringen. In der eigentlichen Lesertabelle wird lediglich der Name, eine eventuelle Sperrung und ein Fremdschlüssel für eine Verbindung zur Tabelle Geschlecht vorgesehen.

Tabellen Nutzerverwaltung

In dieser Tabellenkonstruktion werden gleich zwei Szenarien bedient. Der obere Tabellenstrang ist dabei auf Schulen zugeschnitten. Hier werden keine Adressen benötigt, da die Schüler und Schülerinnen über die Schule selbst ansprechbar sind. Mahnungen müssen nicht postalisch zugestellt werden, sondern auf dem internen Wege weitergegeben werden.

Der Adressstrang ist dagegen bei öffentlichen Bibliotheken notwendig. Hier müssen sämtliche Daten erfasst werden, die zu Erstellung eines Mahnbriefes erforderlich sind.

Die Tabelle **Geschlecht** dient dazu, die richtige Anrede bei Mahnschreiben zu wählen. Die Mahnschreiben sollen schließlich möglichst automatisiert erfolgen. Außerdem gibt es Vornamen, die sowohl für männliche als auch für weibliche Leser stehen können. Deswegen ist die Abspeicherung des Geschlechts auch bei der Erstellung von handgeschriebenen Mahnungen sinnvoll.

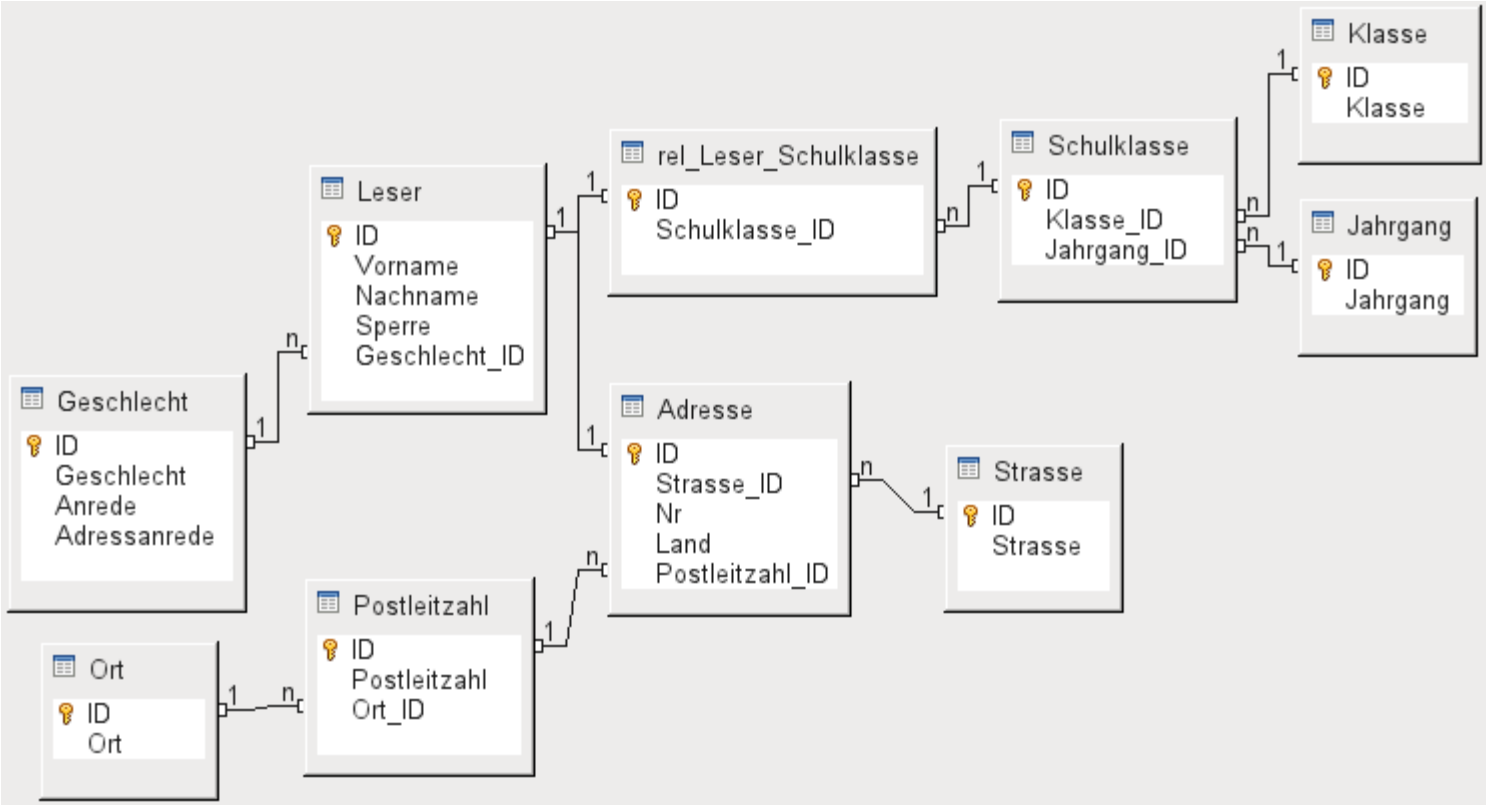


Abbildung 6: Leser - ein Schulklassenstrang und ein Adressenstrang

Die Tabelle **rel_Leser_Schulklasse** steht wie die Tabelle Adresse in einer 1:1-Beziehung zu der Tabelle *"Leser"*. Dies ist gewählt worden, weil entweder die eine oder die andere Möglichkeit beschritten werden soll. Sonst könnte die *"Schulklasse_ID"* direkt in der Tabelle Schüler existieren; gleiches gilt für den gesamten Inhalt der Tabelle Adresse.

Eine **Schulklasse** wird in der Regel durch eine Jahrgangsbezeichnung und einen Klassenzusatz gekennzeichnet. Bei einer 4-zügigen Schule kann dieser Zusatz z.B. von a bis d gehen. Der Zusatz wird in der Tabelle *"Klasse"* eingetragen. Der Jahrgang hat eine separate Tabelle. Sollen am Schluss eines Schuljahres die Leser aufgestuft werden, so wird einfach der Jahrgang für alle geändert.

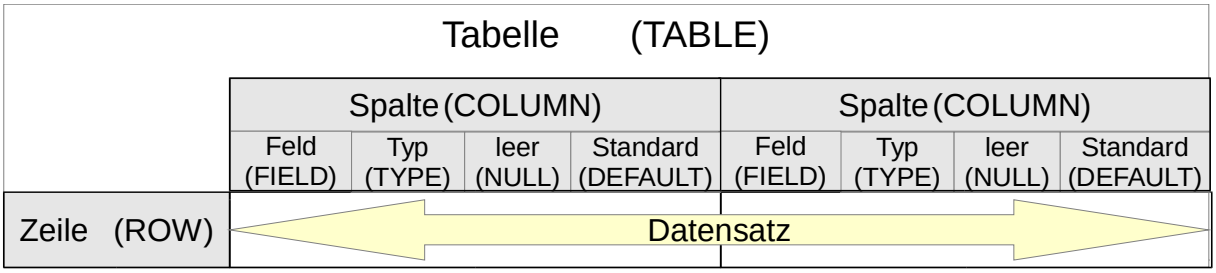
Die **Adresse** wird ebenfalls sehr differenziert dargestellt. Die Straße ist aus der Adresse ausgelagert, da Straßennahmen innerhalb eines Ortes häufiger wiederholt werden. Postleitzahl und Ort sind voneinander getrennt, da oft mehrere Postleitzahlen für einen Ort gelten. Für die Post sind alle Ortsbezeichnungen, die auf die gleiche Postleitzahl passen, in einem Ort zusammengefasst. Es existieren postalisch also deutlich mehr Postleitzahlen als Orte. So werden von der Tabelle Adresse aus gesehen deutlich weniger Datensätze in der Tabelle **Postleitzahl** stehen und noch einmal deutlich weniger Datensätze in der Tabelle **Ort** existieren.

Wie eine derartige Tabellenkonstruktion später sinnvoll zu befüllen ist, wird weiter unten im Kapitel *Formulare* erläutert.

Erstellung von Tabellen

In der Regel wird sich der LibreOffice-User auf die Erstellung von Tabellen mit der grafischen Benutzeroberfläche beschränken. Die direkte Eingabe von SQL-Befehlen ist dann sinnvoll, wenn z.B. ein Tabellenfeld nachträglich an einer bestimmten Position eingefügt werden soll oder Standardwerte nach Abspeicherung der Tabelle noch gesetzt werden sollen.

Bezeichnungen bei Tabellen:



Die obige Skizze zeigt die allgemein übliche Aufteilung von Tabellen in Spalten und Zeilen. Die entsprechenden Datenbankbezeichnungen sind in Klammern hinzugefügt.

Datensätze werden in der Tabelle in einer Zeile gespeichert. Die einzelnen Spalten werden durch das Feld, den Typ und die Festlegung, ob das Feld leer sein darf, weitgehend beschrieben. Je nach Typ kann noch der Umfang an Zeichen festgelegt werden. Außerdem kann ein Standardwert eingegeben werden, der immer dann abgespeichert wird, wenn keine Eingabe erfolgt.

In der grafischen Benutzeroberfläche von Base sind die Begriffe einer Spalte etwas anders umschrieben:

Bezeichnungen in der Base-GUI			
Spalte (COLUMN)			
		Feldeigenschaften	
Feldname (FIELD)	Feldtyp (TYPE)	Eingabe erforderlich (NULL/NOT NULL)	Defaultwert (DEFAULT)

Feld wird zu Feldname, Typ wird zu Feldtyp. Feldname und Feldtyp werden im oberen Bereich des Tabelleneditors eingegeben. Im unteren Bereich gibt es dann die Möglichkeit, unter den Feldeigenschaften die anderen Spalteneigenschaften festzulegen, sofern dies durch die GUI festlegbar ist. Grenzen sind hier z.B., den Defaultwert eines Datumsfeldes mit dem bei der Eingabe aktuellen Datum festzulegen. Dies geht nur über eine entsprechende SQL-Eingabe, siehe dazu: *Direkte Eingabe von SQL-Befehlen*.

Hinweis	Defaultwert: Der Begriff «Defaultwert» in der GUI entspricht nicht dem, was Datenbanknutzer unter Defaultwert verstehen. Die GUI gibt hier einen bestimmten Wert sichtbar vor, der dann mit abgespeichert wird. Der Defaultwert einer Datenbank wird in der Tabellendefinition gespeichert. Er wird dann in das Feld geschrieben, wenn es bei der neuen Erstellung eines Datensatzes leer bleibt. SQL-Defaultwerte erscheinen auch **nicht** *bei der Bearbeitung der Eigenschaften einer Tabelle.*

Erstellung mit der grafischen Benutzeroberfläche

Die Erstellung innerhalb der grafischen Benutzeroberfläche ist in «Erste Schritte – Einführung in Base» ausführlich beschrieben. Hier deshalb nur die Hauptfehlerquellen:

Beim Abspeichern eines Tabellenentwurfs erscheint die Nachfrage, ob ein Primärschlüssel erstellt werden soll. Dies deutet darauf hin, dass ein wesentliches Feld in der Tabelle fehlt. Ohne einen Primärschlüssel kann die HSQLDB-Datenbank auf die Tabelle nicht zugreifen. In der Regel wird dieses Feld mit dem Kürzel "ID" bezeichnet, mit dem Zahlentyp INTEGER versehen und als «Auto-Wert» automatisch mit einer fortlaufenden Nummer versehen. Mit einem Rechtsklick auf das entsprechende Feld kann es zum Primärschlüsselfeld erklärt werden.

Hinweis	Wird nicht direkt beim Anlegen der Tabelle in der grafischen Benutzeroberfläche der Primärschlüssel festgelegt, so ist die anschließend über die grafische Benutzeroberfläche auch nicht mehr möglich. (*Bug 61547*) Stattdessen muss der Primärschlüssel über **Extras** → **SQL** erstellt werden: **`ALTER TABLE "Tabellenname" ADD PRIMARY KEY ("Feldname")`**

Sollen von einer anderen Tabelle in dieser Tabelle Informationen mitgeführt werden (Beispiel: Adressdatenbank, aber ausgelagert Postleitzahlen und Orte), so ist ein Feld mit dem gleichen Datentyp wie dem des Primärschlüssels der anderen Tabelle in die Tabelle aufzunehmen. Angenommen die Tabelle "PLZ_Ort" hat als Primärschlüssel das Feld "ID", als Datentyp 'Tiny Integer'. In der Tabelle Adressen erscheint jetzt ein Feld "ID_PLZ_Ort" mit dem Datentyp 'Tiny Integer'. Es wird also in der Tabelle Adressen immer nur die Zahl eingetragen, die als Primärschlüssel in der Tabelle "PLZ_Ort" steht. Für die Tabelle "Adresse" heißt das: Sie hat einen Fremdschlüssel zusätzlich zum eigenen Primärschlüssel bekommen.

Grundlage bei der Namenswahl von Feldern in der Tabelle: Keine 2 Felder dürfen gleich heißen. Deswegen darf auch nicht ein zweites Feld mit der Bezeichnung "ID" als Fremdschlüssel in der Tabelle "Adressen" auftauchen.

Die Feldtypen können nur begrenzt geändert werden. Eine Aufstufung (längeres Textfeld, größerer Zahlenumfang) ist ohne weiteres möglich, da alle eventuell schon eingegebenen Werte diesem Feldtyp entsprechen. Eine Abstufung wirft eher Probleme auf. Hier droht gegebenenfalls Datenverlust.

Zeitfelder in Tabellen können nicht als Felder mit Bruchteilen einer Sekunde dargestellt werden. Dies geht nur mit einem Timestamp-Feld. Die Tabelle muss entsprechend z.B. so formatiert werden, dass nur Minuten, Sekunden und Zehntelsekunden abgefragt werden: MM:SS,00. Eine Formatierung mit Nachkommastellen ist später in Formularen nur über das formatierte Feld, nicht über das Zeitfeld möglich. Bei der Einstellung der Formatierung muss auf die Landeseinstellung geachtet werden, da sonst statt eines Kommas ein Punkt erforderlich würde.

Für die Erstellung von Feldern, die eine Währung aufnehmen sollen, ist darauf zu achten, dass die Zahlenfelder zwei Nachkommastellen haben. Die Formatierung kann in der Tabellenerstellung der grafischen Benutzeroberfläche in der gewünschten Währung für die Eingabe in die Tabelle vorgenommen werden.

Bei Feldern, die einen Prozentsatz aufnehmen sollen, ist darauf zu achten, dass 1 % bereits als 0,01 gespeichert werden muss. Die Prozentschreibweise beansprucht also schon standardmäßig 2 Nachkommastellen. Sollen Prozentwerte wie 3,45 % abgespeichert werden, so sind also 4 Nachkommastellen bei dem numerischen Wert notwendig.

Einstellung eines Indexes

Manchmal erscheint es sinnvoll, neben dem Primärschlüssel auch andere Felder oder eine Kombination anderer Felder mit einem Index zu versehen. Ein Index dient dazu, Suchergebnisse schneller zu erhalten. Er kann außerdem dazu genutzt werden, Doppeleingaben zu vermeiden.

Jeder Index hat eine fest definierte Sortierreihenfolge. Wird eine Tabelle ohne Sortierung aufgerufen, so richtet sich die Sortierreihenfolge immer nach der Sortierreihenfolge der als Index definierten Felder.

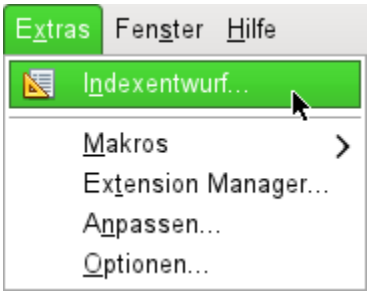


Abbildung 7: Zugriff auf den Indexentwurf

Zuerst muss die Tabelle mit einem rechten Mausklick über das Kontextmenü zum Bearbeiten geöffnet werden. Der Zugriff auf den Indexentwurf erfolgt dann über **Extras** → **Indexentwurf…** .

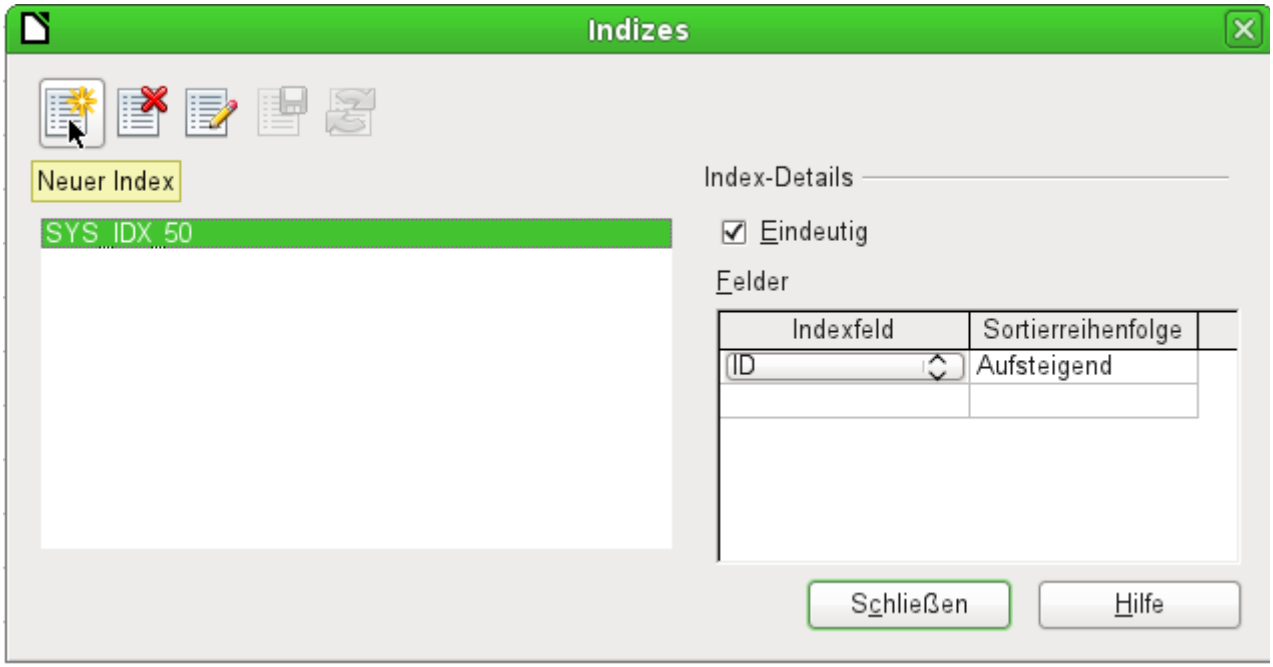


Abbildung 8: Erstellen eines neuen Indexes

Über «Neuer Index» wird ein Index neben dem des Primärschlüssels erstellt.

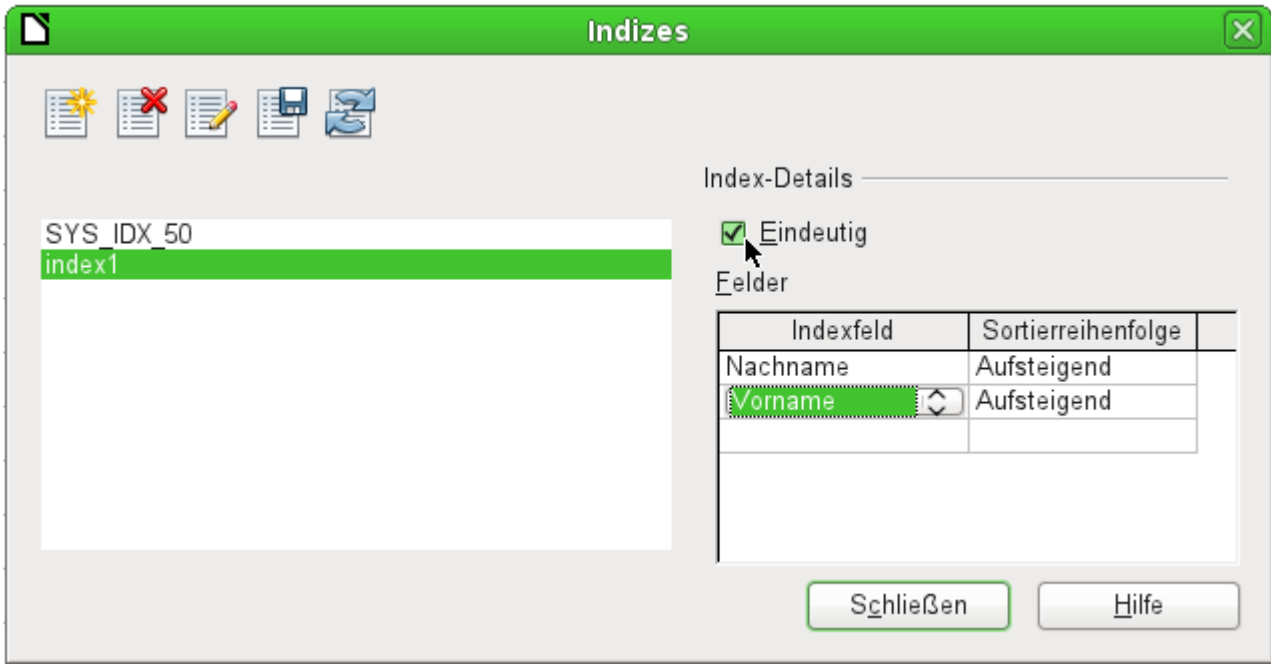


Abbildung 9: Der Index wird als «Eindeutig» definiert.

Dem neuen Index wird automatisch die Bezeichnung «index1» zugewiesen. Im Indexfeld wird ausgewählt, welches Feld bzw. welche Felder über den Index verwaltet werden sollen. Dabei wird gleichzeitig eine Sortierung eingestellt.

Ein Index kann prinzipiell auch über Tabellenfelder erstellt werden, die keine eindeutigen Werte haben. Im obigen Bild ist aber das Index-Detail «Eindeutig» gewählt, so dass in das Feld "Nachname" zusammen mit dem Feld "Vorname" nur Werte eingegeben werden können, die dort in der Kombination noch nicht stehen. So ist z.B. Robert Müller und Robert Maier möglich, ebenso Robert Müller und Eva Müller.

Wird ein Index über ein Feld erstellt, so wird die Eindeutigkeit auf ein Feld bezogen. Ein solcher Index ist in der Regel der Primärschlüssel. In diesem Feld darf jeder Wert nur einmal vorkommen. Beim Primärschlüssel darf allerdings zusätzlich das Feld auf keinen Fall NULL sein.

Eine Sonderstellung für einen eindeutigen Index nimmt in einem Feld das Fehlen eines Eintrages, also NULL, ein. Da NULL alle beliebigen Werte annehmen könnte ist es ohne weiteres erlaubt, bei einem Index über zwei Felder in einem Feld mehrmals hintereinander die gleiche Eingabe zu tätigen, solange in dem anderen Feld keine weitere Angabe gemacht wird.

Hinweis	**NULL** ist für Datenbanken die Bezeichnung für eine leere Zelle, die nichts enthält. Mit einem Feld, das NULL ist kann also nicht gerechnet werden. Im Gegensatz dazu gehen Tabellenkalkulationen bei leeren Feldern automatisch davon aus, dass der Inhalt «0» ist.

Beispiel: In einer Mediendatenbank wird für die Ausleihe die Mediennummer und das Ausleihdatum eingegeben. Wird das Medium zurückgegeben, so wird dies durch ein Rückgabedatum vermerkt. Nun könnte ein Index über die Felder "Mediennummer" und "Rückgabedatum" doch leicht verhindern, dass das gleiche Medium mehrmals ausgeliehen wird, ohne dass die Rückgabe vermerkt wurde. Dies funktioniert aber leider nicht, da das Rückgabedatum ja noch nicht mit einem Wert versehen ist. Der Index verhindert stattdessen, dass ein Medium zweimal mit dem gleichen Datum zurückgegeben wird – sonst nichts.

Mängel der grafischen Tabellenerstellung

Die Reihenfolge der Tabellenfelder kann im Anschluss an den Abspeichervorgang nicht mehr geändert werden. Für eine Darstellung in anderer Reihenfolge ist dann eine Abfrage notwendig. Dies gilt, obwohl die grafische Benutzeroberfläche etwas anderes vortäuscht. Hier kann bei der Tabellenerstellung und bei der Tabellenbearbeitung ein Kontextmenü aufgerufen werden, das z.B. anbietet, Felder auszuschneiden und an anderer Stelle einzufügen. Damit sind dann aber nur die Feldbezeichnungen und die Feldtypen gemeint, nicht aber die Inhalte der Tabelle. Die tauchen nach so einer Änderung mit anderer Feldbezeichnung und eventuell auch anderem Feldtyp wieder auf.[1]

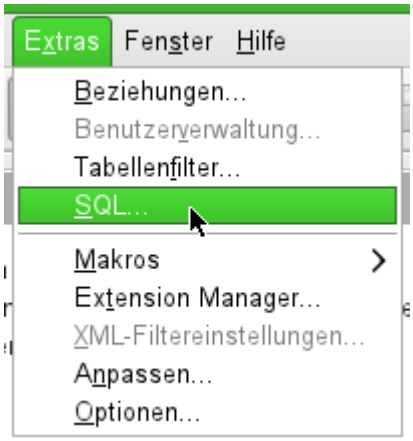
Nur über direkte SQL-Eingabe kann ein neues Feld an eine bestimmte Position innerhalb der Tabelle rutschen. Bereits erstellte Felder sind aber auch hier nicht beweglich.

Eigenschaften der Tabellen müssen sofort festgelegt werden. Welche Felder sollen nicht NULL sein, welche einen Standardwert (Default) erhalten. Diese Eigenschaft kann hinterher innerhalb der grafischen Benutzeroberfläche nicht geändert werden.

Die dort abgelegten Default-Werte haben nichts mit den in der Datenbank selbst liegenden Default-Werten zu tun. So kann dort z.B. bei einem Datum nicht als Standard das aktuelle Datum vorgegeben werden. Dies ist der direkten Eingabe über SQL vorbehalten.

Direkte Eingabe von SQL-Befehlen

Die direkte Eingabe von SQL-Befehlen ist über das Menü **Extras** → **SQL** erreichbar.



Hier werden Befehle im oberen Fensterbereich eingegeben; im unteren Bereich wird der Erfolg oder gegebenenfalls die Gründe für den fehlenden Erfolg (auf Englisch) mitgeteilt. Abfragen können hier nicht dargestellt werden. Für sie ist bei den Abfragen extra die Möglichkeit gegeben, die Abfrage im SQL-Modus zu bearbeiten.

1 *Bug 51605*

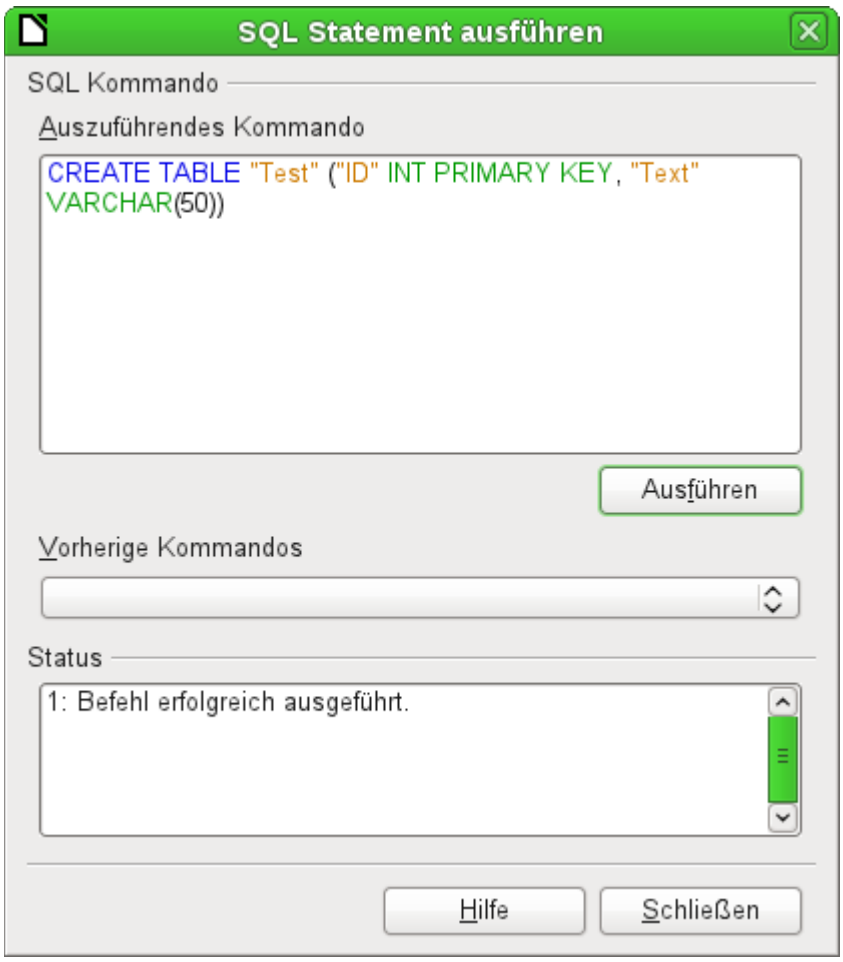


Abbildung 10: Fenster für direkte Eingabe von SQL-Befehlen

Eine Übersicht der für die eingebaute HSQLDB möglichen Eingaben ist unter *http://www.hsqldb.org/doc/1.8/guide/ch09.html* zu finden. Die dortigen Inhalte werden in den folgenden Abschnitten erklärt. Einige Befehle machen nur Sinn, wenn es sich dabei um eine externe HSQLDB handelt (Benutzerangaben usw.) Sie werden gegebenenfalls im Abschnitt «Datenbankverbindung zu einer externen HSQLDB» aufgeführt.

Hinweis	LibreOffice liegt die Version 1.8.0 der HSQLDB zugrunde. Die aktuell erhältliche Serverversion hat die Version 2.2. Die Funktionen der neuen Version sind umfangreicher. Sie sind direkt über *http://hsqldb.org/web/hsqlDocsFrame.html* zu erreichen. Die Beschreibung der Version 1.8 erfolgt jetzt unter *http://www.hsqldb.org/doc/1.8/guide/* . Außerdem ist sie in Installationspaketen zur HSQLDB enthalten, die von *http://sourceforge.net/projects/hsqldb/files/hsqldb/* heruntergeladen werden können.

Tabellenerstellung

Ein einfacher Befehl um eine gebrauchstüchtige Tabelle zu erstellen, ist z. B.

```
CREATE TABLE "Test" ("ID" INT PRIMARY KEY, "Text" VARCHAR(50));
```

`CREATE TABLE "Test"`: Erschaffe eine Tabelle mit dem Namen "Test".
`( )`: mit den hierin enthaltenen Feldnamen, Feldtypen und Zusätzen.
`"ID" INT PRIMARY KEY, "Text" VARCHAR(50)`: Feldname "ID" mit dem Zahlentyp Integer als Primärschlüssel, Feldname "Text" mit dem Texttyp variable Textlänge und der Textbegrenzung auf 50 Zeichen.

```
CREATE [MEMORY | CACHED | [GLOBAL] TEMPORARY | TEMP | TEXT] TABLE
"Tabellenname" ( <Felddefinition> [, ...] [,
<Bedingungsdefinition>...] ) [ON COMMIT {DELETE | PRESERVE} ROWS];
```

[MEMORY | CACHED | [GLOBAL] TEMPORARY | TEMP | TEXT]:

Die Standardeinstellung ist hier **MEMORY**. Die HSQLDB erstellt also grundsätzliche alle Tabellen im Arbeitsspeicher. Dies gilt auch für die Tabellen, die über LibreOffice Base in der internen Datenbank geschrieben werden. Eine andere Möglichkeit wäre, die Tabellen auf die Festplatte zu schreiben und nur über den Arbeitsspeicher den Zugriff auf die Festplatte puffern zu lassen (**CACHED**). Tabellen im Format **TEXT** sind in der rein auf **MEMORY** ausgelegten internen Datenbank nicht beschreibbar, auf **TEMPORARY** bzw. **TEMP** kann Base nicht zugreifen. Die SQL-Befehle werden hier wohl abgesetzt, die Tabellen aber nicht in der grafischen Benutzeroberfläche angezeigt (und damit auch nicht über die grafische Benutzeroberfläche löschbar) und die Eingaben (über SQL) auch nicht über die grafische Benutzeroberfläche des Abfragemoduls anzeigbar, es sei denn die automatische Löschung des Inhaltes nach dem endgültigen Abspeichern ist ausgeschaltet. Eine Abfrage ergibt hier eine Tabelle ohne Inhalt.

Tabellen, die mit SQL direkt gegründet wurden, werden nicht direkt angezeigt. Hier muss entweder über **Ansicht → Tabellen aktualisieren** eine Auffrischung erfolgen oder die Datenbank einfach geschlossen und erneut geöffnet werden.

<Felddefinition>:

```
"Feldname" Datentyp [(Zeichenanzahl[,Nachkommastellen])] [{DEFAULT
"Standardwert" | GENERATED BY DEFAULT AS IDENTITY (START WITH <n>[,
INCREMENT BY <m>])}] | [[NOT] NULL] [IDENTITY] [PRIMARY KEY]
```

Erlaubte Standardwerte innerhalb der Felddefinition:

Für Textfelder kann ein Text in einfachen Anführungsstrichen oder **NULL** eingegeben werden. Die einzige SQL-Funktion, die erlaubt ist, ist **CURRENT_USER**. Dies ergibt allerdings nur dann einen Sinn, wenn die HSQLDB als externe Serverdatenbank mit mehreren Nutzern betrieben wird.

Für Datums- und Zeitfelder kann ein Datum, eine Zeit oder eine Kombination aus Datum und Zeit in einfachen Anführungsstrichen oder **NULL** eingegeben werden. Dabei ist zu beachten, dass das Datum amerikanischen Konventionen entspricht (yyyy-mm-dd), die Zeitangabe das Format hh:mm:ss hat und der Datums_Zeit_Wert das Format yyyy-mm-dd hh:mm:ss.

SQL-Funktionen, die erlaubt sind:

für das aktuelle Datum	-	**CURRENT_DATE, TODAY**
für die aktuelle Zeit	-	**CURRENT_TIME, NOW**
für den aktuellen Datums-Zeit-Wert	-	**CURRENT_TIMESTAMP, NOW**.

Für boolsche Felder (Ja/Nein) können die Ausdrücke **FALSE, TRUE, NULL** gesetzt werden. Diese sind ohne einfache Anführungszeichen einzugeben.

Für numerische Felder ist jede in dem Bereich gültige Zahl sowie NULL möglich. Auch hier sind, zumindest bei **NULL**, keine einfachen Anführungszeichen einzugeben. Bei der Eingabe von Nachkommazahlen ist darauf zu achten, dass die Dezimalstellen durch einen Punkt und nicht durch ein Komma getrennt werden.

Für Binärfelder (Bilder etc.) ist jeder gültige Hexadezimalstring in einfachen Anführungsstrichen sowie **NULL** möglich. Beispiel für einen Hexadezimalstring: '0004ff' bedeutet 3 Bytes, zuerst 0, als zweites 4 und zum Schluss 255 (0xff). Da Binärfelder in der Praxis nur für Bilder eingesetzt werden müsste also der Binärcode des Bildes bekannt sein, das den Defaultwert bilden soll.

NOT NULL → der Feldwert kann nicht NULL sein. Diese Bedingung kann lediglich in der Felddefinition mit angegeben werden.

Hinweis	Hexadezimalsystem: Zahlen werden in einem Stellenwertsystem von 16 dargestellt. Die Ziffern 0 bis 9 und die Buchstaben a bis f ergeben pro Spalte 16 Ziffern im Mischsystem. Bei zwei Feldern kommen dann 16*16=256 mögliche Werte dabei zustande. Das entspricht schließlich 1 Byte (2^8).

<Bedingungsdefinition>:

```
[CONSTRAINT "Name"]
UNIQUE ( "Feldname 1" [,"Feldname 2"...] ) |
PRIMARY KEY ( "Feldname 1" [,"Feldname 2"...] ) |
FOREIGN KEY ( "Feldname 1" [,"Feldname 2"...] )
REFERENCES "anderer Tabellenname" ( "Feldname_1" [,"Feldname 2"...])
[ON {DELETE | UPDATE}
{CASCADE | SET DEFAULT | SET NULL}] |
CHECK(<Suchbedingung>)
```

Bedingungsdefinitionen (Constraints) definieren Bedingungen, die beim Einfügen der Daten erfüllt sein müssen. Die Constraints können mit einem Namen versehen werden.
`UNIQUE ("Feldname")` → der Feldwert muss innerhalb des Feldes einzigartig sein
`PRIMARY KEY ("Feldname")` → der Feldwert muss einzigartig sein und kann nicht **`NULL`** sein (Primärschlüssel)
`FOREIGN KEY ("Feldname") REFERENCES <"anderer Tabellenname"> ("Feldname")` → Die aufgeführten Felder dieser Tabelle sind mit den Feldern einer anderen Tabelle sind verknüpft. Der Feldwert muss auf «Referentielle Integrität» geprüft werden (Fremdschlüssel), d.h. Es muss ein entsprechender Primärschlüssel in der anderen Tabelle existieren, wenn hier ein Wert eingetragen wird.
`[ON {DELETE | UPDATE} {CASCADE | SET DEFAULT | SET NULL}]` → Wenn ein Fremdschlüssel besteht, so ist hier zu klären, wie damit verfahren werden soll, wenn z.B. der fremde Datensatz gelöscht wird. Z.B. macht es keinen Sinn, in einer Ausleihtabelle einer Bibliothek eine Nutzernummer weiter zu führen, wenn der Nutzer selbst gar nicht mehr existiert. Die entsprechenden Datensätze müssten behandelt werden, so dass die Beziehung zwischen den Tabellen stimmig bleibt. In der Regel würde der entsprechende Datensatz einfach gelöscht. Dies geschieht mit **`ON DELETE CASCADE`**.
`CHECK(<Suchbedingung>)` → Wird wie eine **`WHERE`**-Bedingung formuliert, bezieht sich aber nur auf den aktuellen Datensatz.

Mit Constrains wird vor allem gearbeitet, wenn die Beziehung zwischen Tabellen oder der Index für bestimmte Felder festgelegt werden soll.

[ON COMMIT {DELETE | PRESERVE} ROWS]:

Der Inhalt von Tabellen des Typs **`TEMPORARY`** oder **`TEMP`** wird nach Beendigung der Arbeit mit dem Datensatz standardmäßig gelöscht (**`ON COMMIT DELETE ROWS`**). Hier kann also nur ein flüchtiger Datensatz erstellt werden, der Informationen für andere Aktionen, die gleichzeitig laufen, vorhält.

Sollen diese Tabellentypen Daten für eine ganze Sitzung (Aufruf einer Datenbank und Schließen einer Datenbank) zur Verfügung stehen, so kann hier **`ON COMMIT PRESERVE ROWS`** gewählt werden.

Tabellenänderung

Manchmal wünscht sich der User, dass ein zusätzliches Feld an einer bestimmten Stelle in die Tabelle eingebaut wird. Angenommen es gibt die Tabelle "Adresse" mit den Feldern "ID", "Name", "Strasse" usw. Jetzt fällt dem Nutzer auf, dass vielleicht eine Unterscheidung in Name und Vorname sinnvoll wäre:

```
ALTER TABLE "Adresse" ADD "Vorname" VARCHAR(25) BEFORE "Name";
```

ALTER TABLE "Adresse": Ändere die Tabelle mit dem Namen "Adresse".
ADD "Vorname" VARCHAR(25): füge das Feld "Vorname" mit einer Länge von 25 Zeichen hinzu.
BEFORE "Name": und zwar vor dem Feld "Name".

Die Möglichkeit, die Position nach dem Erstellen einer Tabelle für zusätzliche Felder zu bestimmen, bietet die GUI nicht.

```
ALTER TABLE "Tabellenname" ADD [COLUMN] <Felddefinition> [BEFORE
"bereits_existierender_Feldname"];
```

Die zusätzliche Bezeichnung **COLUMN** ist dabei nicht erforderlich, da keine Alternativen zur Auswahl stehen.

```
ALTER TABLE "Tabellenname" DROP [COLUMN] "Feldname";
```

Das Feld "Feldname" wird aus der Tabelle "Tabellenname" gelöscht. Dies wird allerdings unterbunden, wenn das Feld in einem View oder als Fremdschlüssel in einer anderen Tabelle Bedeutung hat.

```
ALTER TABLE "Tabellenname" ALTER COLUMN "Feldname" RENAME TO
"neuer_Feldname"
```

Ändert den Namen eines Feldes

```
ALTER TABLE "Tabellenname" ALTER COLUMN "Feldname" SET DEFAULT <Stan-
dardwert>};
```

Fügt dem Feld einen bestimmten Standardwert hinzu. **NULL** entfernt einen bestehenden Standardwert.

```
ALTER TABLE "Tabellenname" ALTER COLUMN "Feldname" SET [NOT] NULL
```

Setzt oder entfernt eine **NOT NULL** Bedingung für ein Feld.

```
ALTER TABLE "Tabellenname" ALTER COLUMN <Felddefinition>;
```

Die Felddefinition entspricht der aus der *Tabellenerstellung* mit den folgenden Einschränkungen:

- Das Feld muss bereits ein Primärschlüsselfeld sein um die Eigenschaft **IDENTITY** zu akzeptieren. **IDENTITY** bedeutet, dass das Feld die Eigenschaft «Autowert» erhält. Dies ist nur bei **INTEGER** oder **BIGINT** möglich. Zu den Feldtypenbezeichnungen siehe den Anhang dieses Handbuches.
- Wenn das Feld bereits die Eigenschaft **IDENTITY** hat und sie wird nicht in die Felddefinition erneut aufgenommen, so wird die bereits existierende Eigenschaft **IDENTITY** entfernt.
- Der Standardwert wird der der neuen Felddefinition sein. Wenn die Definition des Standardwertes leer gelassen wird, so wird ein bereits bestehender entfernt.
- Die Eigenschaft **NOT NULL** wird in die neue Definition übernommen, wenn nicht anders definiert. Dies entspricht dem Umgang mit dem Standardwert.
- Abhängig von der Art der Änderung muss eventuell die Tabelle leer sein, damit die Änderung durchgeführt werden kann. Auf jeden Fall wird die Änderung dann funktionieren, wenn die Änderung grundsätzlich möglich ist (z.B. Änderung von **NOT NULL** auf **NULL**) und die existierenden Werte alle umgewandelt werden können (z.B. von **TINYINT** zu **INTEGER**).

```
ALTER TABLE "Tabelle" ADD PRIMARY KEY ("Feldname1", "Feldname2" ...);
```

Dieser Befehl erstellt im Nachhinein einen Primärschlüssel, auch über mehrere Felder.

```
ALTER TABLE "Tabellenname"
ALTER COLUMN "Feldname" RESTART WITH <neuer_Feldwert>
```

Dieser Befehl wird ausschließlich für ein **IDENTITY** Feld genutzt. Damit wird der nächste Wert eines Feldes mit Autowert-Funktion festgelegt. Dies kann z.B. genutzt werden, wenn eine Daten-

bank erst einmal mit Testdaten versehen wurde, bevor sie mit den eigentlichen Daten bestückt wurde. Dann wird der Inhalt der Tabellen gelöscht und der neue Feldwert z.B. als 1 festgelegt.

```
ALTER TABLE "Tabellenname"
ADD [CONSTRAINT "Bedingungsname"] CHECK (<Suchbedingung>);
```

Dies fügt eine mit **CHECK** eingeleitete Suchbedingung hinzu. Solch eine Bedingung wird nicht auf bereits bestehende Datensätze angewandt, sondern bei allen zukünftigen Änderungen und neu erstellten Datensätzen berücksichtigt. Wird kein Bedingungsname definiert, so wird automatisch eine Bezeichnung zugewiesen. Beispiel:

```
ALTER TABLE "Ausleihe" ADD CHECK
(IFNULL("Rueckdatum","Leihdatum")>="Leihdatum")
```

Die Tabelle **"Ausleihe"** soll in Bezug auf Fehleingaben abgesichert werden. Es soll vermieden werden, dass ein Rückgabedatum angegeben wird, das vor dem Ausleihdatum liegt. Taucht jetzt bei der Eingabe des Rückgabedatums dieser Fehler auf, so erscheint die Fehlermeldung **Check constraint violation …**

```
ALTER TABLE "Tabellenname"
ADD [CONSTRAINT "Bedingungsname"] UNIQUE ("Feldname1",
"Feldname2" ...);
```

Hier wird hinzugefügt, dass die benannten Felder nur jeweils verschiedene Werte beinhalten dürfen. Werden mehrere Felder benannt, so gilt dies für die Kombination von Feldern. **NULL** wird hierbei nicht berücksichtigt. Ein Feld kann also ohne weiteres mehrmals die gleichen Werte haben, wenn das andere Feld bei den entsprechenden Datensätzen **NULL** ist.

Der Befehl wird nicht funktionieren, wenn bereits eine **UNIQUE** – Bedingung für die gleiche Felderkombination existiert.

```
ALTER TABLE "Tabellenname"
ADD [CONSTRAINT "Bedingungsname"] PRIMARY KEY ("Feldname1", "Feld-
name2" ...);
```

Fügt einen Primärschlüssel, gegebenenfalls mit einer Bedingungsdefinition, einer Tabelle hinzu. Die Syntax der Bedingungsdefinition entsprecht der der Erstellung bei einer Tabelle.

```
ALTER TABLE "Tabellenname"
ADD [CONSTRAINT "Bedingungsname"] FOREIGN KEY ("Feldname1", "Feld-
name2" ...)
REFERENCES "Tabellenname_der_anderen_Tabelle"
("Feldname1_andere_Tabelle", "Feldname2_andere_Tabelle" ...)
[ON {DELETE | UPDATE} {CASCADE | SET DEFAULT | SET NULL}];
```

Hiermit wird eine Fremdschlüsselbedingung (**FOREIGN KEY**) zur Tabelle hinzugefügt. Die Syntax ist die gleiche wie bei der Erstellung einer Tabelle.

Das Verfahren wird mit einer Fehlermeldung beendet, wenn nicht für jeden Wert in der Tabelle ein entsprechender Wert aus der Tabelle mit dem entsprechenden Schlüsselfeld vorhanden ist.

Beispiel: Die Tabellen "Name" und "Adresse" sollen miteinander verbunden werden. In der Tabelle "Name" gibt es ein Feld mit der Bezeichnung "Adresse_ID". Dies soll mit seinen Werte mit dem Feld "ID" der Tabelle "Adresse" verbunden werden. Steht in "Adresse_ID" bereits der Wert 1, in dem "ID" der Tabelle "Adresse" aber nicht, so kann die Verbindung nicht funktionieren. Ebenfalls unmöglich ist es, wenn der Feldtyp in beiden Feldern nicht übereinstimmt.

```
ALTER TABLE "Tabellenname" DROP CONSTRAINT "Bedingungsname";
```

Der Befehl entfernt eine mit Namen versehene Bedingung (**UNIQUE**, **CHECK**, **FOREIGN KEY**) aus einer Tabelle.

```
ALTER TABLE "Tabellenname" RENAME TO "neuer_Tabellenname";
```

Mit diesem Befehl schließlich wird einfach nur der Name einer Tabelle geändert.

Mit der Wahl des Datentyp CHAR wird eine fixe Breite festgelegt. Gegebenenfalls wird Text mit Leerzeichen aufgefüllt. Bei einer Umstellung auf VARCHAR bleiben diese Leerzeichen erhalten. Sollen die Leerzeichen entfernt werden, so gelingt dies mittels

```
UPDATE "Tabellenname" SET "Feldname" = RTRIM("Feldname")
```

Hinweis	Bei der Änderung einer Tabelle über SQL wird die Änderung zwar in der Datenbank übernommen, nicht aber unbedingt sofort überall in der GUI sichtbar und verfügbar. Wird die Datenbank geschlossen und wieder geöffnet, so werden die Änderungen auch in der GUI angezeigt.

Tabellen löschen

```
DROP TABLE "Tabellenname" [IF EXISTS] [RESTRICT | CASCADE];
```

Löscht die Tabelle "Tabellenname".

`IF EXISTS` schließt aus, dass eine Fehlermeldung erscheint, falls diese Tabelle nicht existiert.

`RESTRICT` ist die Standardeinstellung und muss nicht definitiv gewählt werden, d.h. ein Löschen wird dann nicht ausgeführt, wenn die Tabelle mit irgendeiner anderen Tabelle durch einen Fremdschlüssel verbunden wurde oder auf die Tabelle mit einem View (Ansicht) Bezug genommen wird. Abfragen sind davon nicht berührt, da die innerhalb der HSQLDB nicht gespeichert sind.

Wird statt **`RESTRICT CASCADE`** gewählt, so werden alle Beziehungen zu der Tabelle "Tabellenname" gelöscht. In den verknüpften Tabellen werden dann alle Fremdschlüsselfelder auf NULL gesetzt. Alle Views, in denen auf die entsprechende Tabelle Bezug genommen wird, werden komplett gelöscht.

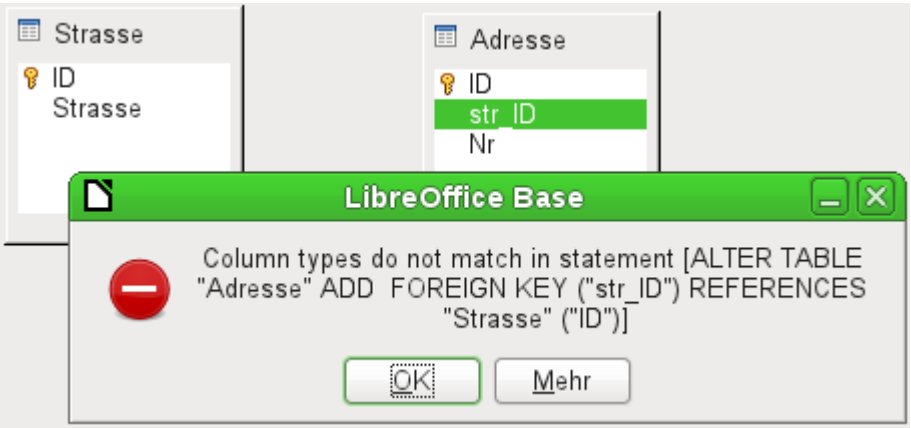
Verknüpfung von Tabellen

Prinzipiell kommt eine Datenbank auch ohne die Verknüpfung von Tabellen aus. Der Nutzer muss dann bei der Eingabe selbst darauf achten, dass die Beziehungen zwischen den Tabellen stimmig bleiben. In der Regel geschieht dies, indem er sich entsprechende Formulare erstellt, die dies bewerkstelligen sollen.

Das Löschen von Datensätzen bei verknüpften Tabellen ist nicht so einfach möglich. Angenommen es würde aus der Tabelle *Strasse* in *Abbildung 6* eine *Strasse* gelöscht, die aber durch die Verknüpfung mit der Tabelle *Adresse* in der Tabelle *Adresse* noch als Fremdschlüssel vertreten ist. Der Verweis in der Tabelle *Adresse* würde ins Leere gehen. Hiergegen sperrt sich die Datenbank, sobald der Relationenentwurf erstellt wurde. Um die *Strasse* löschen zu können, muss die Vorbedingung erfüllt sein, dass sie nicht mehr in *Adresse* benötigt wird.

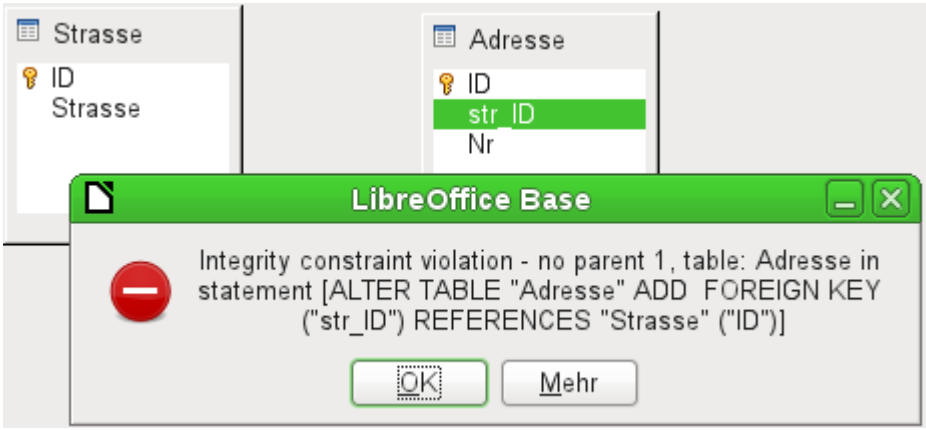
Die grundsätzlichen Verknüpfungen werden über **Extras → Beziehungen** festgelegt. Hierbei wird eine Verbindungslinie von dem Primärschlüssel einer Tabelle zum zu definierenden Sekundärschlüssel gezogen.

Die folgenden Fehlermeldungen können beim Ziehen so einer Verbindung auftreten:



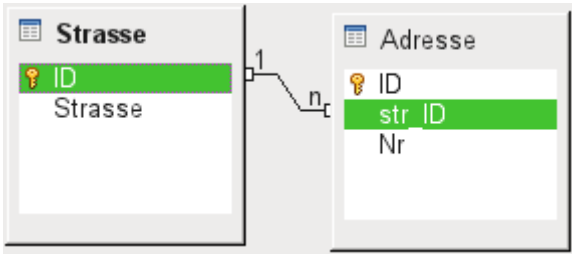
Die Meldung gibt einen englischen Text sowie das zu der Fehlermeldung führende SQL-Kommando wieder. Eine gute Möglichkeit also, auch an dieser Stelle etwas über die Sprache zu erfahren, mit der die Datenbank arbeitet.

«Column types do not match in statement» - die Spaltentypen stimmen in der SQL-Formulierung nicht überein. Da das SQL-Kommando gleich mitgeliefert wird, müssen das die Spalten *Adresse.str_ID* und *Strasse.ID* sein. Zu Testzwecken wurde hier das eine Feld als Integer, das andere als 'Tiny Integer' definiert. So eine Verbindung lässt sich nicht erstellen, da das eine Feld nicht die gleichen Werte annehmen kann wie das andere Feld.

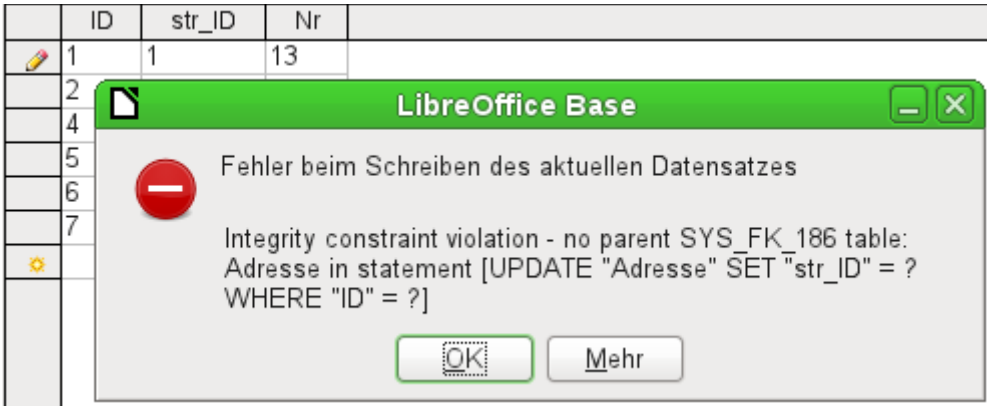


Jetzt stimmen die Spaltentypen überein. Die SQL-Formulierung (statement) ist die gleiche wie beim ersten Zugriff. Aber wieder taucht ein Fehler auf:

«Integrity constraint violation – no parent 1, table: Adresse ...» - die Integrität der Beziehung ist nicht gewährleistet. In dem Feld der Tabelle Adresse, also *Adresse.str_ID*, gibt es eine Ziffer 1, die es im Feld *Strasse.ID* nicht gibt. Parent ist hier die Tabelle *Strasse*, weil deren Primärschlüssel vorhanden sein muss. Dieser Fehler tritt häufig auf, wenn zwei Tabellen miteinander verbunden werden sollen, bei denen in den Feldern der Tabelle mit dem zukünftigen Fremdschlüssel schon Daten eingegeben wurden. Steht in dem Fremdschlüssel-Feld ein Eintrag, der in der Parent-Tabelle (Eltern-Tabelle, also der Tabelle, au der der Primärschlüssel gestellt wird) nicht vorhanden ist, so würde ein ungültiger Eintrag erzeugt.



Ist die Verbindung erfolgreich erzeugt worden und wird später versucht einen entsprechend fehlerhaften Eintrag in die Tabelle einzugeben, so kommt die folgende Fehlermeldung:



Also wiederum die Integritätsverletzung. Base weigert sich, für das Feld *str_ID* nach der Verknüpfung den Wert 1 anzunehmen, weil die Tabelle *Strasse* so einen Wert im Feld *ID* nicht enthält.

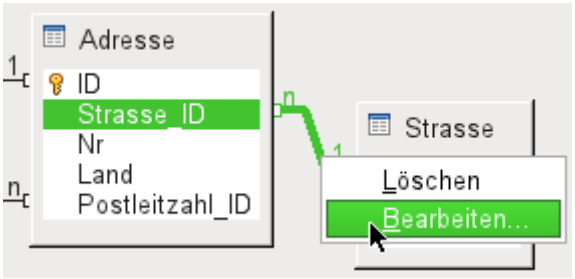
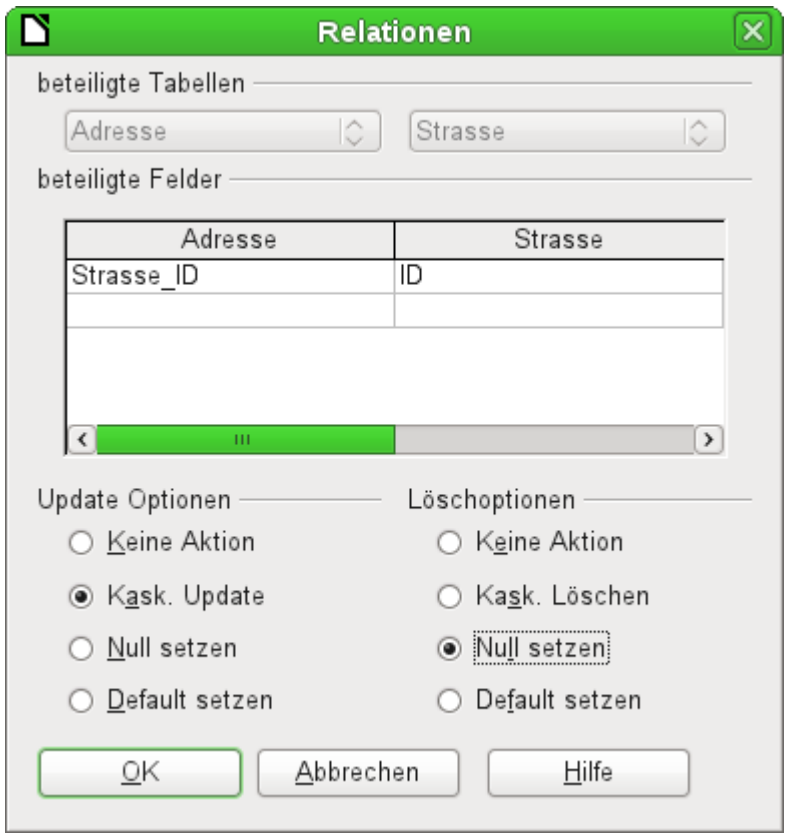


Abbildung 11: Durch Klick mit der rechten Maustaste können Verknüpfungen bearbeitet werden.

Die Eigenschaften der Verknüpfung können so bearbeitet werden, dass beim Löschen von Datensätzen aus der Tabelle *Strasse* gleichzeitig eventuell vorhandene Einträge in der Tabelle *Adresse* auf NULL gesetzt werden.



Die oben abgebildeten Eigenschaften beziehen sich immer auf eine Aktion, die mit einer Änderung des Datensatzes aus der Tabelle zusammenhängt, zu der der betroffene Primärschlüssel gehört. In unserem Fall ist dies die Tabelle *Strasse.* Wird also dort ein **der Primärschlüssel eines Datensatzes "ID" geändert (Update)**, so können die folgenden Aktionen ausgeführt werden:

Keine Aktion:

Eine Änderung des Primärschlüssels *Strasse.ID* kann in diesem Fall nicht vorgenommen werden, da die Relation ansonsten zerstört wird. Da dies die Standardeinstellung der Relation ist gehen Nutzer häufig davon aus, dass eine Änderung des Primärschlüssels unmöglich ist. Die anderen Update-Optionen ermöglichen allerdings so eine Änderung.

Kask. Update:

Bei einer Änderung des Primärschlüsselwertes *Strasse.ID* allerdings wird der Fremdschlüsselwert automatisch auf den neuen Stand gebracht. Die Koppelung wird dadurch nicht beeinträchtigt. Wird z.B. der Wert von 3 auf 4 geändert, so wird bei allen Datensätzen aus *Adresse*, in denen der Fremdschlüssel *Adresse.Strasse_ID* 3 lautete, stattdessen eine 4 eingetragen.

Null setzen:

Alle Datensätze, die sich auf den Primärschlüssel bezogen haben, haben jetzt in dem Fremdschlüsselfeld *Adresse.Strasse_ID* stattdessen keinen Eintrag mehr stehen, sind also NULL.

Default setzen:

Wird der Primärschlüssel *Strasse.ID* geändert, so wird der damit ursprünglich verbundene Wert aus *Adresse.Strasse_ID* auf den dafür vorgesehenen Standardwert gesetzt. Hierfür ist allerdings eine eindeutige Definition eines Standardwertes erforderlich. Dies scheint die grafische Benutzeroberfläche bis zur Version LO 3.5 nicht bereitzustellen. Wird per SQL der Default-Wert durch

```
ALTER TABLE "Adresse" ALTER COLUMN "Strasse_ID" SET DEFAULT 1;
```

gesetzt, so übernimmt die Beziehungsdefinition auch die Zuweisung, dass bei einem Update auf diesen Wert zurückgegriffen werden kann. Wird also der Primärschlüssel aus der Tabelle *Strasse* geändert, so wird in der Tabelle *Adresse* das Fremdschlüsselfeld auf 1 gesetzt. Dies bietet sich z.B. an, wenn auf jeden Fall jeder Datensatz eine Straßenzuweisung erhalten soll, also nicht NULL sein soll. Aber Achtung: Ist 1 nicht belegt, so wird eine Verbindung zu einem nicht vorhandenen Wert erzeugt. Hier scheint die HSQLDB nicht mit letzter Konsequenz durchdacht. Es ist also möglich, die Integrität der Relationen zu zerstören.

Vorsicht 	Ist der Defaultwert im Fremdschlüsselfeld nicht durch einen Primärschlüssel der Ursprungstabelle belegt, so wird eine Verbindung zu einem nicht vorhandenen Wert erzeugt. Es ist also möglich, die referentielle Integrität der Datenbank zu zerstören.

Vor diesem Hintergrund scheint es sinnvoll, diese Einstellung nicht in Betracht zu ziehen.

Wird ein Datensatz aus der Tabelle *Strasse* **gelöscht**, so stehen die folgenden Optionen zur Verfügung:

Keine Aktion:
Es wird nichts unternommen. Ist ein Datensatz aus der Tabelle *Adresse* von der Löschung betroffen, so wird die Löschung abgelehnt. Es existiert schließlich weiterhin ein entsprechender Fremdschlüssel in der Tabelle *Adresse.*

Wie bei den Update-Optionen kann nur mit dieser Option eine Löschung des Datensatzes unterbunden werden. Die weiteren Optionen geben hingegen den Weg vor, den die Datenbank beschreiten soll, falls von dem Datensatz in der Tabelle *Strasse* ein Fremdschlüssel in der Tabelle *Adresse* betroffen ist.

Kaskadiert Löschen:
Wird ein Datensatz aus der Tabelle *Strasse* gelöscht und ist davon ein Datensatz aus der Tabelle *Adresse* betroffen, so wird auch dieser gelöscht.

Das mag in diesem Zusammenhang merkwürdig klingen, ergibt aber bei anderen Tabellenkonstruktionen sehr wohl einen Sinn. Angenommen es existiert eine Tabelle mit CDs und eine Tabelle, die die Titel, die auf diesen CDs enthalten sind, speichert. Wird nun ein Datensatz aus der CD-Tabelle gelöscht, so stehen lauter Titel in der anderen Tabelle, die gar keinen Bezug mehr haben, also gar nicht mehr vorhanden sind. Hier ist dann ein kaskadierendes Löschen sinnvoll. So muss der Nutzer nicht vor dem Entfernen der CD aus der Datenbank erst sämtliche Titel löschen.

Null setzen:
Dieses Verhalten entspricht der gleichlautenden Update-Option.

Default setzen:
Dieses Verhalten entspricht ebenfalls der gleichlautenden Update-Option. Die Bedenken bei dieser Option sind entsprechend die gleichen.

Tipp	Sollen möglichst Fehlermeldungen der Datenbank vermieden werden, da sie dem Datenbanknutzer vielleicht nicht deutbar sind, so sind auf jeden Fall die Einstellungen «*Keine Aktion*» zu vermeiden.

Eingabe von Daten in Tabellen

Datenbanken, bestehend aus einer Tabelle, erfordern in der Regel keine Formulare zur Eingabe, es sei denn sie enthalten ein Feld für Bildmaterial. Sobald allerdings eine Tabelle den Fremd-

schlüssel einer anderen Tabelle enthält, muss der Nutzer entweder auswendig wissen, welche Schlüsselnummern er eintragen muss, oder er muss die andere Tabelle zum Nachschauen gleichzeitig offen halten. Dafür wird dann spätestens ein Formular sinnvoll.

Eingabe über die grafische Benutzeroberfläche der Tabelle

Die Tabelle aus dem Tabellencontainer wird durch einen Doppelklick geöffnet. Wird der Primärschlüssel durch ein automatisch hochzählendes Feld erstellt, so enthält eins der jetzt zu sehenden Felder bereits den Text «AutoWert». Im «AutoWert»-Feld ist keine Eingabe möglich. Dieser Wert kann gegebenenfalls erst nach Abspeicherung des Datensatzes geändert werden.

Abbildung 12: Eingabe in Tabellen - Spalten ausblenden.

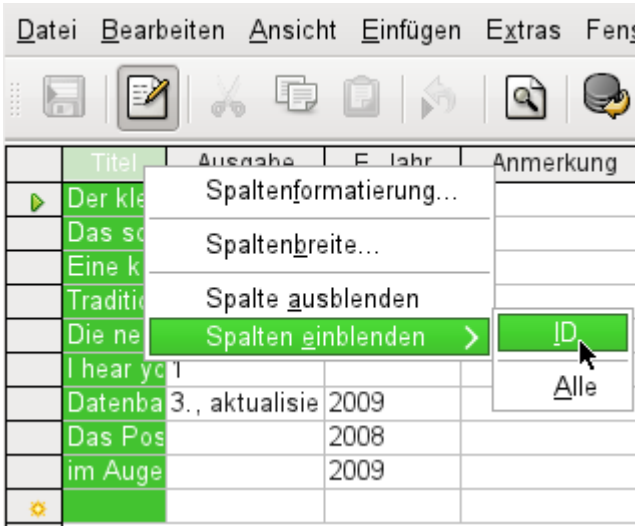


Abbildung 13: Eingabe in Tabellen - Spalten wieder einblenden.

Einzelne Spalten der Tabellenansicht können auch ausgeblendet werden. Wenn z.B. das Primärschlüsselfeld nicht unbedingt sichtbar sein soll, so lässt sich das in der Tabelleneingabe einstellen. Diese Einstellung wird als Einstellung der GUI abgespeichert. Die Spalte existiert in der Tabelle weiter und kann gegebenenfalls auch wieder eingeblendet werden.

Die Eingabe in die Tabellenzellen erfolgt in der Regel von links nach rechts über Tastatur und Tabulator. Natürlich ist auch die Nutzung der Maus möglich.

Nach Erreichen des letzten Feldes eines Datensatzes springt der Cursor automatisch in den nächsten Datensatz. Die vorhergehende Eingabe wurde dabei abgespeichert. Eine zusätzliche Abspeicherung unter **Datei → Speichern** ist nicht nötig und nicht möglich. Die Daten sind bereits in der Datenbank gelandet, bei der HSQLDB also im Arbeitsspeicher. Sie werden (aus Sicht der Datensicherheit leider) erst auf der Festplatte abgespeichert, wenn Base geschlossen wird. Wenn Base also aus irgendwelchen Gründen nicht ordnungsgemäß beendet werden kann, kann dies immer noch zu Datenverlusten führen.

Fehlt eine Eingabe, die vorher im Tabellenentwurf als zwingend notwendig deklariert wurde (**`NOT NULL`**), so wird eine entsprechende Fehlermeldung erzeugt:
`Attempt to insert null into a non-nullable column …`
Die entsprechende Spalte, die Tabelle dazu und der von der GUI abgesetzte SQL-Befehl werden angezeigt.

Die Änderung eines Datensatzes ist entsprechend einfach möglich: Feld aufsuchen, geänderten Wert eingeben und Datenzeile wieder verlassen.

Zum Löschen eines Datensatzes wird der Datensatz auf dem Zeilenkopf markiert, dann die rechte Maustaste gedrückt und **Zeilen löschen** gewählt.

Hilfreich zum Aufsuchen des entsprechenden Datensatzes sind hier die Sortier-, Such- und Filterfunktionen.

Sortieren von Tabellen

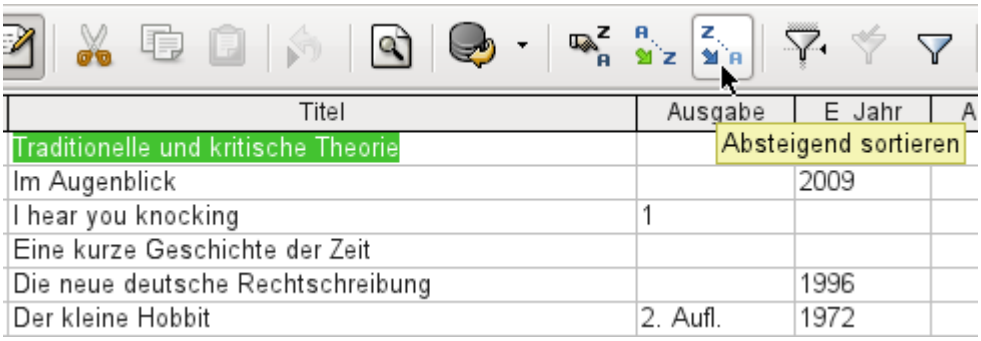


Abbildung 14: Schnelle Sortierung

Die schnelle Sortiervariante verbirgt sich hinter den Buttons **A→Z** bzw. **Z→A**. Ein Feld innerhalb einer Spalte wird aufgesucht, ein Mausklick auf den Button und nach der Spalte wird sortiert. Hier wurde gerade die Spalte "Titel" absteigend sortiert.

Mit der schnellen Variante kann immer nur nach einer Spalte sortiert werden. Für eine Sortierung nach mehreren Spalten ist eine weitere Sortierfunktion vorgesehen:

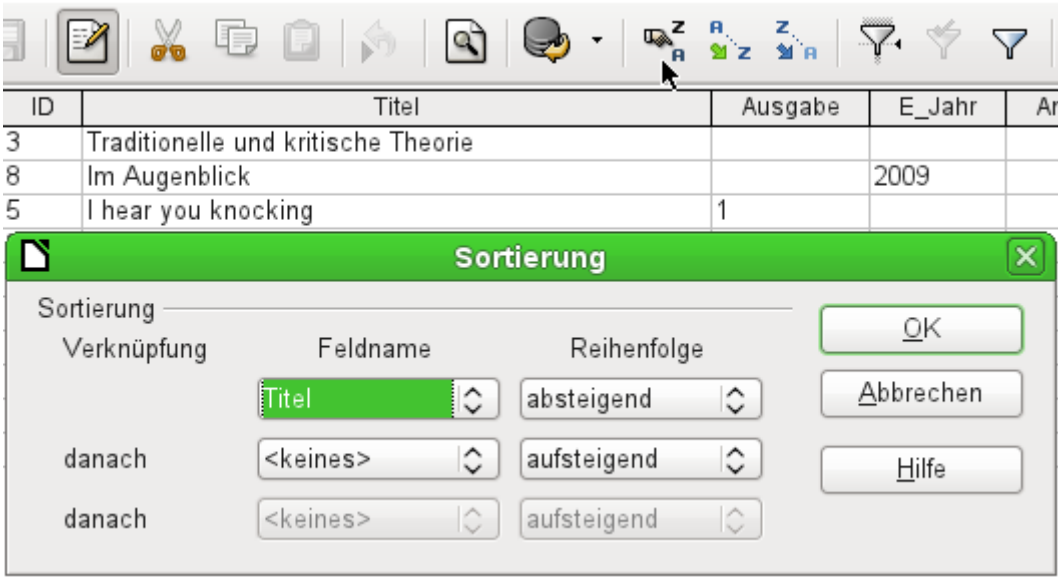
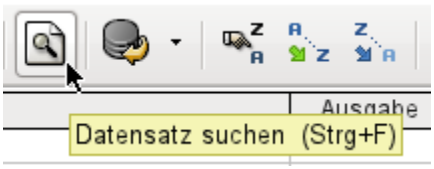


Abbildung 15: Sortierung nach mehreren Spalten

Der Feldname der Spalte sowie die jeweilige Sortierreihenfolge werden ausgesucht. Wurde vorher bereits eine schnelle Sortierung vorgenommen, so ist in der ersten Zeile der entsprechende Feldname und die Sortierreihenfolge bereits eingetragen.

Suchen in Tabellen



Die Funktion zum Suchen von Datensätzen ist recht umfangreich und für durch Suchmaschinen verwöhnte Nutzer nicht gerade die erste Wahl, um einen bestimmten Datensatz zu finden.

Tipp	Bevor die Suche aufgerufen wird sollte auf jeden Fall darauf geachtet werden, dass die zu durchsuchenden Spalten von der Breite her weit genug eingestellt sind, um den gefundenen Datensatz auch anzuzeigen. Die Suchfunktion bleibt nämlich im Vordergrund und lässt keine Korrektur der Einstellungen der Spaltenweite in der darunterliegenden Tabelle zu. Um an die Tabelle zu gelangen muss die Suche also abgebrochen werden.

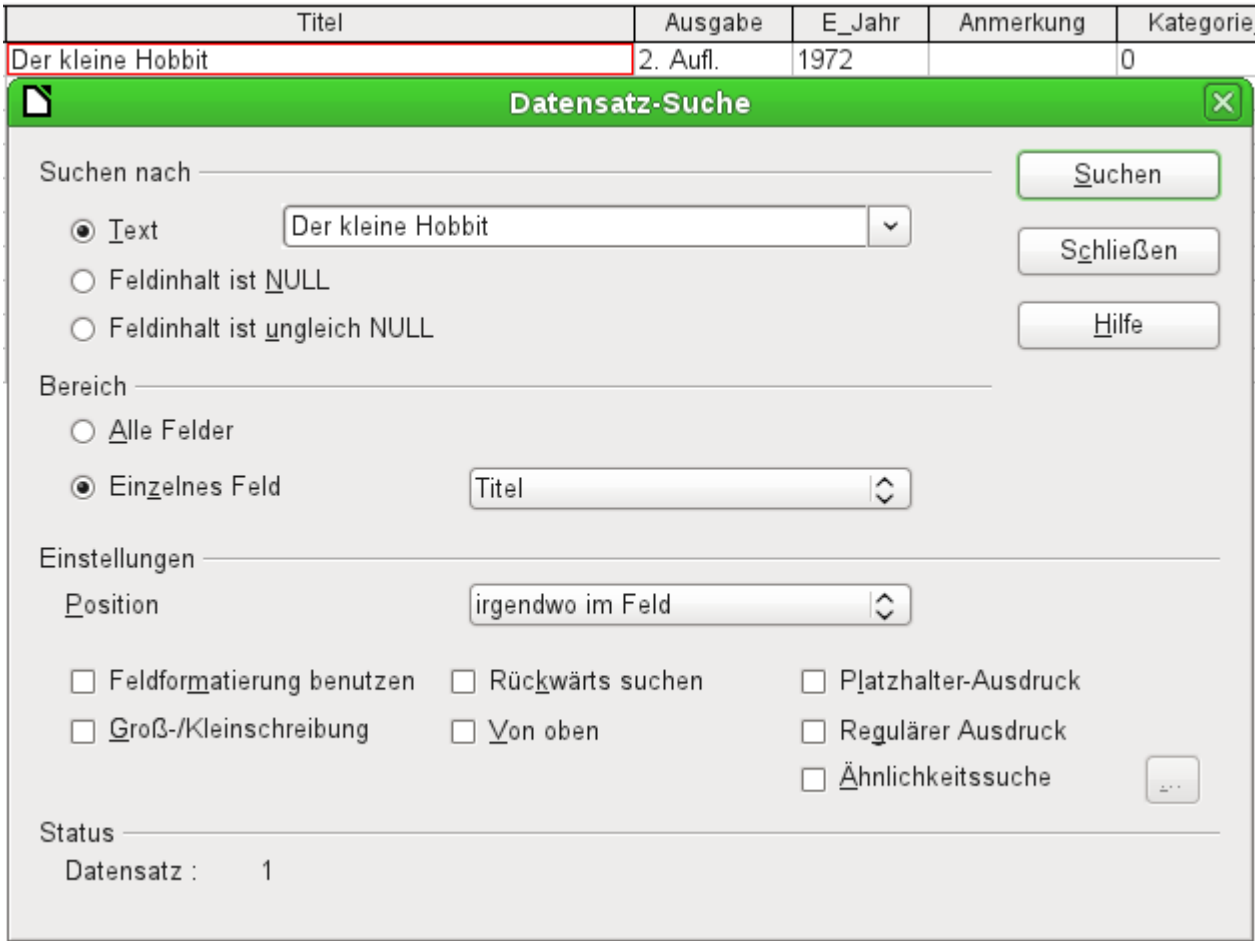


Abbildung 16: Eingabemaske zur Datensatzsuche

Die Suche übernimmt beim Aufruf den Begriff des Feldes, von dem aus sie aufgerufen wurde.

Damit die Suche effektiv verläuft, sollte der Suchbereich möglichst weit eingegrenzt sein. So dürfte es sinnlos sein, den obigen Text aus dem Feld "Titel" in dem Feld "Autor" zu suchen. Stattdessen wird bereits als einzelnes Feld der Feldname "Titel" vorgeschlagen.

Die weiteren Einstellungen der Datensatzsuche können die Suche nach bestimmten Kombinationen vereinfachen. Als Platzhalter-Ausdruck sind hier die in SQL üblichen Platzhalterbezeichnungen («_» für ein variables Zeichen, «%» für eine beliebige Anzahl variabler Zeichen und «\» als Escape-Zeichen, um auch nach den variablen Zeichen selbst suchen zu können).

Reguläre Ausdrücke werden in der Hilfe von LibreOffice unter diesem Begriff ausführlich aufgeführt. Ansonsten gibt sich die Hilfestellung zu diesem Modul recht sparsam. Ein Klick auf den Button **`Hilfe`** landet bei LO 3.3.4 und LO 3.5.2 auf einer leeren Seite.

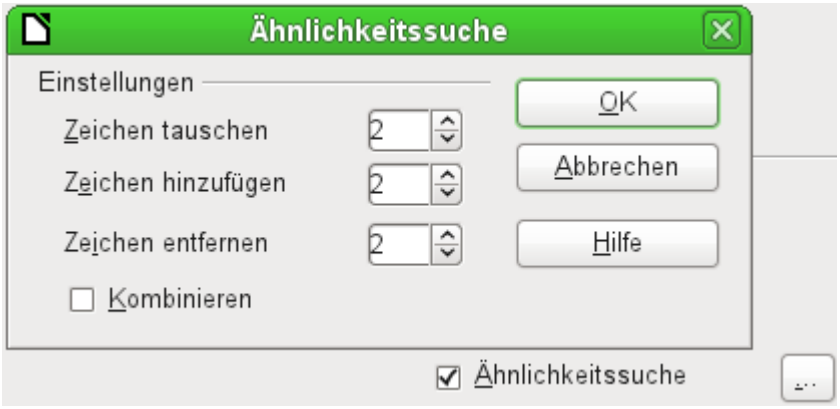


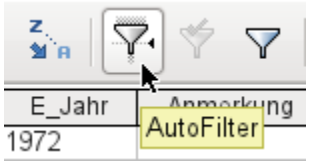
Abbildung 17: Eingrenzung der Ähnlichkeitssuche

Die Ähnlichkeitssuche lässt sich vor allem dann nutzen, wenn es darum geht, Schreibfehler auszuschließen. Je höher die Werte in den Einstellungen gesetzt werden, desto mehr Datensätze werden schließlich in der Trefferliste verzeichnet sein.

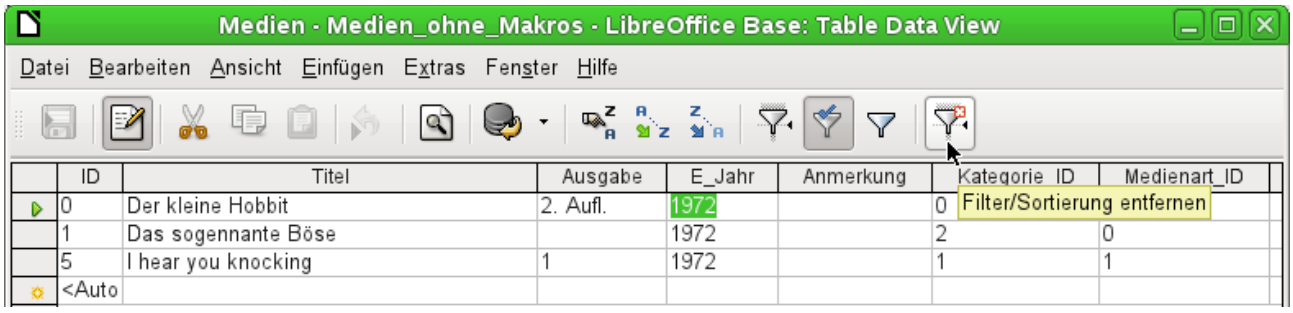
Insgesamt ist dieses Suchmodul eher etwas für Leute, die durch häufige Anwendung genau wissen, an welchen Stellen sie zum Erreichen eines Ziels drehen müssen. Für den Normaluser dürfte die Möglichkeit, Datensätze durch Filter zu finden, schneller zum Ziel führen.

Für Formulare ist in einem der folgenden Kapitel beschrieben, wie mittels SQL, und erweitert mit Makros, eine Stichwortsuche schneller zum Ziel führt.

Filtern von Tabellen



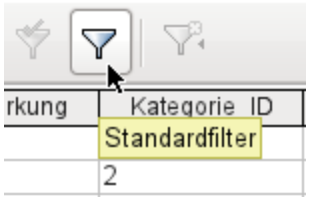
Die schnelle Filterung läuft über den AutoFilter. Der Cursor wird in ein Feld gesetzt, der Filter übernimmt nach einem Klick auf den Button diesen Feldinhalt. Es werden nur noch die Datensätze angezeigt, die dem Inhalt des gewählten Feldes entsprechen. Die folgende Abbildung zeigt die Filterung nach einem Eintrag in der Spalte "E_Jahr".



Der Filter ist aktiv. Dies ist an dem Filtersymbol mit einem grünen Haken zu erkennen. Das Filtersymbol erscheint gedrückt. Wird der Button erneut betätigt, so bleibt der Filter selbst erhalten, es werden aber wieder alle Datensätze angezeigt. So kann gegebenenfalls wieder in den Filterzustand zurückgeschaltet werden.

Durch Betätigung des ganz rechts stehenden Filtersymbols lassen sich alle bereits veranlassten Filterungen und Sortierungen wieder entfernen. Der Filter wird wieder inaktiv und kann nicht mehr mit seinem alten Wert aufgerufen werden.

Tipp	In eine Tabelle, die gefiltert oder durch Suche eingegrenzt wurde, können dennoch ganz normal Daten eingegeben werden. Sie bleiben so lange in der Tabellenansicht stehen, bis die Tabelle durch Betätigung des Buttons Aktualisieren aktualisiert wird.



Mit dem Standardfilter öffnet sich ein Fenster, in dem ähnlich der Sortierung eine Filterung über mehrere Zeilen ausgeführt werden kann. Ist bereits vorher ein AutoFilter eingestellt, so zeigt die erste Zeile des Standardfilters bereits diesen vorgefilterten Wert an.

Abbildung 18: Umfangreiche Datenfilterung im Standardfilter

Der Standardfilter bringt viele Funktionen einer SQL-Datenfilterung mit. Die folgenden SQL-Bedingungen stehen zur Verfügung:

Bedingung GUI	***Beschreibung***
=	Vollständige Gleichheit, entspricht dem Begriff wie, wenn keine zusätzlichen Platzhalterbezeichnungen verwendet werden.
<>	Ungleich
<	Kleiner als
<=	Kleiner als und gleich
>	Größer als
>=	Größer als und gleich
wie	Für Text, in Hochkommata geschrieben (**' '**); «_» für ein variables Zeichen, «%» für eine beliebige Anzahl variabler Zeichen. In SQL entspricht **wie** dem Begriff **LIKE**
nicht wie	Umkehrung von **wie**, in SQL **NOT LIKE**
leer	Kein Inhalt auch nicht eine Leertaste. In SQL entspricht dies dem Begriff **NULL**
nicht leer	Umkehrung von leer, in SQL **NOT NULL**

Bevor die Verknüpfung eines Filterkriteriums mit dem nächsten Filterkriterium erfolgen kann, muss in der Folgezeile zumindest schon einmal ein Feldname ausgewählt worden sein. In der obigen Abbildung steht dort statt eines Feldnamens «-keiner-», so dass die Verknüpfung inaktiv ist. Als Verknüpfung stehen hier **UND** und **ODER** zur Verfügung.

Als Feldname kann hier sowohl ein neuer Feldname als auch ein bereits ausgewählter Feldname erscheinen.

Selbst bei großen Datenbeständen dürfte sich bei geschickter Filterung die Anzahl der angezeigten Datensätze mit diesen 3 Bedingungsmöglichkeiten doch auf einen übersichtlichen Bestand eingrenzen lassen.

Auch für die Filterung werden für Formulare in einem der folgenden Kapitel einige weitere Möglichkeiten vorgestellt, die die GUI so nicht zur Verfügung stellt.

Eingabemöglichkeiten über SQL direkt

Die Eingabe direkt über SQL ist vor allem dann sinnvoll, wenn mehrere Datensätze mit einem Befehl eingefügt, geändert oder gelöscht werden sollen.

Neue Datensätze einfügen

```
INSERT INTO "Tabellenname" [( "Feldname" [,...] )]
{ VALUES("Feldwert" [,...]) | <Select-Formulierung>};
```

Wird kein "Feldname" benannt, so müssen die Felder komplett und in der richtigen Reihenfolge der Tabelle als Werte übergeben werden. Dazu zählt auch das gegebenenfalls automatisch hochzählende Primärschlüsselfeld. Die Werte, die übergeben werden, können auch das Ergebnis einer Abfrage (<Select-Formulierung>) sein. Genauere Erläuterungen hierzu weiter unten.

```
INSERT INTO "Tabellenname" ("Feldname") VALUES ('Test');
CALL IDENTITY();
```

In die Tabelle wird in der Spalte "Name" der Wert 'Test' eingegeben. Das automatisch hochzählende Primärschlüsselfeld "ID" wird nicht angerührt. Der entsprechende Wert für die "ID" wird mit CALL IDENTITY() anschließend ausgelesen. Dies ist bei der Verwendung von Makros wichtig, damit entsprechend mit dem Wert dieses Schlüsselfeldes weiter gearbeitet werden kann.

```
INSERT INTO "Tabellenname" ("Feldname") SELECT "anderer_Feldname" FROM
"Name_anderer_Tabelle";
```

In die erste Tabelle werden jetzt so viele neue Datensätze in "Feldname" eingefügt, wie in der Spalte "anderer_Feldname" der zweiten Tabelle enthalten sind. Die SELECT-Formulierung kann hier natürlich einschränkend wirken.

Bestehende Datensätze ändern

```
UPDATE "Tabellenname" SET "Feldname" = <Expression> [, ...] [WHERE
<Expression>];
```

Vor allem bei der Änderung vieler Datensätze bietet es sich an, doch einmal die SQL-Befehlseingabe aufzusuchen. Angenommen alle Schüler einer Klasse sollen zum neuen Schuljahr um eine Jahrgangsstufe heraufgesetzt werden:

```
UPDATE "Tabellenname" SET "Jahrgang" = "Jahrgang"+1
```

Schneller geht es nicht: Alle Datensätze werden mit einem Befehl geändert. Natürlich muss jetzt noch nachgesehen werden, welche Schüler denn davon nicht betroffen sein sollen. Einfacher wäre es, vorher in einem Ja/Nein-Feld die Wiederholungen anzukreuzen und dann nur diejenigen eine Stufe heraufzusetzen, die nicht angekreuzt wurden:

```
UPDATE "Tabellenname" SET "Jahrgang" = "Jahrgang"+1 WHERE "Wiederho-
lung" = FALSE
```

Diese Bedingung funktioniert allerdings nur dann, wenn das Feld nur die Werte **`FALSE`** und **`TRUE`** annehmen kann, also nicht **`NULL`**. Sicherer wäre die Formulierung **`WHERE "Wiederholung" <> TRUE`**.

Auch andere Rechenschritte sind beim Update möglich. Wenn z.B. Waren ab 150,- € zu einem Sonderangebot herausgegeben werden sollen und der Preis um 10% herabgesetzt werden soll geschieht das mit dem folgenden Befehl:

```
UPDATE "Tabellenname" SET "Preis" = "Preis"*0,9 WHERE "Preis" >= 150
```

Bestehende Datensätze löschen

```
DELETE FROM "Tabellenname" [WHERE <Expression>];
```

Ohne einen eingrenzenden Bedingungsausdruck wird durch

```
DELETE FROM "Tabellenname"
```

der gesamte Inhalt der Tabelle gelöscht.

Da ist es dann doch besser, wenn der Befehl etwas eingegrenzt ist. Wird z.B. der Wert des Primärschlüssels angegeben, so wird nur genau ein Datensatz gelöscht:

```
DELETE FROM "Tabellenname" WHERE "ID" = 5;
```

Sollen bei einer Medienausleihe die Datensätze von Medien, die zurückgegeben wurden, gelöscht werden, so geht dies mit

```
DELETE FROM "Tabellenname" WHERE NOT "RueckgabeDatum" IS NULL;
```

oder alternativ mit

```
DELETE FROM "Tabellenname" WHERE "RueckgabeDatum" IS NOT NULL;
```

Mängel dieser Eingabemöglichkeiten

Eingaben mit einer Tabelle alleine berücksichtigt nicht die Verknüpfungen zu anderen Tabellen. Am Beispiel einer Medienausleihe sei das hier verdeutlicht:

	ID	Medien_ID	Leser_ID	Leih_Datum	Rueck_Datum	Verlaengerung	Medien_ID_BC
	0	1	0	02.11.11	04.11.11		
	1	2	2	15.10.11	25.02.12	2	
	2	0	3	02.11.11	04.04.12	1	
	3	3	0	04.11.11	28.11.11	2	
	9	5	0	28.11.11			
	10	4	0	28.11.11	04.04.12		

Die Ausleihtabelle besteht aus Fremdschlüsseln für das auszuleihende Medium *Medien_ID* und den entsprechenden Nutzer *Leser_ID* sowie einem Ausleihdatum *Leih_Datum*. In die Tabelle werden also bei der Ausleihe zwei Zahlenwerte (Mediennummer und Benutzernummer) und ein Datum eingetragen. Der Primärschlüssel wird im Feld *ID* automatisch erstellt. Ob der Benutzer zu der Nummer passt bleibt unsichtbar, es sei denn, eine zweite Tabelle mit den Benutzern wird gleichzeitig offen gehalten. Ob das Medium mit der korrekten Nummer ausgeliehen wird ist genauso wenig einsehbar. Hier muss sich die Ausleihe auf das Etikett auf dem Medium oder auf eine weitere geöffnete Tabelle verlassen.

All dies lässt sich mit Formularen wesentlich besser zusammenfassen. Hier können die Nutzer und die Medien durch Listenfelder nachgeschlagen werden. Im Formular stehen dann sichtbar die Nutzer und die Medien, nicht die versteckten Nummern. Auch kann das Formular so aufgebaut werden, dass zuerst ein Nutzer ausgewählt wird, dann das Ausleihdatum eingestellt wird und jede

Menge Medien diesem einen Datum durch Nummer zugeordnet werden. An anderer Stelle werden dann diese Nummern wieder mit entsprechender genauer Medienbezeichnung sichtbar gemacht.

Die Eingabe in Tabellen ist in Datenbanken daher nur bei einfachen Tabellen sinnvoll. Sobald Tabellen in Relation zueinander gesetzt werden, bietet sich besser ein entsprechendes Formular an. In Formularen können diese Relationen durch Unterformulare oder Listenfelder gut bedienbar gemacht werden.