LibreOffice
The Document Foundation

Base

Kapitel 3
Tabellen

Dokumentationen zu LibreOffice unter de.libreoffice.org

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 01.02.2023. Basierend auf der LibreOffice Version 7.5.

Inhalt

Kapitel 3 Tabellen 1
- Allgemeines zu Tabellen 4
- Beziehungen von Tabellen 4
  - Beziehungen zwischen Tabellen allgemein 4
  - Tabellen und Beziehungen der Beispieldatenbank 7
    - Tabellen Medienaufnahme 7
    - Tabellen Ausleihe 9
    - Tabellen Nutzerverwaltung 9
- Erstellung von Tabellen 10
  - Erstellung mit der grafischen Benutzeroberfläche 12
    - Primärschlüssel 16
    - Formatierung von Feldern 17
    - Einstellung eines Indexes 19
    - Änderung bestehender Tabellen 21
    - Probleme bei der Änderung von Tabellen 23
    - Mängel der grafischen Tabellenerstellung 25
  - Direkte Eingabe von SQL-Befehlen 25
    - Tabellenerstellung 27
    - Tabellenänderung 31
    - Tabellen löschen 33
    - Funktionserweiterung durch Trigger bei Firebird 34
- Verknüpfung von Tabellen 34
- Eingabe von Daten in Tabellen 39
  - Eingabe über die grafische Benutzeroberfläche der Tabelle 39
    - Sortieren von Tabellen 41
    - Suchen in Tabellen 42
    - Filtern von Tabellen 44
  - Eingabemöglichkeiten über SQL direkt 46
    - Neue Datensätze einfügen 46
    - Bestehende Datensätze ändern 47
    - Bestehende Datensätze löschen 48
  - Import von Daten aus anderen Datenquellen 49
    - Importierte Daten an bestehende Daten einer Tabelle anfügen 50
    - Neue Tabelle beim Import erstellen 52
    - Daten Aufsplitten beim Import 54
  - Mängel dieser Eingabemöglichkeiten 55
  - Tabellen verstecken 56

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 in den Spalten "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 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 in der Einführung im Kapitel «Eine einfache Datenbank – Testbeispiel im Detail» 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 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:

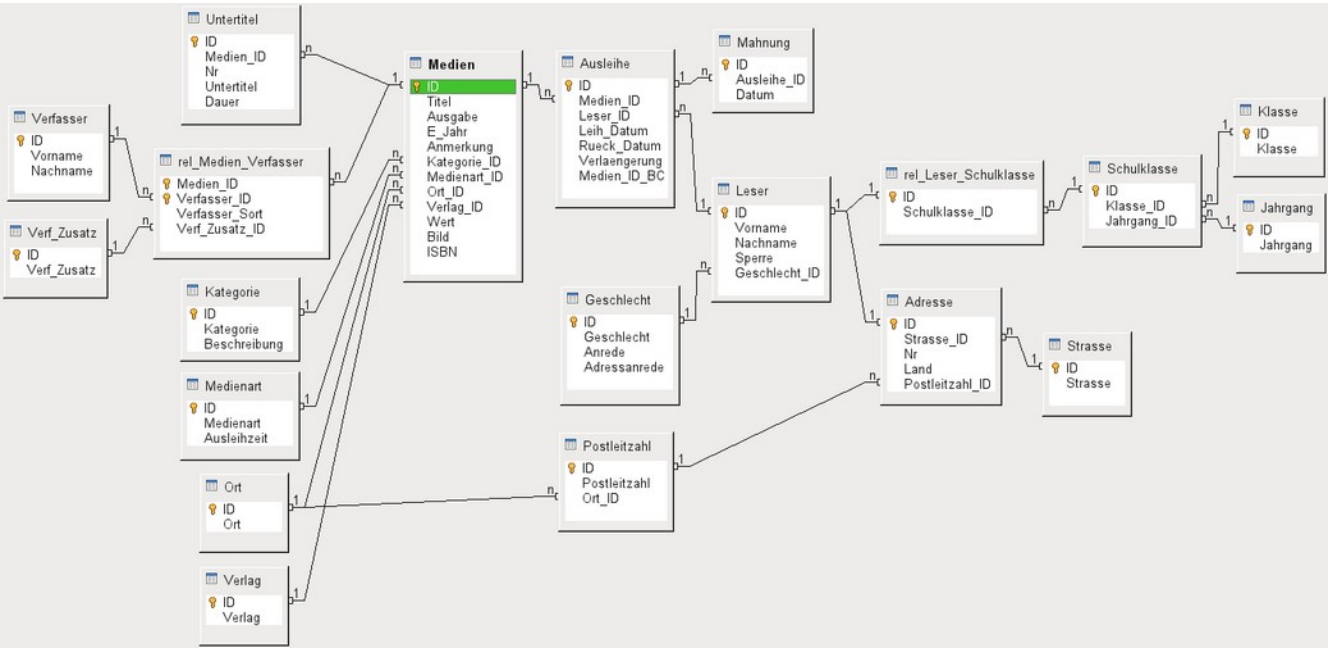


Abbildung 1: Beziehungsentwurf der Beispieldatenbank «Medien_ohne_Makros»

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.

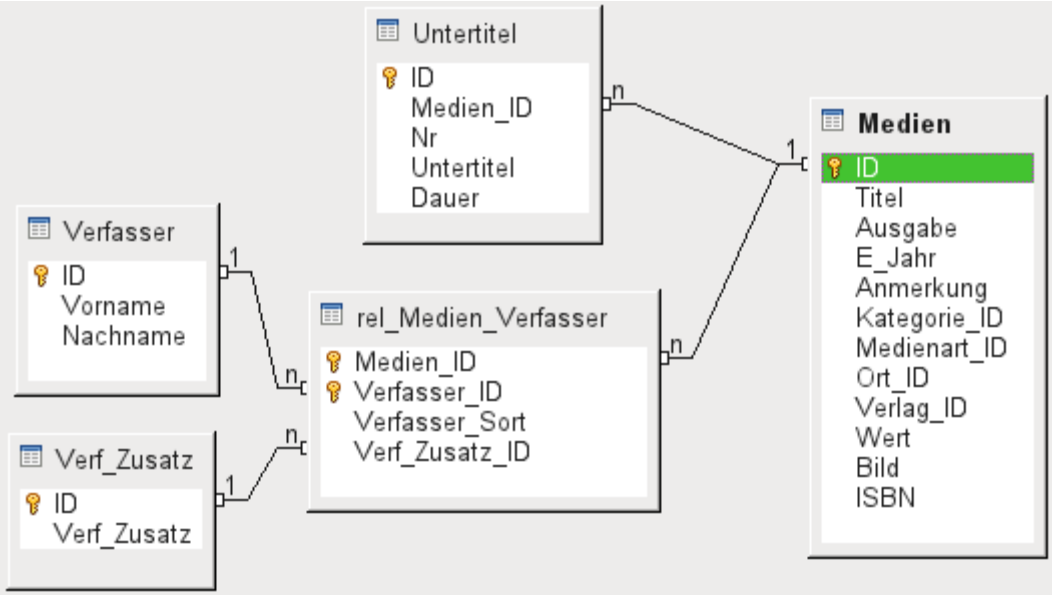


Abbildung 2: Beispiel 1:n-Beziehung; n:m-Beziehung

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 Verfassern 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:

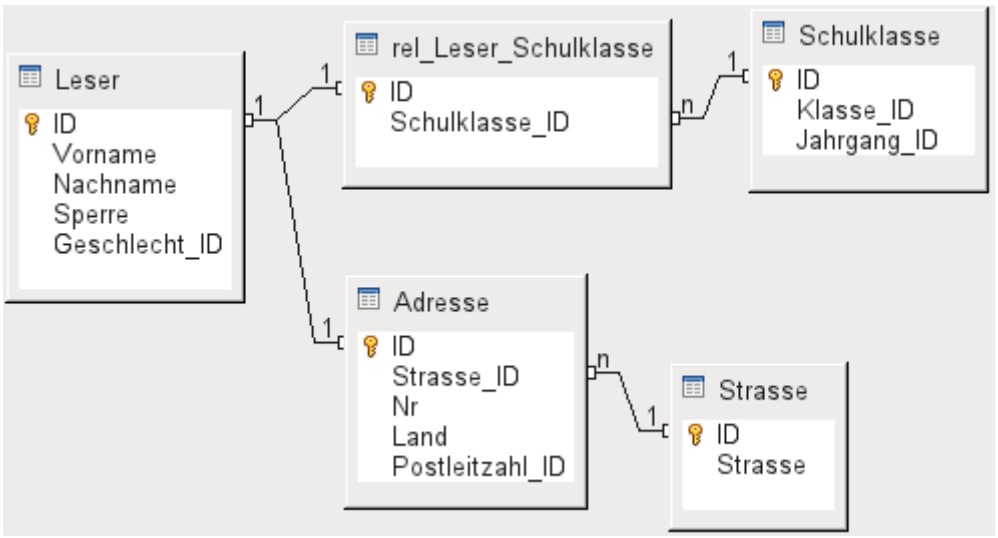


Abbildung 3: Beispiel 1:1-Beziehung

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 Klassenzuweisung über den entsprechenden Fremdschlüssel in "rel_Leser_Schulklasse".

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.

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 ähnliches 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.

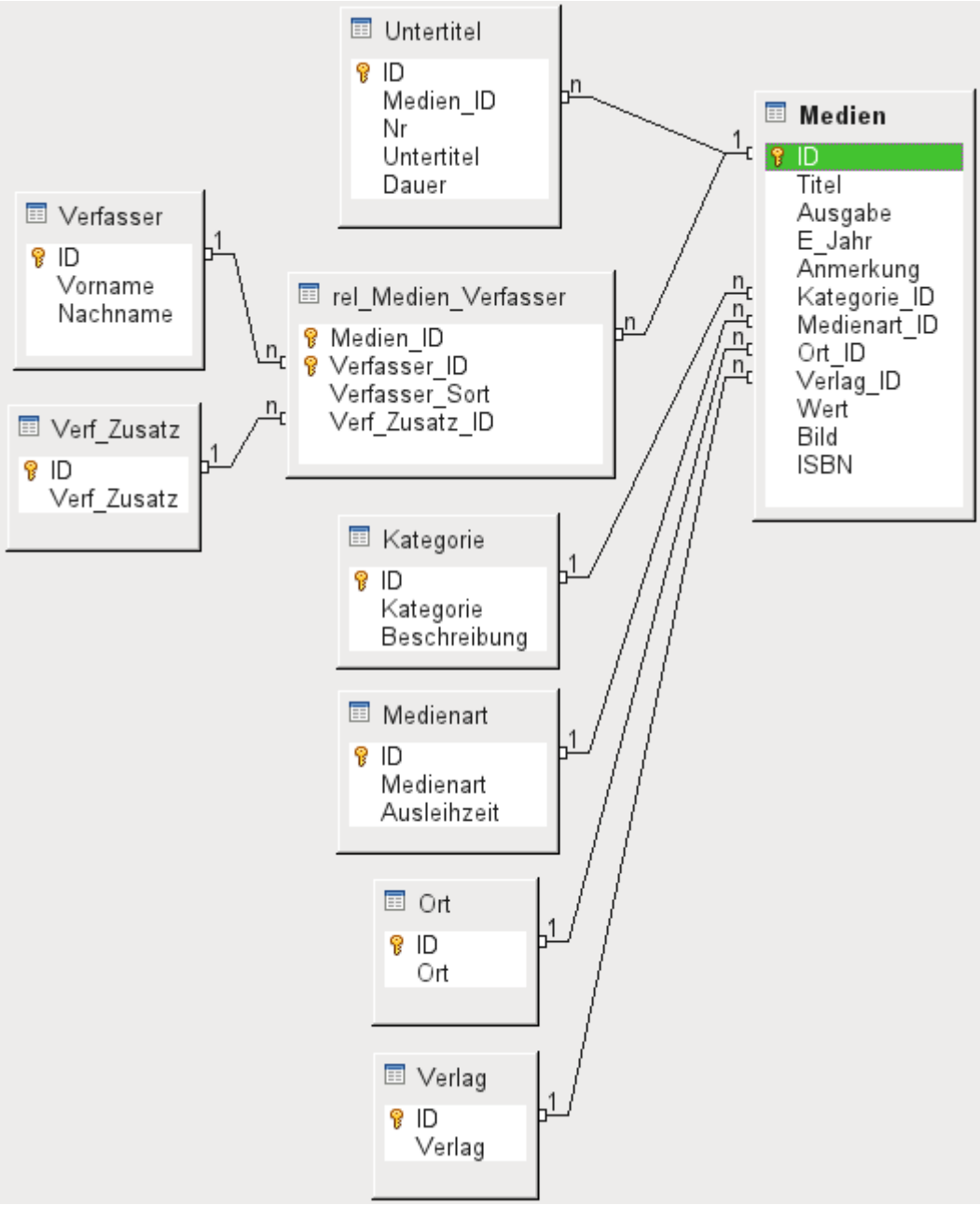


Abbildung 4: Medienaufnahme

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

Tabellen Ausleihe

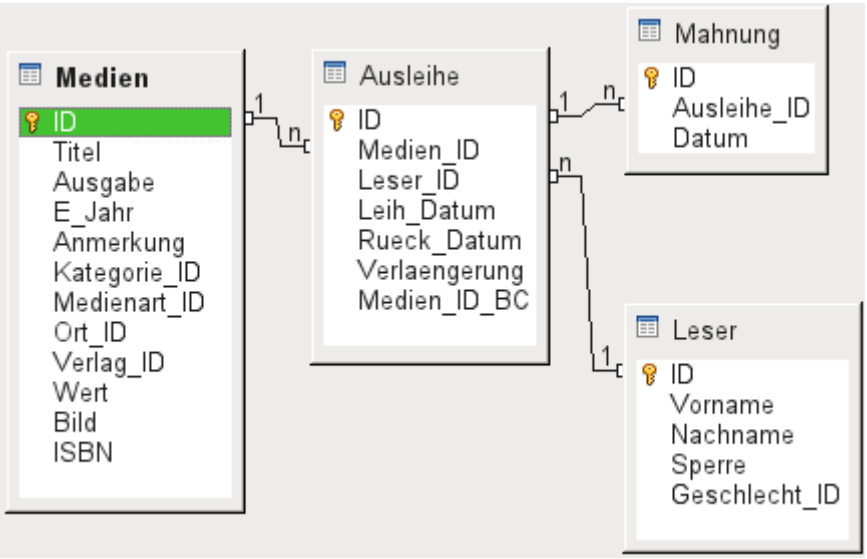


Abbildung 5: Ausleihe

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.

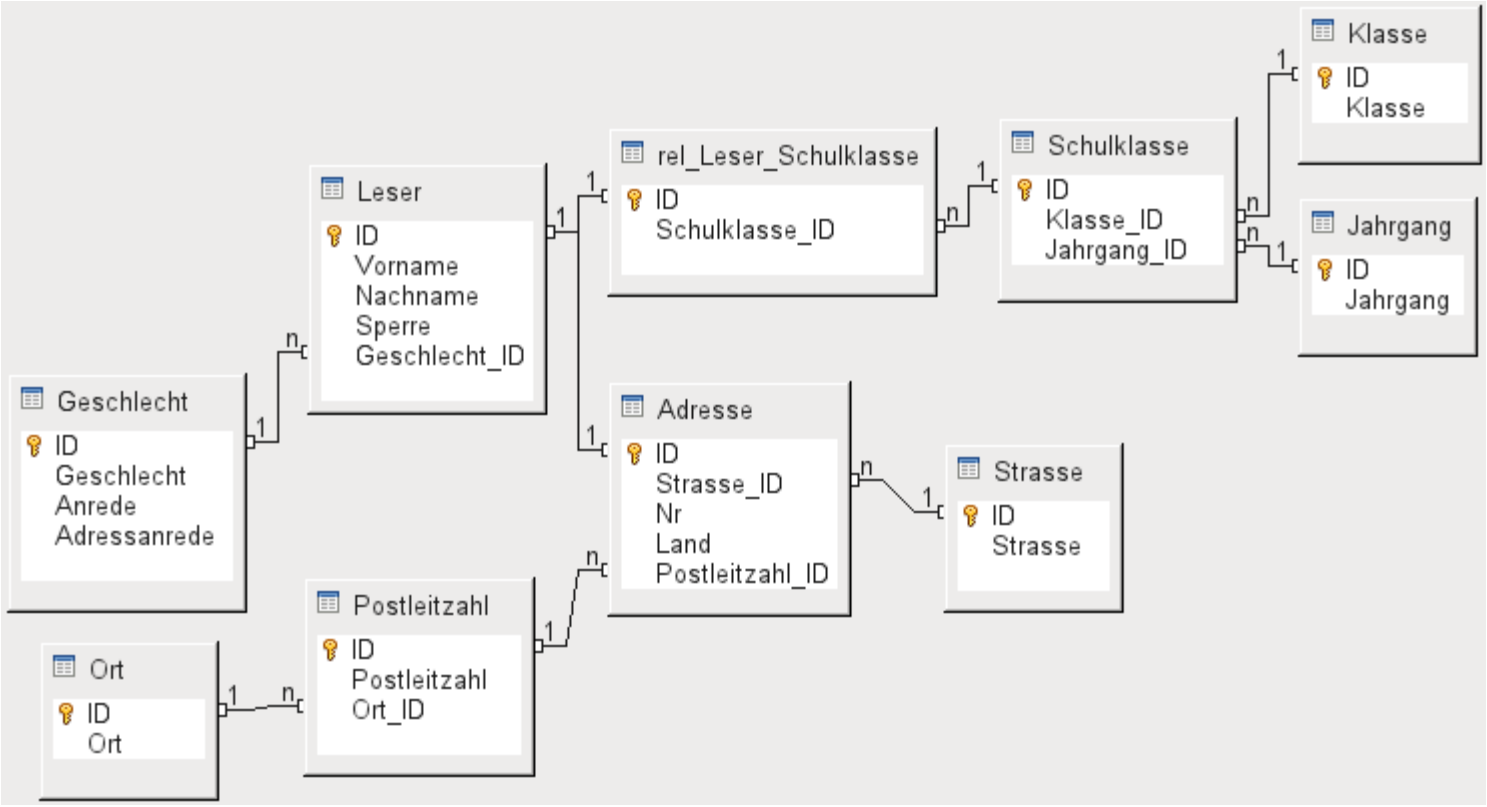


Abbildung 6: Leser - ein Schulklassenstrang und ein Adressenstrang

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.

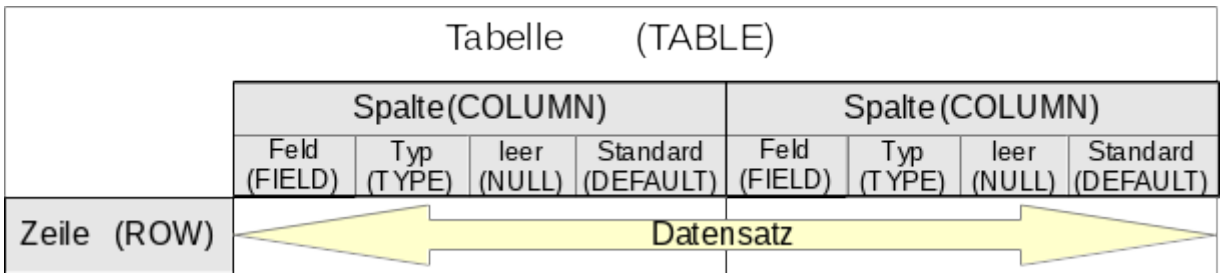
Die "**Adresse"** wird ebenfalls sehr differenziert dargestellt. Die Straße ist aus der Adresse ausgelagert, da Straßennahmen innerhalb eines Ortes häufiger wiederholt werden. Postleitzahl und Ort sind voneinander getrennt, da oft mehrere Postleitzahlen für einen Ort gelten. Für die Post sind alle Ortsbezeichnungen, 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.

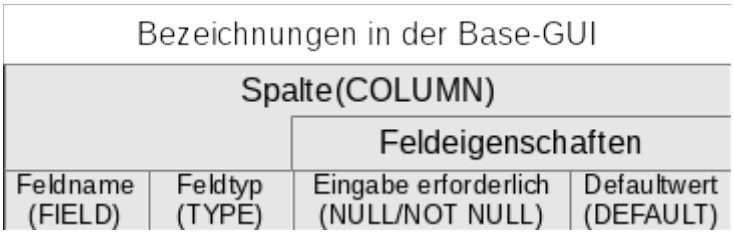
Bezeichnungen bei Tabellen:



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:



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 die *Felddefinition* im Kapitel *Direkte Eingabe von SQL-Befehlen*.

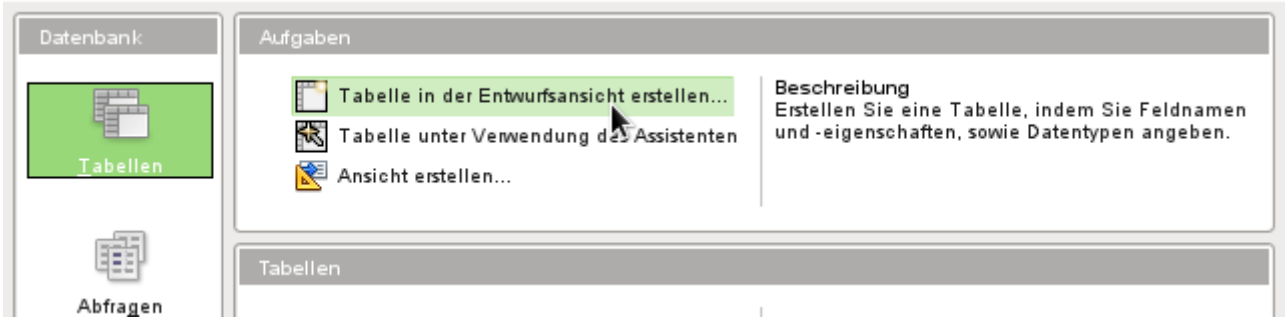
Hinweis	Defaultwert: Der Begriff «Defaultwert» in der GUI entspricht nicht dem, was Datenbanknutzer unter Defaultwert verstehen. Die GUI gibt hier einen bestimmten Wert sichtbar vor, der dann mit abgespeichert wird. Der Wert steht bereits bei der Eingabe eines neuen Datensatzes direkt in der Tabelle. Auch in einem Formular erscheint dieser Wert von vornherein bei der Eingabe eines neuen Datensatzes und muss, wenn er nicht auftauchen soll, extra gelöscht werden. Der Defaultwert einer Datenbank wird in der Tabellendefinition gespeichert. Er wird dann in das Feld geschrieben, wenn es bei der neuen Erstellung eines Datensatzes nicht bearbeitet wurde und deshalb nicht im SQL-Code auftaucht. SQL-Defaultwerte erscheinen auch **nicht** bei der Bearbeitung der Eigenschaften einer Tabelle.

Hinweis	Bei der Nutzung der internen Firebird-Datenbank muss berücksichtigt werden, dass die Tabellenbezeichnungen und Feldbezeichnungen nicht mehr als 31 Zeichen haben dürfen. Längere Bezeichnungen lässt Firebird nicht zu. Sonderzeichen werden dabei aufgrund der Codierung als 2 oder sogar mehr Zeichen gewertet. Bei Tabellennamen sollte auf die Nutzung von Umlauten und anderen Sonderzeichen besser verzichtet werden. So kann es bei manchen Datenbanktreibern z. B. bei der Verwendung von Leerzeichen zu Problemen kommen. AutoWert-Felder in PostgreSQL werden mit dem direkten Treiber dann nicht mehr einwandfrei erkannt. Besser einen Unterstrich als ein Leerzeichen nutzen!

Hinweis	Die Firebird-Datenbank hat in der Standardeinstellung Probleme mit der Sortierung von Groß- und Kleinschreibung und auch Umlauten. Der folgende Schritt sollte **vor der Erstellung der Tabellen** in Firebird einmal ausgeführt werden. Über **Extras → SQL** wird die direkte Eingabe geöffnet. `001 ALTER CHARACTER SET UTF8 SET DEFAULT COLLATION UNICODE` wird eingegeben und durch Ausführen bestätigt. Dabei erfolgt eine Rückmeldung, die für den Normalnutzer nicht verständlich ist. Die Sortierung funktioniert in den neu erstellten Tabellen aber jetzt korrekt.

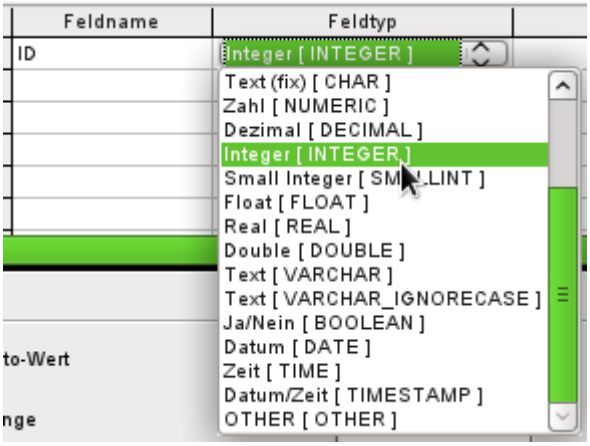
Erstellung mit der grafischen Benutzeroberfläche

Die Erstellung innerhalb der grafischen Benutzeroberfläche wird beispielhaft für die Tabelle Medien Schritt für Schritt erklärt.



Durch einen Klick auf **Tabelle in der Entwurfsansicht erstellen** wird der Tabelleneditor gestartet.

1. ID-Feld:
   - In der ersten Spalte wird «ID» als **Feldname** eingegeben. Danach wird die die Tabulator-Taste gedrückt, um in die Spalte **Feldtyp** zu wechseln. Alternativ kann auch mit der Maus die Auswahlliste der nächsten Spalte angeklickt werden.



   - **`Integer [INTEGER]`** als Feldtyp sollte aus der Auswahlliste gewählt werden. Die Standardeinstellung ist **`Text [VARCHAR]`**. Integer kann 10-stellige Zahlen speichern. Integer ist außerdem der Zahlentyp, der in der grafischen Benutzeroberfläche als einziger mit einem automatisch hoch zählenden Wert verbunden werden kann.

Tipp	Die Erstellung einer Verknüpfung zur Auswahl aus der Auswahlliste **Feldtyp**: Drücken Sie die Taste für den ersten Buchstaben der Wahl. Sie können die Auswahl durch wiederholtes Drücken des Buchstabens wechseln. Zum Beispiel wechselt das Drücken der Taste *D* von «Datum» auf «Datum/Zeit» bzw. danach auf «Dezimal».

- «ID» wird als Primärschlüssel gesetzt, indem mit der rechten Maustaste auf das grüne Dreieck auf der linken Seite der Zeile «ID» geklickt und **Primärschlüssel** in dem Kontextmenü ausgesucht wird. Dies stellt ein Schlüssel-Symbol vor «ID».

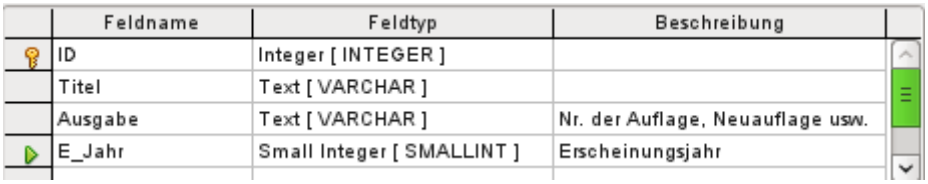
Hinweis	Der Primärschlüssel dient nur einem Zweck – nämlich zur eindeutigen Identifizierung des Datensatzes. Daher kann ein beliebiger Name für dieses Feld verwendet werden. Im Beispiel wurde die allgemein übliche Bezeichnung ID (Identifikation) verwendet.

- Bei den Feldeigenschaften für das Feld «ID» wird die Eigenschaft **Auto-Wert → Ja** gesetzt. Damit wird der Primärschlüssel automatisch hoch gezählt. Die Zählung beginnt bei der internen HSQLDB-Datenbank mit '0'.
  Der Auto-Wert lässt sich nur für **ein** Feld einer Tabelle einstellen. Die Auswahl **Auto-Wert → Ja** erzeugt automatisch den Primärschlüssel für das Feld, sofern der Primärschlüssel nicht schon vorher bestimmt wurde.

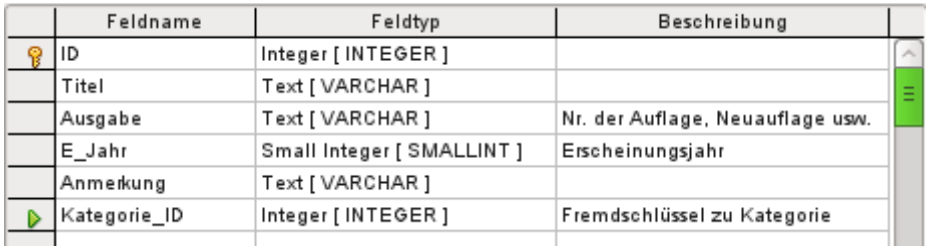
2. Das nächste Feld ist das Feld «Titel»
   - Der Feldname «Titel» wird in der Spalte «Feldname» eingegeben.

- Der Feldtyp muss hier nicht geändert werden, da er bereits auf **Text [VARCHAR]** eingestellt ist.

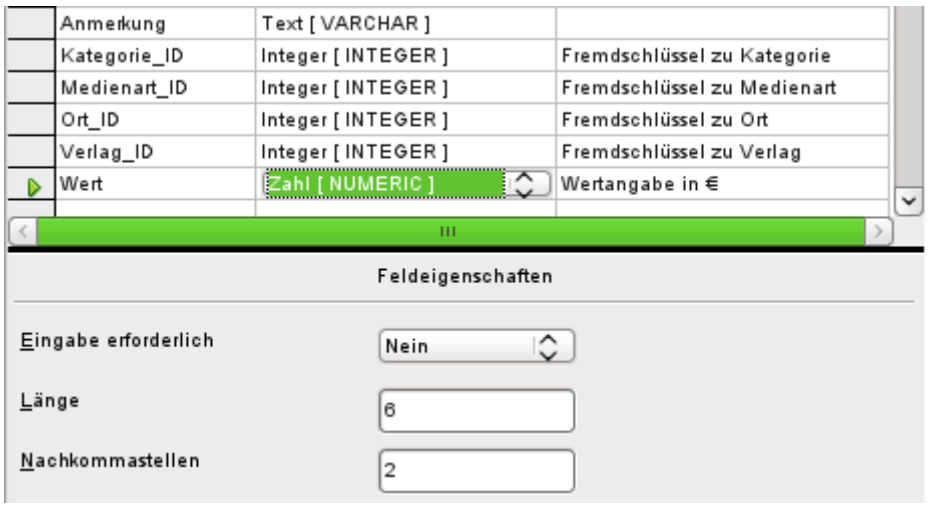
- In den Feldeigenschaften muss die Länge des Feldes angepasst werden. Die Standardlänge ist hier bei den neueren LO-Versionen 100, sollte für den Medientitel aber auf 250 Zeichen erweitert werden.
- In den Feldeigenschaften sollte wenigstens bei diesem Feld **Eingabe erforderlich → Ja** gewählt werden. Ein Medium ohne Titel wird keinen Sinn machen.



3. **Beschreibung** kann alles sein; die Spalte kann auch leer gelassen werden. Die Beschreibung dient nur zur Erklärung des Feldinhaltes für Personen, die sich später einmal den Tabellenentwurf ansehen wollen.
4. Für das Feld «E_Jahr» wird der Zahlentyp **Small Integer [SMALLINT]** gewählt. Dabei handelt es sich um eine maximal fünfstellige ganze Zahl. Mit dem Erscheinungsjahr soll zwar nicht gerechnet werden, es sollte aber zumindest vermieden werden, dass dort Buchstaben auftauchen.



5. Für das Feld «Kategorie_ID» wird der Feldtyp **Integer [INTEGER]** gewählt. Da in der Tabelle «Kategorie» der Primärschlüssel diesen Feldtyp haben soll muss auch der hier eingetragene Fremdschlüssel den gleichen Feldtyp haben.
   Das gilt dann entsprechend auch für die nachfolgenden Fremdschlüssel «Medienart_ID», «Ort_ID» und «Verlag_ID».



6. Für die Wertangabe wird der Feldtyp **`Zahl [NUMERIC]`** oder auch **`Dezimal [DECIMAL]`** gewählt. Diese beiden Zahlenfelder können Zahlen mit Nachkommastellen speichern.
   - Bei den **Feldeigenschaften** wird eine **Länge** von 6 Zeichen eingestellt. Das dürfte für die Wertangabe der Medien reichen.
   - Die Zahl der **Nachkommastellen** wird auf 2 Zeichen festgelegt. Damit ist jetzt maximal eine Wertangabe von 9999,99 möglich, da das Komma nicht als Stelle mitgerechnet wird.
   - Eine Formatierung der Ausgabe als € ist nicht notwendig. Diese Einstellung erfolgt im Formular. Sie kann bei Bedarf auch später bei einer eventuellen direkten Eingabe in die Tabelle vorgenommen werden. Die Tabellenformatierung beeinflusst nicht die Formatierung im Formular, wohl aber die Formatierung bei Abfragen.

7. Für das Feld «ISBN» wird der Feldtyp **`Zahl [NUMERIC]`** gesetzt. Dieser kann genau auf die Feldlänge einer ISBN-Nummer eingestellt werden. ISBN-Nummern sind 10 oder 13 Zeichen lang. Sie werden also später als Zahl ohne Trenner gespeichert.
   Die Länge wird entsprechend auf maximal 13 Zeichen eingestellt. Nachkommastellen bleiben dabei auf 0 gesetzt.
8. Die Tabelle wird mit dem Namen «Medien» abgespeichert.

Die zentrale Tabelle für die Beispieldatenbank wurde nun erstellt. Mit dem entsprechenden Verfahren können alle weiteren Tabelle ebenfalls erstellt werden. Achten Sie immer darauf, dass der Feldtyp zusammen mit den Feldeigenschaften vorbestimmt, was in dem Feld abgespeichert werden kann. Dies ist anders als in einer Tabellenkalkulation, die vollkommen durchmischte Eingaben in einer Spalte zulassen.

Hinweis	Die Reihenfolge der Felder in der Tabelle kann nur bis zum ersten Abspeichern über die grafische Benutzeroberfläche beeinflusst werden. Sie lässt sich später über **Extras → SQL** nur bei Verwendung von FIREBIRD ändern. Bei Verwendung der internen HSQLDB sind die bereits erstellten Felder unbeweglich. In Abfragen, Formularen oder Berichten ist die Reihenfolge allerdings weiterhin frei zusammenstellbar.

Primärschlüssel

Wird beim Tabellenentwurf kein Primärschlüssel festgelegt, so erscheint beim Abspeichern eines Tabellenentwurfs 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 interne Datenbank auf die Tabelle nicht zugreifen. In der Regel wird dieses Feld mit dem Kürzel "ID" bezeichnet, mit dem Zahlentyp INTEGER versehen und als «AutoWert» automatisch mit einer fortlaufenden Nummer versehen. Mit einem Rechtsklick auf das entsprechende Feld kann es zum Primärschlüsselfeld erklärt werden.

Hinweis	Die Notwendigkeit, einen Primärschlüssel zu wählen, wird nicht durch die verwendete Datenbank sondern durch die Benutzeroberfläche vorgeschrieben. Grundsätzlich ist es möglich, auch in der internen HSQLDB Tabellen ohne Primärschlüssel zu erstellen. Sie lassen sich dann allerdings nicht über die grafische Benutzeroberfläche mit Daten befüllen. Stattdessen können die Daten der Tabelle über **Extras → SQL** direkt mit SQL-Befehlen geändert werden. Auch der Zugriff über Makros ist möglich. Dieses Verhalten lässt sich natürlich auch bewusst nutzen: Wenn eine Datenbank an andere Personen weitergegeben werden soll, aber die Veränderung bestimmter Tabellen möglichst erschwert werden soll, so reicht es, einfach den Primärschlüssel (nicht das Feld, nur den Schlüssel!) zu löschen. Dafür darf die Tabelle allerdings nicht über **Extras → Beziehungen** mit anderen Tabellen verknüpft sein.

Es können ohne weiteres auch mehrere Felder zum gemeinsamen Primärschlüssel erstellt werden. Hierzu müssen nur die entsprechenden Felder gemeinsam markiert werden (Taste *Strg* oder *Shift* gedrückt halten). Dann kann über den Rechtsklick der Primärschlüssel allen markierten Feldern zugewiesen werden.

Sollen von einer anderen Tabelle in dieser Tabelle Informationen mitgeführt werden (Beispiel: Adressdatenbank, aber ausgelagert Tabellen jeweils für 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 "Adresse" 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.

Vorsicht 	Für die Erstellung eines Fremdschlüssels ist immer der gleiche Datentyp in Primärschlüsselfeld und Fremdschlüsselfeld notwendig. Bei der Verwendung von Feldern des Typs VARCHAR wird aber nicht berücksichtigt, welche Länge das Feld hat. Es ist also sehr wohl möglich, ein Primärschlüsselfeld mit VARCHAR(10) zu definieren, bei dem Fremdschlüsselfeld aber nur VARCHAR(5) zu wählen. Dann lassen sich nicht alle Werte aus dem Primärschlüsselfeld in das Fremdschlüsselfeld übertragen.

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 "Adresse" 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 bei der HSQLDB nicht als Felder mit Bruchteilen einer Sekunde dargestellt werden. Dies geht nur mit einem Timestamp-Feld. Mit der grafischen Benutzeroberfläche wird allerdings nur ein Timestamp-Feld erzeugt, das Datum, Stunde, Minute und Sekunde abspeichert. Dieses Feld muss über **Extras → SQL** noch entsprechend verändert werden.

```
001 ALTER TABLE "Tabellenname" ALTER COLUMN "Feldname" TIMESTAMP(6)
```

Mit dem Parameter '6' wird das Timestamp-Feld bei der internen HSQLDB auch für Bruchteile von Sekunden aufnahmefähig.
Die FIREBIRD-Datenbank benötigt diesen Parameter nicht, kann aber erst ab LO 6.1 über die GUI Millisekunden in Zeitfeldern oder Timestampfeldern speichern.

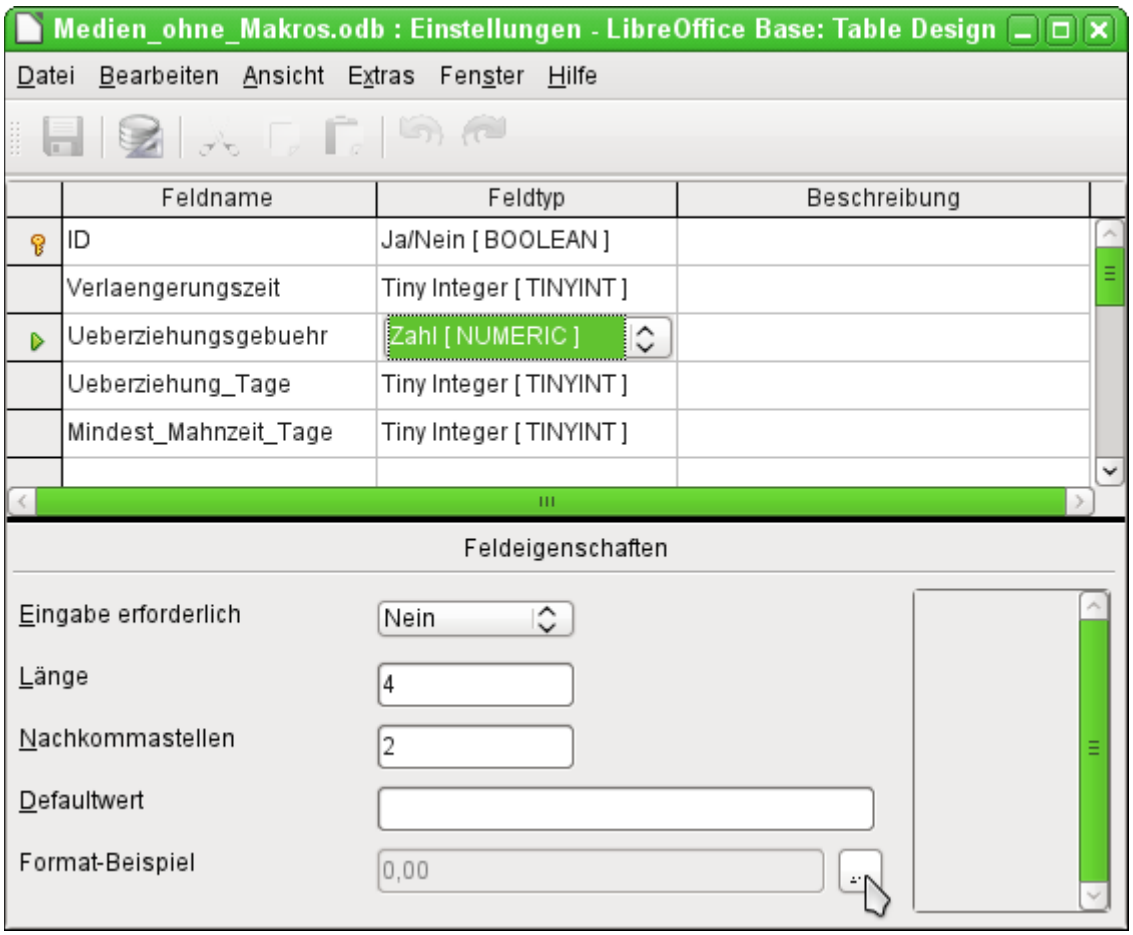
Formatierung von Feldern

Die Formatierung stellt die Werte der Datenbank für den Nutzer dar und erlaubt eine Eingabe von Werten in Abhängigkeit von landesüblichen Eingabeformen. Ohne Formatierung werden Dezimalstellen mit einem Punkt abgetrennt (**4.21** statt **4,21**), Datumswerte im Format **2014-12-22** dargestellt. Bei der Einstellung der Formatierung muss deshalb auf die Landeseinstellung geachtet werden.

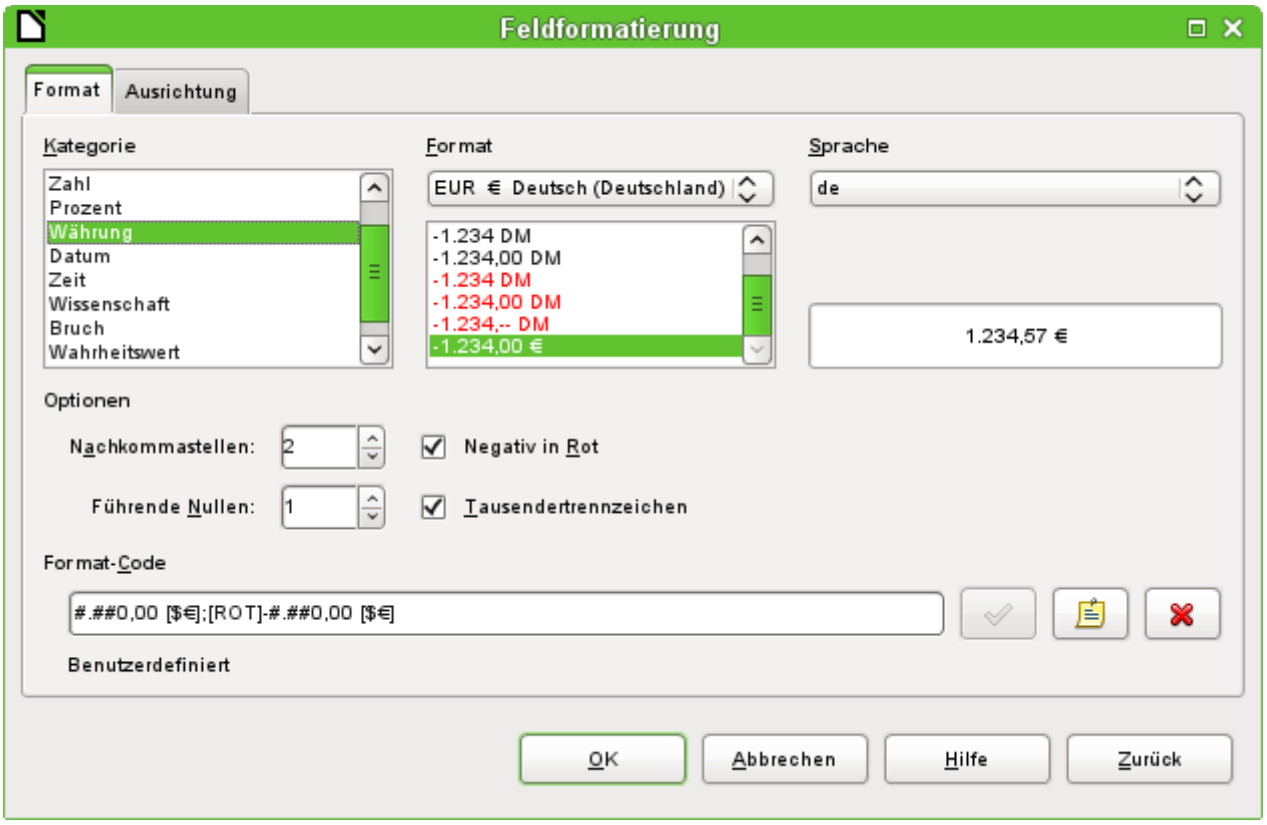
Die Formatierung zeigt die Inhalte nur an. Eine Datumsdarstellung mit einer zweistelligen Jahreszahl wird dennoch als vierstellige Jahreszahl gespeichert. Wird ein Feld für eine Zahl mit zwei Nachkommastellen erstellt, wie es bei der Überziehungsgebühr in dem folgenden Beispiel der Fall ist, so wird auch eine Zahl mit zwei Nachkommastellen gespeichert, auch wenn irrtümlich die Formatierung vielleicht ohne Nachkommastellen erstellt wurde. Eine Zahl mit zwei Nachkommastellen lässt sich sogar in ein Feld eingeben, das ohne Nachkommastellen formatiert wurde. Die Nachkommastellen verschwinden scheinbar bei der Eingabe, werden aber wieder sichtbar, wenn die Formatierung angepasst wird.

Soll nur eine Zeit, aber kein Datum gespeichert werden, so sollte die Tabelle entsprechend z.B. so formatiert werden, dass nur Minuten, Sekunden und Zehntelsekunden abgefragt werden: MM:SS,00. Eine Formatierung mit Nachkommastellen im Millisekundenbereich ist später in Formularen nur über das formatierte Feld, nicht über das Zeitfeld möglich.

Die Formatierung von Feldern wird bei Erstellung der Tabelle oder auch anschließend in den Feldeigenschaften über einen gesonderten Dialog vorgenommen:



Über den Button in **Feldeigenschaften → Format-Beispiel** wird der Dialog zur Änderung des Formates gestartet.



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. Dies hat allerdings nur Auswirkungen auf die Eingabe in der Tabelle und auf Abfragen, die den Wert ohne Umrechnungen auslesen. In Formularen muss die Währungsbezeichnung gesondert formatiert werden.

Hinweis	Base speichert zur Zeit nur die Formatierungen der Tabelle ab, die entweder beim Erstellen der Tabelle oder bei der Eingabe von Daten über die Spaltenköpfe erfolgt. Dies gilt auch für die Spaltenbreiten bei der Eingabe. Gesonderte Formatierungen von Abfragen werden dagegen nicht gespeichert. Bei Abfragen wird, sofern das von der GUI her möglich ist, auf die Formatierung der Felder in den Tabellen zurückgegriffen.

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. Bei einer Tabelle mit Primärschlüssel, wie dies in der internen HSQLDB üblich ist, ist allerdings der Index des Primärschlüssels als erster eindeutiger Index für die Sortierung bereits maßgebend.

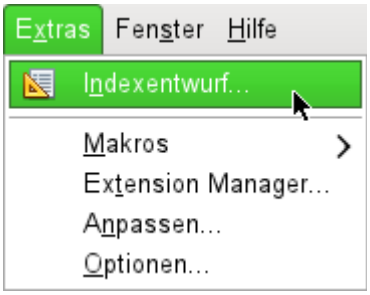


Abbildung 7: Zugriff auf den Indexentwurf

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...** oder direkt über den entsprechenden Button in der Menüleiste des Tabellenentwurfes.

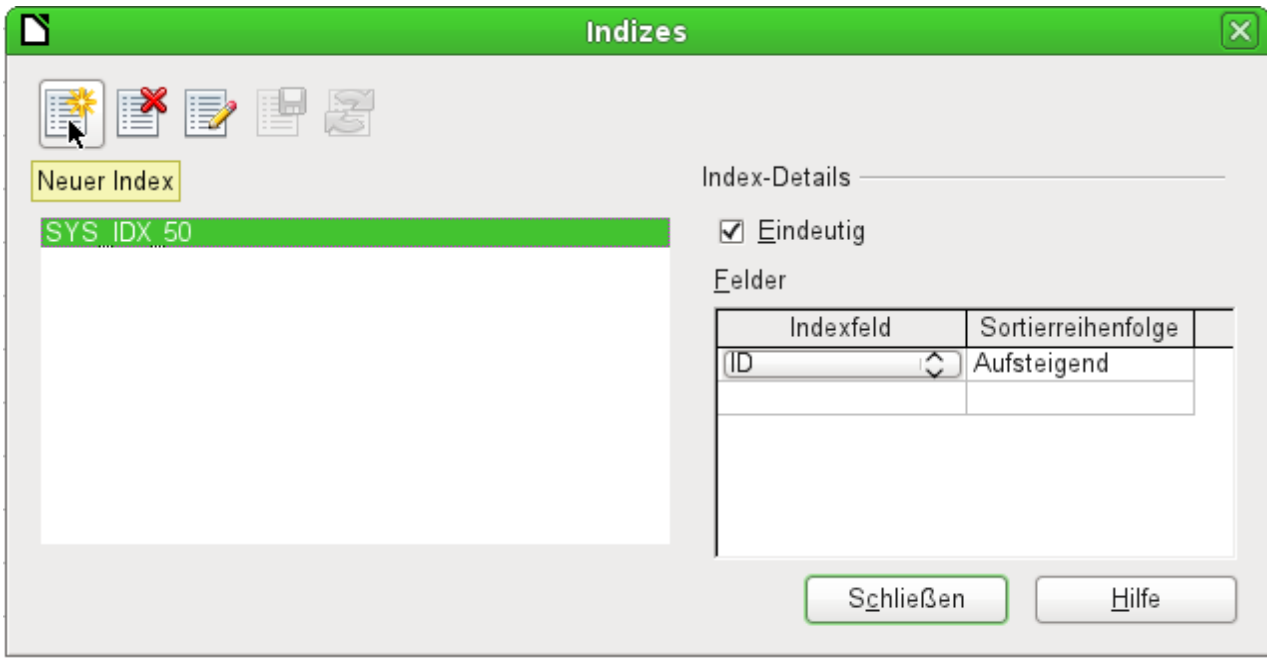


Abbildung 8: Erstellen eines neuen Indexes

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

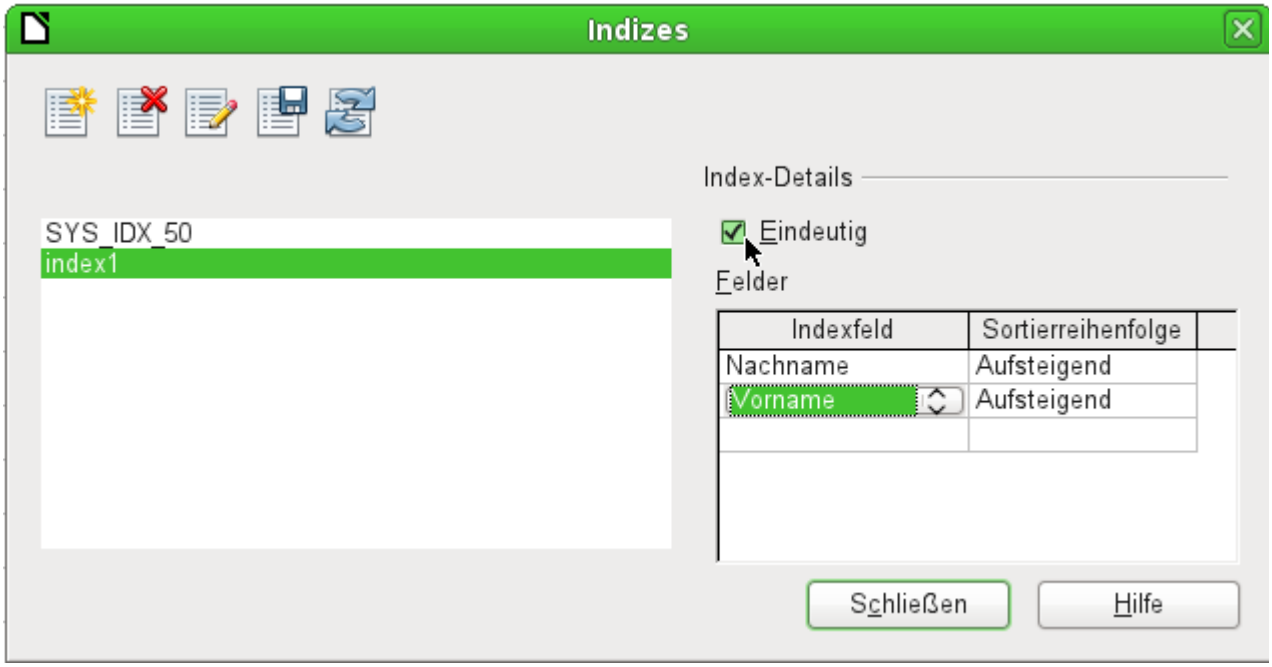


Abbildung 9: Der Index wird als «Eindeutig» definiert.

Dem neuen Index wird automatisch die Bezeichnung «index1» zugewiesen. Diese Bezeichnung kann geändert werden. 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.

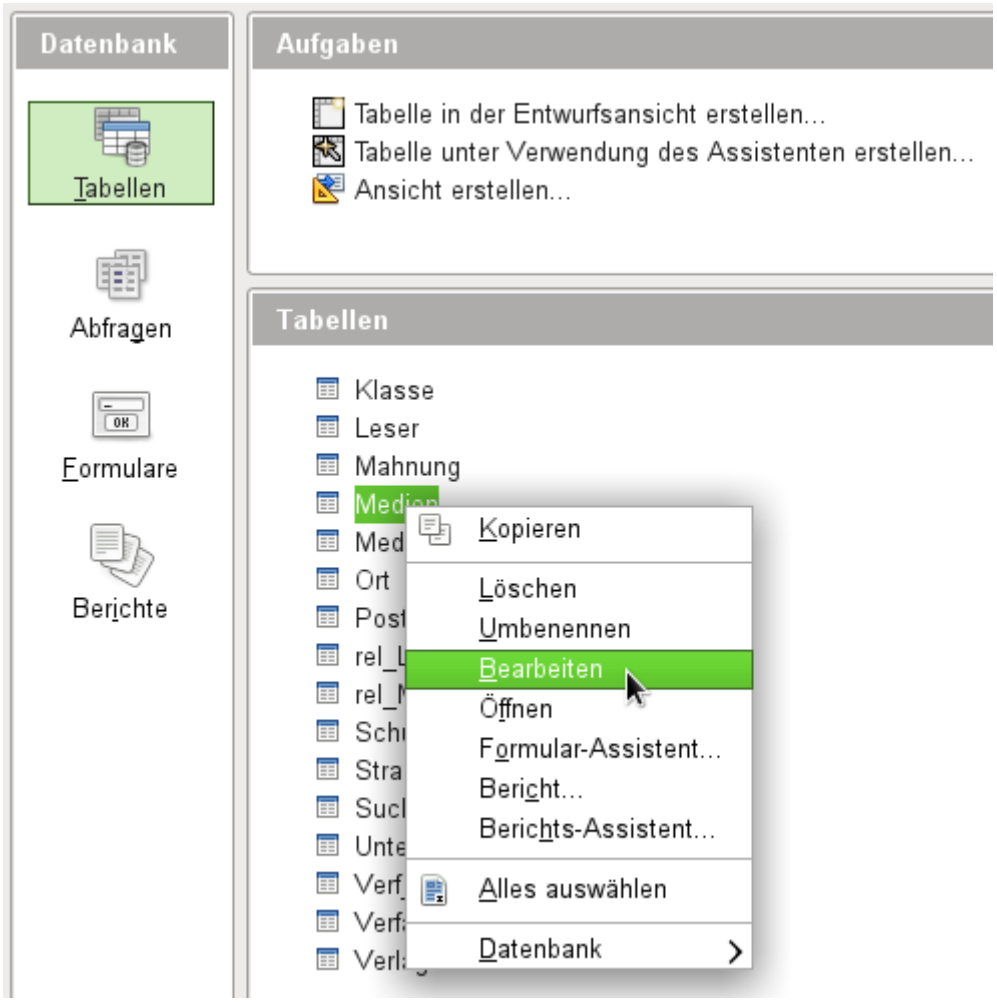
Eine Sonderstellung für einen eindeutigen Index nimmt in einem Feld das Fehlen eines Eintrages, also NULL, ein. Da NULL alle beliebigen Werte annehmen könnte ist es ohne weiteres erlaubt, bei einem Index über zwei Felder in einem Feld mehrmals hintereinander die gleiche Eingabe zu tätigen, solange in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

Änderung bestehender Tabellen

Nicht alle Tabellen werden bereits beim Erstellen so weit durchgeplant sein, dass keine Änderungen mehr vorzunehmen sind. Mit einem rechten Mausklick auf eine Tabelle wird das folgende Kontextmenü sichtbar:



Mit einem Klick auf **Bearbeiten** öffnet sich der grafische Bearbeitungsmodus. Hier können Feldnamen, Feldtypen und Formate geändert werden. Natürlich können auch weitere Felder hinzugefügt werden.

Die Änderungen sind allerdings nicht völlig problemlos möglich. Soll ein Feldname geändert werden, so ist es angeraten, zuerst den neuen Namen an den alten Feldnamen anzuhängen und dann die vorherige Benennung zu entfernen. Andernfalls verschwindet das ganze Feld.

Neue Felder können nur am Ende der Liste hinzugefügt werden, auch wenn der Bearbeitungsmodus etwas anderes scheinbar ermöglicht. Das Einfügen eines Feldes mitten in eine Liste ist in der GUI nicht möglich.

Auch Kommentare zu vorher erstellten Feldern werden bei einer anschließenden Tabellenänderung oft nicht mit abgespeichert.

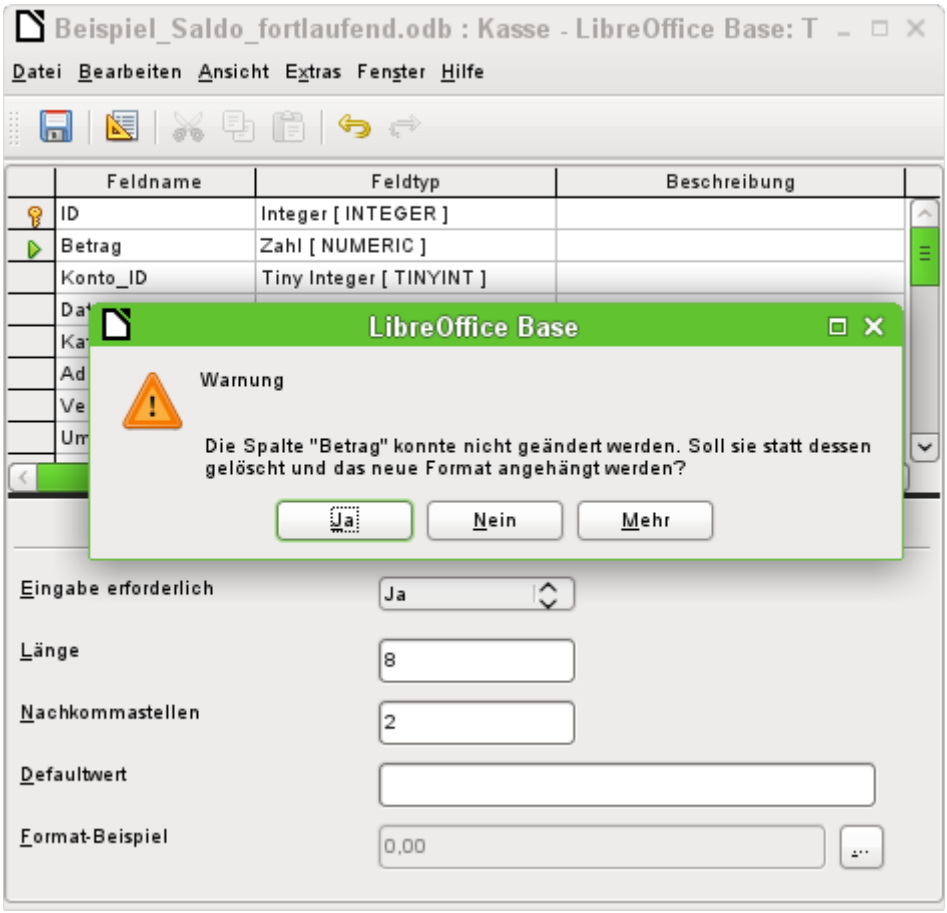
Sind bereits erst einmal mehrere Tabellen erstellt worden, so können die Beziehungen zwischen den Tabellen eine Änderung von Feldeigenschaften blockieren. Als Beziehungen nimmt die Datenbank sowohl die unter **Extras → Beziehungen** erstellten Verknüpfungen als auch die in Tabellenansichten erstellten Beziehungen wahr.

Hinweis	Bei der internen ~~Firebird~~-Datenbank ist es nicht möglich, Tabellennamen zu ändern. Auch ist es nicht möglich, die Namen für Primärschlüsselfelder zu ändern. Hier hilft dann nur die Tabelle zu kopieren und neu einzufügen. Der Assistent ermöglicht schließlich die Eingabe neuer Namen für die Tabelle und auch neuer Bezeichnungen für das Primärschlüsselfeld.

Probleme bei der Änderung von Tabellen

Tabellen sollten am besten direkt bei der Erstellung komplett mit allen nötigen Einstellungen versehen werden. Sollen hinterher die Eigenschaften von Feldern verändert werden (Feldname, erforderliche Eingabe usw.), so kann das häufig zu einer Fehlermeldung führen, die nicht Ursache der GUI ist, sondern in der darunterliegenden Datenbank begründet ist.

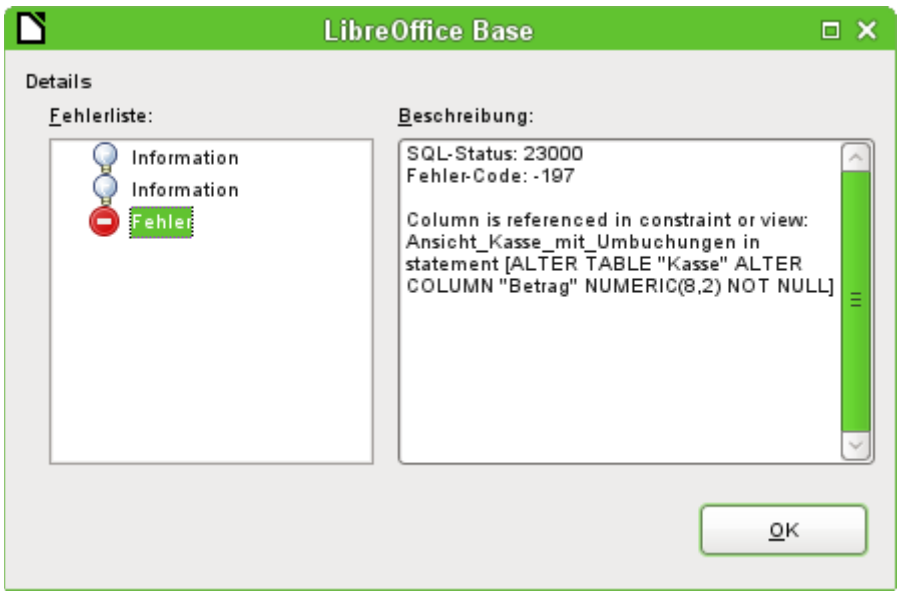
Die folgende Tabelle aus einer dem Handbuch nicht beiliegenden Datenbank zeigt hier gleich mehrere Ursachen, die eine Änderung eines Feldes in der Tabelle «Kasse» unmöglich machen.



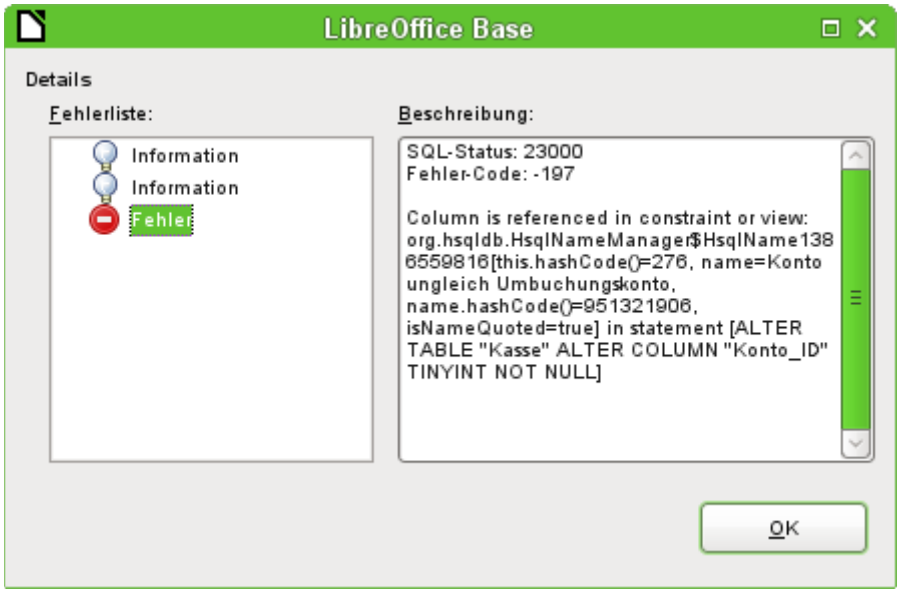
Hier sollte das Feld "Betrag" in der Eigenschaft «Eingabe erforderlich» auf «Ja» umgestellt werden. Das Warnsymbol macht bereits darauf aufmerksam: Die Änderung kann zu einem Verlust von Daten führen. Eine einfache Änderung ist nicht möglich. Bereits vorher wurde ausgeschlossen, dass in dem Feld "Betrag" eventuell ein Datensatz ohne Eingabe existiert.

Ein Klick auf Ja führt zu einer weiteren Fehlermeldung, da die Datenbank die Löschung nicht zulässt. Ein Klick auf Nein bricht den gesamten Vorgang ab. Häufig wird die Frage nach Mehr Informationen gar nicht erst gestellt, weil die Informationen nur mit etwas gründlicherem Wissen auch sinnvoll zu einer anderen Handlungsweise und dann zum gewünschten Ziel führen können.

Hinweis	Bei der internen FIREBIRD Datenbank sind grundsätzlich keine Änderungen eines Feldes von einer nicht erforderlichen zu einer erforderlichen Eingabe über die GUI möglich, sofern die Tabelle erst einmal abgespeichert wurde. Hier hilft nur die *Tabellenänderung* über direktes SQL



Die Fehlermeldung **`Column is referenced in constraint or view`** bedeutet:
Auf die Spalte mit dem Feldnamen "Betrag" wird an anderer Stelle der Datenbank bereits Bezug genommen. Dies kann eine *Bedingungsdefinition* oder eine «Tabellenansicht» sein, die nach dem Erstellen der Tabelle von dem Nutzer erstellt wurde. In der obigen Abbildung wird noch darauf hingewiesen, wie der Name der Bedingungsdefinition oder Ansicht heißt: "Ansicht_Kasse_mit_Umbuchungen". Damit ist für den Nutzer klar, an welcher Stelle angesetzt werden muss. Zuerst sollte der SQL-Code der Ansicht z.B. in einer Abfrage gesichert werden, dann die Ansicht gelöscht werden und danach kann ein neuer Versuch gestartet werden.



Wieder die entsprechende Meldung, nur mit einer umfangreicheren Erklärung. Nur der sinnvollen Benennung der Bedingung «Konto ungleich Umbuchungskonto» ist es zu verdanken, dass die Bedingungsdefinition auch auffindbar ist. Hier ist der Spalte mit dem Feldnamen "Konto_ID" die Bedingung zugeordnet worden, dass eine weitere Spalte in der gleichen Tabelle im gleichen Datensatz nicht den gleichen Wert haben darf. Erst wenn diese Bedingung wieder entfernt wird ist es möglich, einen erneuten Versuch zu starten, die Spalte zu verändern.

Taucht jetzt noch wieder ein Fehler auf, so liegt dieser häufig darin begründet, dass das entsprechende Feld mit einem Feld einer anderen Tabelle in der Beziehungsdefinition verknüpft wurde. Hier muss dann zuerst die Beziehung unter **Extras → Beziehungen** gelöst werden, bevor die Änderung vorgenommen werden kann.

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]

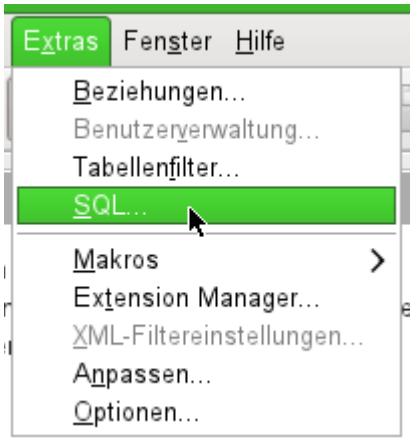
Nur über direkte SQL-Eingabe kann ein neues Feld an eine bestimmte Position innerhalb der Tabelle rutschen. Bereits erstellte Felder sind bei der internen HSQLDB nicht beweglich. Für Firebird müssen die Felder erstellt werden und können dann an die entsprechende Position verschoben werden.

Eigenschaften der Tabellen sollten sofort festgelegt werden. Welche Felder sollen nicht NULL sein, welche einen Standardwert (Default) erhalten. Diese Eigenschaft kann hinterher häufig nur unter Berücksichtigung der oben genannten Fehlermeldungen 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 unter «Ausgabe» dargestellt werden, wenn das Markierfeld angekreuzt wird.
SQL-Befehl direkt ausführen schickt den Befehl direkt an die Datenbank ohne eventuell noch Code nach Base-internen Regeln zu verändern («EscapeProcessing»).
Ist der Dialog einmal geöffnet, so werden die dort erstellten Befehle zwischengespeichert und können erneut aufgerufen werden.

Hinweis	Befehle, die hier eingegeben werden, werden direkt an die Datenbank weiter gegeben. Die grafische Oberfläche von Base bekommt davon erst einmal nichts mit. Das fällt auf, wenn z.B. neue Tabellen über **Extras → SQL** erstellt werden. Damit die Tabellen auch in der grafischen Oberfläche erscheinen muss **Ansicht → Tabellen aktualisieren** ausgelöst werden.

1 *Bug 51605*

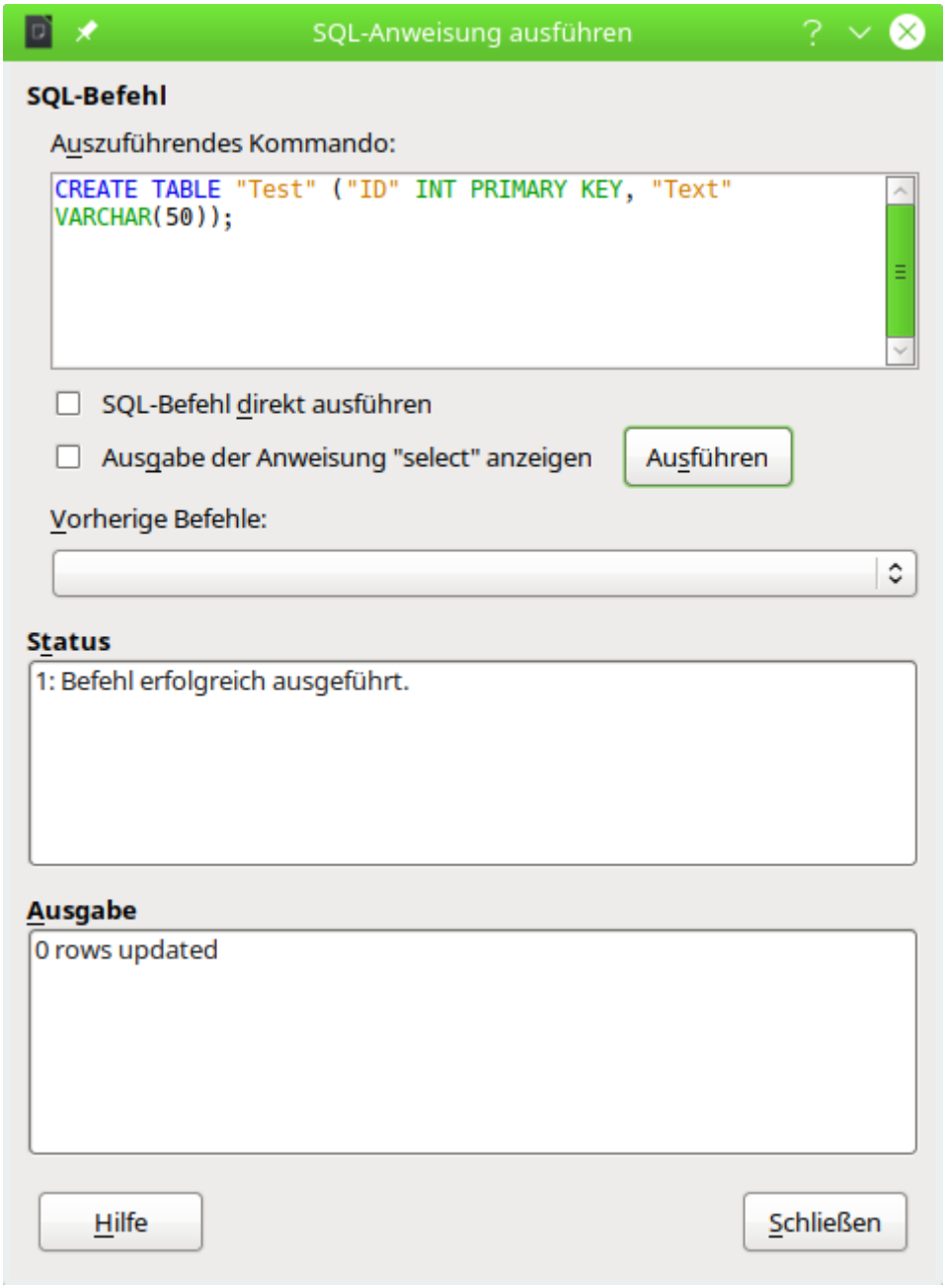


Abbildung 10: Fenster für direkte Eingabe von SQL-Befehlen

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 2.5. 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.

Für die interne Firebird-Datenbank wurden die entsprechenden Informationen aus *http://www.firebirdsql.org/en/reference-manuals/* herangezogen. Hier liegt zum Erscheinen der Version LO 6.1 allerdings nur eine ausführliche Dokumentation zu Firebird-Version 2.5, nicht zur in LO 6.1 verbauten Firebird-Version 3.0, vor. Auch hier gilt natürlich: Längst nicht alle Funktionen der externen Datenbank haben für die interne Datenbank eine Bedeutung. Manche der intern gut nutzbaren Funktionen sind leider auch noch nicht in LO 7.4 umgesetzt.

Viele Funktionen stehen für beide Datenbanken zur Verfügung. Unterschiede zwischen der HSQLDB und Firebird werden entsprechend gekennzeichnet. Mit Grün ist gekennzeichnet, wenn eine Funktion für eine Datenbank zur Verfügung steht. Mit ~~Rot und durchgestrichen~~ ist gekennzeichnet, wenn eine Funktion für die entsprechende Datenbank nicht verfügbar ist.

Tabellenerstellung

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

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

Hinweis

Ein normaler SQL-Befehl endet in der Regel mit einem Semikolon. Bei der internen HSQLDB können so mehrere SQL-Befehle wie der obige **CREATE TABLE** – Befehl direkt hintereinander in **Extras → SQL** eingegeben und dann ausgeführt werden.

Firebird kennt zwar auch die Beendigung mit dem Semikolon, hat damit aber wegen der Prozeduren Probleme. Dort muss für die Ausführung mehrerer Befehle der folgende Code verwendet werden:

```
001 EXECUTE BLOCK AS
002 BEGIN
003    CREATE TABLE "Person" (…);
004    CREATE TABLE "Ort" (…);
005    CREATE TABLE "Beruf" (…);
006 END
```

So können auch mit Firebird mehrere Kommandos direkt eingegeben und auf einmal ausgeführt werden. Dies gelingt aber nicht sicher mit allen SQL-Kommandos.

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.

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

[MEMORY | CACHED | [GLOBAL] TEMPORARY | TEMP | TEXT]: (Hsqldb**, ~~Firebird~~)**

Die Standardeinstellung ist hier **CACHED**. Die HSQLDB erstellt also grundsätzliche alle Tabellen in einem Zwischenspeicher auf dem Datenträger. Dies gilt auch für die Tabellen, die über LibreOffice Base in der internen Datenbank geschrieben werden, selbst wenn der Zusatz **CACHED** nicht angegeben wird. Eine andere Möglichkeit wäre, die Tabellen im Arbeitsspeicher erstellen zu lassen (**MEMORY**).

Hinweis

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

Eine Texttabelle wird in der HSQLDB erstellt. Sie muss jetzt mit einer externen Textdatei (z. B. einer *.csv-Datei) verbunden werden:

```
001 SET TABLE "Text" SOURCE "Text.csv";
```

Die Datei "Text.csv" muss jetzt natürlich die entsprechenden Felder in der entsprechenden Reihenfolge enthalten. Es können bei der Erstellung der Verbindung zusätzliche Optionen gewählt werden. Details hierzu: *http://www.hsqldb.org/doc/1.8/guide/guide.html#set_table_source-section*

Texttabellen sind nicht gegenüber anderen Programmen schreibgeschützt. Es kann also passieren, dass ein anderes Programm/ein anderer Nutzer gerade die Tabelle ändert, während Base darauf zugreift. Texttabellen taugen in der Hauptsache zum Datenaustausch zwischen verschiedenen Programmen.

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 anzeigbar, es sei denn die automatische Löschung des Inhaltes nach dem endgültigen Abspeichern ist ausgeschaltet. Eine Abfrage ergibt hier eine Tabelle ohne Inhalt.

GLOBAL TEMPORARY ist hier die einzige Zusatzoption, die auch Firebird bietet. Die Tabelle wird in der grafischen Benutzeroberfläche zwar angezeigt, aber Eingaben auch direkt über SQL sind nicht abrufbar. Firebird lässt außerdem auch noch eine Definition als **EXTERNAL FILE** (direkt nach dem Tabellennamen) zu. Die Tabelle wird erstellt, ist aber für die Nutzung durch den Server gesperrt.

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

<Felddefinition>:

```
001 "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.

Hinweis

Falls nicht bereits, wie vorher erwähnt, eine *Sortierung von Groß- und Kleinschreibung und auch Umlauten* für Firebird eingestellt wurde, sollte das **vor der Erstellung von Tabellen** jetzt durchgeführt werden.

Es ist auch möglich, für jedes Feld in einer Tabelle extra die entsprechende **Collation** mit anzugeben:

```
001 CREATE TABLE "Tabellenname" (
002    "ID" INTEGER NOT NULL PRIMARY KEY,
003    "Name" VARCHAR(50) COLLATE UNICODE
004  )
```

Dadurch wird die Sortierung des Feldes an die normale Sortierung mit Umlauten und Sonderzeichen angepasst. Andernfalls funktionieren sämtliche Sortierfunktionen bei der Verwendung von Umlauten nicht erwartungsgemäß.

Leider ist diese Form der Sortierung nach der Tabellenerstellung nicht änderbar.

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	-	**CURDATE**, **TODAY** (HSQLDB), **CURRENT_DATE** (HSQLDB, FIREBIRD), **DATE 'NOW'** (FIREBIRD)
für die aktuelle Zeit	-	**CURTIME**, **NOW** (HSQLDB), **CURRENT_TIME** (HSQLDB, FIREBIRD), **TIME 'NOW'** (FIREBIRD)
für den aktuellen Datums-Zeit-Wert	-	**NOW** (HSQLDB), **CURRENT_TIMESTAMP** (HSQLDB, FIREBIRD), **TIMESTAMP 'NOW'** (FIREBIRD).

Zusätzlich bietet Firebird: **DATE 'TODAY'**, **DATE 'YESTERDAY'** und **DATE 'TOMORROW'**.

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.

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.

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

Beispiel:

```
001 CREATE TABLE "Test" (
002 "ID" INT GENERATED BY DEFAULT AS IDENTITY (START WITH 10) PRIMARY KEY,
003 "Name" VARCHAR(50) NOT NULL,
004 "Datum" DATE DEFAULT CURRENT_DATE
005 );
```

Eine Tabelle "Test" wird erstellt. Das Schlüsselfeld "ID" wird als Autowert definiert. Der Autowert soll mit der Zahl 10 beginnen. Für FIREBIRD ist zusätzlich notwendig, dass **PRIMARY KEY** zusätzlich zur der Definition des generierten Wertes erwähnt wird. Sonst fehlt hier der Primärschlüssel und eine Eingabe ist nur über SQL möglich.
Das Eingabefeld "Name" ist ein Textfeld für maximal 50 Zeichen. Es darf nicht leer sein.
Zuletzt kommt ein Datumsfeld "Datum", das als Standardwert das aktuelle Datum speichert, wenn nicht ein anderes Datum eingegeben wurde. Dieser Standardwert wird aber nur bei der Erstellung eines neuen Datensatzes wirksam. Wird ein Datum gelöscht, so bleibt der Inhalt anschließend leer.

<Bedingungsdefinition>:

```
001 [CONSTRAINT "Name"]
002 UNIQUE ( "Feldname 1" [,"Feldname 2"...] ) |
```

```
003 PRIMARY KEY ( "Feldname 1" [,"Feldname 2"...] ) |
004 FOREIGN KEY ( "Feldname 1" [,"Feldname 2"...] ) REFERENCES "anderer
    Tabellenname" ( "Feldname 1" [,"Feldname 2"...]) [ON {DELETE | UPDATE}
005 {CASCADE | SET DEFAULT | SET NULL}] |
006 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 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. Die Bedingungen sind unter «WHERE SQL-Expression» im Kapitel «Abfragen» aufgelistet.

```
001 CREATE TABLE "Zeitmessung" (
002 "ID" INT PRIMARY KEY,
003 "Startzeit" TIME,
004 "Zielzeit" TIME,
005 CHECK ("Startzeit" <= "Zielzeit"));
```

Mit der CHECK-Bedingung wird ausgeschlossen, dass eine "Zielzeit" eingegeben wird, die kleiner als die "Startzeit" ist. Wird dies versucht, so erscheint eine englischsprachige Fehlermeldung, die ungefähr folgendermaßen aussieht:

Check constraint violation SYS_CT_357 table: Zeitmessung …

Der Suchbedingung wird hier gleich ein Name zugewiesen, der allerdings nicht sehr aussagekräftig ist. Stattdessen bietet es sich an, direkt bei der Tabellendefinition den Namen zu definieren:

```
001 CREATE TABLE "Zeitmessung" (
002 "ID" INT PRIMARY KEY,
003 "Startzeit" TIME,
004 "Zielzeit" TIME,
005 CONSTRAINT "Startzeit<=Zielzeit" CHECK ("Startzeit" <= "Zielzeit"));
```

Damit wird die Fehlermeldung etwas klarer. Der Name der Bedingung drückt dann wenigstens aus, worauf sie sich bezieht.

Mit Constraints kann auch der Bereich für ein Feld eingegrenzt werden:

```
001 CREATE TABLE "Monat" (
002 "Monat" TINYINT PRIMARY KEY,
003 CONSTRAINT "Monatszahl" CHECK ("Monat" BETWEEN 1 AND 12));
```

Hier wird das TinyInteger-Feld auf die Zahlen von 1 bis 12 begrenzt.

Mit Constraints wird vor allem gearbeitet, wenn die Beziehung zwischen Tabellen oder der Index für bestimmte Felder festgelegt werden soll. Die Constraints werden, bis auf die CHECK-Bedingung, in der GUI unter **Extras → Beziehungen** und als Indexentwurf in dem Tabellenentwurf unter **Extras → Indexentwurf** festgelegt.

[ON COMMIT {DELETE | PRESERVE} ROWS]: (HSQLDB)

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:

```
001 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". (HSQLDB)

In FIREBIRD muss der Schritt der Zuordnung nach dem Einfügen des Feldes erfolgen. Dafür ist die Position universell eben auch für alte Felder wählbar:

```
001 ALTER TABLE "Adresse" ADD "Vorname" VARCHAR(25);
002 ALTER TABLE "Adresse" ALTER "Vorname" POSITION 2;
```

POSITION: Ändere die Position des Feldes zu der entsprechenden Stelle, hier als 2. Feld. Wird die Zahl kleiner 1 gewählt, so erscheint eine Fehlermeldung. Wird die Zahl größer als die Zahl der Felder gewählt, so wird das Feld als letztes Feld gesetzt.

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

```
001 ALTER TABLE "Tabellenname" ADD <Felddefinition> [BEFORE
    "bereits_existierender_Feldname"];
002 ALTER TABLE "Tabellenname" DROP "Feldname";
```

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

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

Dies ändert den Namen eines Feldes. **RENAME TO** ist bei der HSQLDB erforderlich. Bei FIREBIRD funktioniert die Namensänderung nur mit **TO**.

```
001 ALTER TABLE "Tabellenname" ALTER COLUMN "Feldname" SET DEFAULT
    <Standardwert>;
```

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

```
001 ALTER TABLE "Tabelle" ALTER COLUMN "Datum" SET DEFAULT CURRENT_DATE;
```

Dies ändert ein Datumsfeld so, dass das momentane Datum eingefügt wird, wenn des Feld "Datum" leer ist.

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

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

```
001 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 «Datentypen des Tabelleneditors» im 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**).

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

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

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

Dieser Befehl wird bei der HSQLDB 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 Datenbank 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. Der neue Feldwert ist dabei ohne einfache Anführungszeichen als Zahl einzugeben.

Bei FIREBIRD muss der Zugriff anders lauten:

```
001 ALTER TABLE "Tabellenname"
002 ALTER "Feldname" RESTART WITH <letzter_Feldwert>;
```

Hier muss also zwingend der Hinweis **COLUMN** aus dem HSQLDB-Befehl wegfallen. Außerdem ist bei der Nummer die des letzten Feldwertes anzugeben, damit von dem aus die neue Nummer gebildet werden kann.

```
001 ALTER TABLE "Tabellenname"
002 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:

```
001 ALTER TABLE "Ausleihe" ADD CHECK
    (COALESCE("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 …**

```
001 ALTER TABLE "Tabellenname"
002 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.

```
001 ALTER TABLE "Tabellenname"
002 ADD [CONSTRAINT "Bedingungsname"] PRIMARY KEY ("Feldname1",
    "Feldname2" ...);
```

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

```
001 ALTER TABLE "Tabellenname"
002 ADD [CONSTRAINT "Bedingungsname"] FOREIGN KEY ("Feldname1",
    "Feldname2" ...)
003 REFERENCES "Tabellenname_der_anderen_Tabelle" ("Feldname1_andere_Tabelle",
    "Feldname2_andere_Tabelle" ...)
004 [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 Werten 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.

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

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

```
001 ALTER TABLE "Tabellenname" RENAME TO "neuer_Tabellenname"; (HSQLDB)
```

Mit diesem Befehl schließlich wird einfach nur der Name einer Tabelle geändert.
Mit ~~FIREBIRD~~ ist es nicht möglich, einen Tabellennamen zu ändern. Stattdessen muss hier eine neue Tabelle mit neuem Namen und den alten Daten erstellt werden.

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. Die Änderungen werden auch dann angezeigt, wenn im Tabellencontainer **Ansicht → Tabellen aktualisieren** aufgerufen wird.

Tabellen löschen

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

Löscht die Tabelle "Tabellenname".
Firebird kennt hier keine weiteren Optionen. Sobald über Verknüpfungen zu anderen Tabellen oder Ansichten Probleme auftauchen, wird das Löschen mit einer Fehlermeldung abgebrochen.

IF EXISTS schließt aus, dass eine Fehlermeldung erscheint, falls diese Tabelle nicht existiert. (HSQLDB)

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 einer Ansicht (*View*) Bezug genommen wird. Abfragen sind davon nicht berührt, da die innerhalb der HSQLDB nicht gespeichert sind. (HSQLDB)

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 Tabellenansichten (*Views*), in denen auf die entsprechende Tabelle Bezug genommen wird, werden komplett gelöscht. (HSQLDB)

Funktionserweiterung durch Trigger bei Firebird

Die interne HSQLDB erfordert für die Nutzung von weiteren Funktionen separat erstellte Bibliotheken. Dagegen lassen sich automatisch ablaufende Trigger in Firebird direkt nutzen[2]. Dies soll an einem Beispiel zur Absicherung eines Datensatzes gegen eine Löschung verdeutlicht werden. Die Eingabe erfolgen alle über **Extras → SQL**.

```
001 CREATE EXCEPTION EX_NAME_WICHTIG
002 'Löschen des Datensatzes nicht möglich. Inhalt als wichtig vermerkt.';
```

Zuerst wird eine zu erscheinende Fehlermeldung mit eindeutigem Namen definiert. Diese Meldung erscheint in einem Firebird-Dialog, wenn versucht wird, einen Datensatz zu löschen, bei dem das zu dem Datensatz gehörende Ja/Nein-Feld "wichtig" mit 'ja' markiert wurde.

Anschließend wird ein Lösch-Trigger, wiederum mit eindeutigem Namen, definiert. Der Trigger wird jedes Mal ausgeführt, wenn in der Tabelle "Name" ein Datensatz gelöscht werden soll.

```
001 CREATE TRIGGER DELETE_NAME
002 ACTIVE BEFORE DELETE ON "Name"
003 AS
004 BEGIN
005    IF (OLD."wichtig" = TRUE) THEN
006    EXCEPTION EX_NAME_WICHTIG;
007 END;
```

Der Trigger startet vor dem Löschen eines Datensatzes. Wenn in dem aktuellen zu löschenden Datensatz (**OLD.**, Datensatz vor der Änderung) der Inhalt des Feldes "wichtig" markiert ist (**TRUE**), dann erscheint die Ausnahmemeldung und die Löschung wird unterbunden.

In Tabellen ist es möglich, über den **Default**-Wert einen **TIMESTAMP** zu erstellen. Allerdings greift dieser Defaultwert nur, wenn der Datensatz neu erstellt wird und das entsprechende Feld beim Abspeichern NULL ist. So etwas wie die letzte Datensatzänderung kann darüber nicht nachvollzogen werden. So etwas kann über Makros oder über einen Trigger automatisch erledigt werden:

```
001 CREATE OR ALTER TRIGGER BEFORE_IN_UP_NAME FOR "Name"
002 ACTIVE BEFORE INSERT OR UPDATE POSITION 0
003 AS
004 BEGIN
005    NEW."letzte_Aenderung" = CURRENT_TIMESTAMP;
006 END;
```

Dieser Trigger bezieht sich wiederum auf die Tabelle "Name". Wenn ein INSERT oder UPDATE zu der Tabelle aufgerufen wird, dann wird dieser Trigger ausgelöst. Mit der Nummer der POSITION kann die Reihenfolge bestimmt werden, in der Trigger arbeiten. Die Nummerierung beginnt mit '0'. Der Datensatz nach der Änderung (**NEW.**) soll in dem Feld "letzte_Aenderung" den aktuellen Zeitstempel enthalten.

Verknüpfung von Tabellen

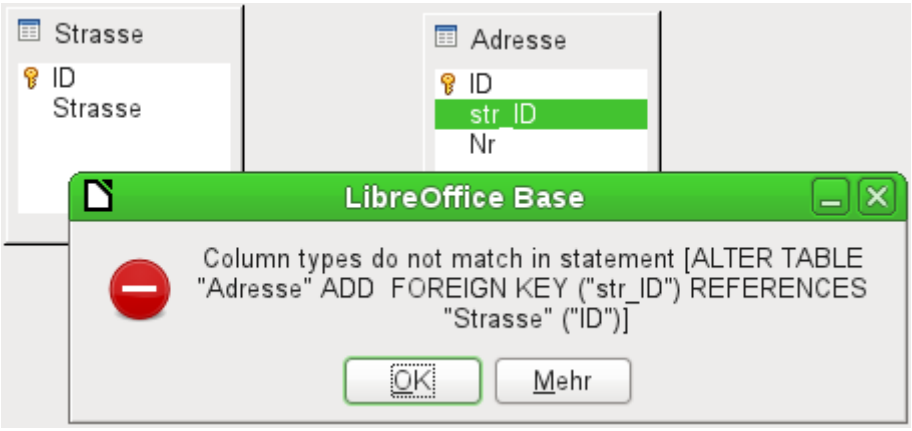
Prinzipiell kommt eine Datenbank auch ohne die Verknüpfung von Tabellen aus. Der Nutzer muss dann bei der Eingabe selbst darauf achten, dass die Beziehungen zwischen den Tabellen stimmig bleiben. In der Regel geschieht dies, indem er sich entsprechende Formulare erstellt, die dies bewerkstelligen sollen.

2 Weiterführende Informationen enthält die Dokumentation zu Firebird: *https://firebirdsql.org/manual/de/*

Das Löschen von Datensätzen bei verknüpften Tabellen ist nicht so einfach möglich. Angenommen es würde aus der Tabelle "Strasse" in *Abbildung 11* 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.

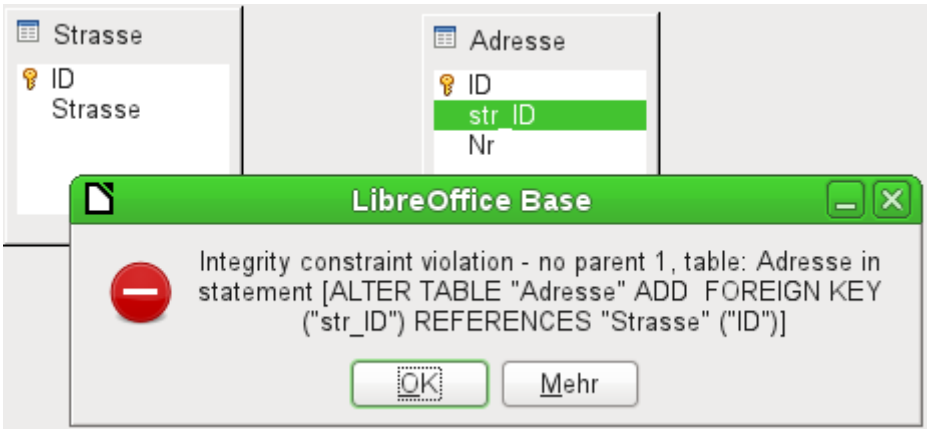
Die grundsätzlichen Verknüpfungen werden über **Extras → Beziehungen** festgelegt. Hierbei wird eine Verbindungslinie von dem Primärschlüssel einer Tabelle zum zu definierenden Sekundärschlüssel gezogen.

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, aus der der Primärschlüssel gestellt wird) nicht vorhanden ist, so würde ein ungültiger Eintrag erzeugt.

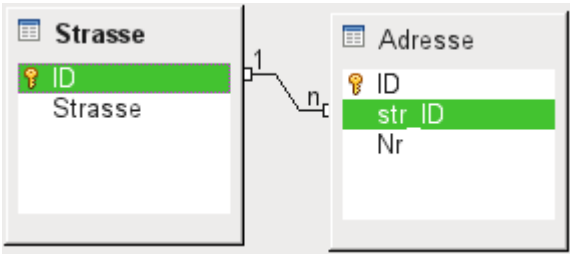
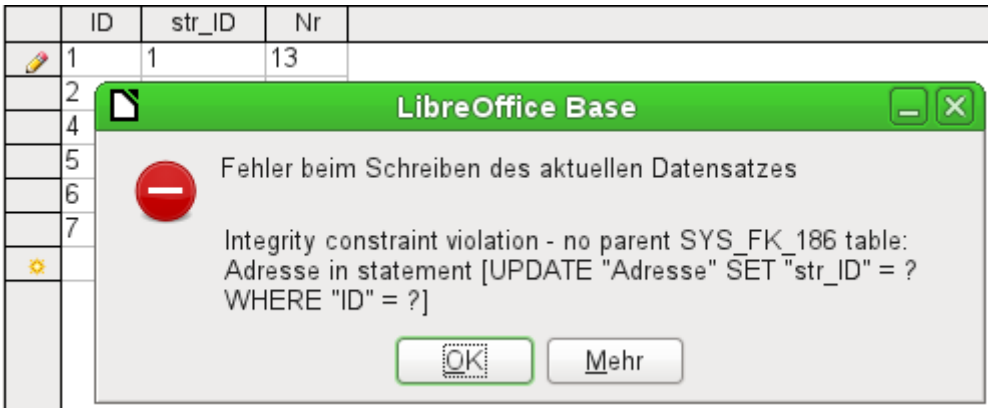


Abbildung 11: Erstellte Verbindung zwischen "Strasse" und "Adresse" mit "str_ID" als Fremdschlüssel.

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.

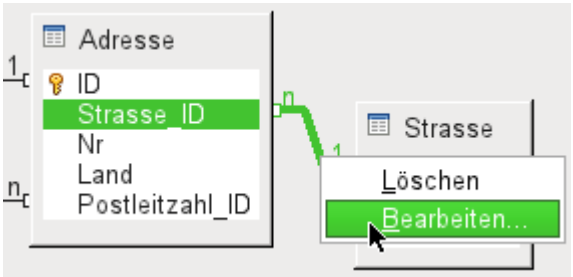


Abbildung 12: Durch Klick mit der rechten Maustaste können Verknüpfungen, hier am Beispiel der Mediendatenbank, bearbeitet werden.

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.

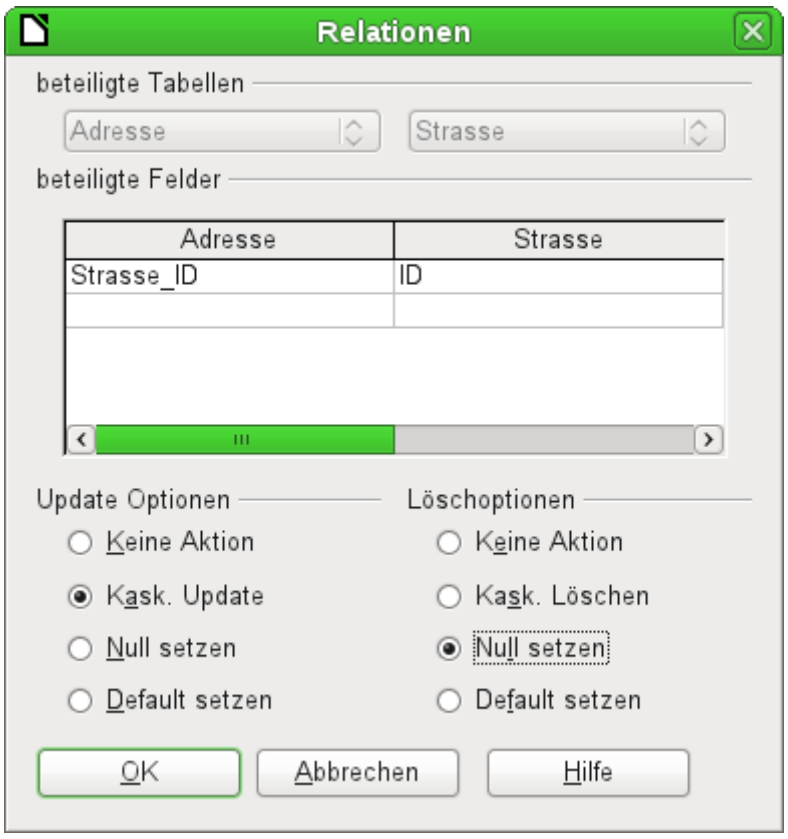


Abbildung 13: Dialog «Relationen»

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 **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. Dieser Standardwert wird nicht durch die grafische Benutzeroberfläche bereit gestellt. Wird per SQL der Default-Wert durch

```
001 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 interne 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 gleich lautenden Update-Option.

Default setzen:
Dieses Verhalten entspricht ebenfalls der gleich lautenden 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.

In **Extras → Beziehungen** können durch Ziehen mit der Maus nur Fremdschlüssel erstellt werden, die sich auf ein Feld einer Tabelle beziehen. Für die Verbindung mit einer Tabelle, die einen zusammengesetzten Primärschlüssel hat, muss eventuell in **Extras → Beziehungen** der Menüpunkt **Einfügen → Neue Relation** oder der entsprechende Button aufgesucht werden. Es erscheint dann der *Dialog «Relationen»* mit der freien Auswahl der beteiligten Tabellen.

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 Fremdschlüssel 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 hoch zählendes 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.

Enthält die Tabelle keinen Primärschlüssel, so erlaubt Base bei den meisten Datenbankverbindungen auch keine Eingabe. Auch muss **Bearbeiten → Daten bearbeiten** eingeschaltet sein. Diese Einstellung ist allerdings standardmäßig eingeschaltet und wird auch nicht gespeichert. Es kann also beim Aufruf von Tabellen nicht zu einer nicht möglichen Eingabe führen.

Abbildung 14: Eingabe in Tabellen - Spalten ausblenden.

Abbildung 15: 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.

Hinweis	Eine in der Tabelle ausgeblendete Spalte ist auch in Abfragen nicht mehr sichtbar, die über die GUI erstellt werden. Wird die Spalte dort also benötigt, so muss die Abfrage in direktem SQL-Code gestellt werden. Dann ist die Eingabe von Daten in die Abfrage aber nicht mehr möglich.

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 bei der HSQLDB nicht nötig und nicht möglich. Die Daten sind bereits in der Datenbank gelandet, bei der HSQLDB also im Arbeitsspeicher. Sie wer-

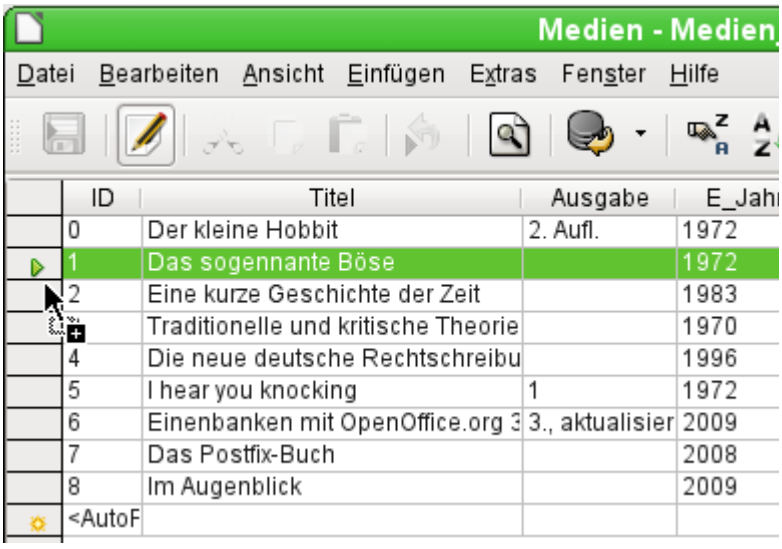
den (aus Sicht der Datensicherheit leider) erst auf der Festplatte abgespeichert, wenn Base geschlossen wird. Wenn Base also aus irgendwelchen Gründen nicht ordnungsgemäß beendet werden kann, kann dies immer noch zu Datenverlusten führen. Bei FIREBIRD hingegen muss die Datenbank ausdrücklich vor dem Schließen abgespeichert werden.

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.

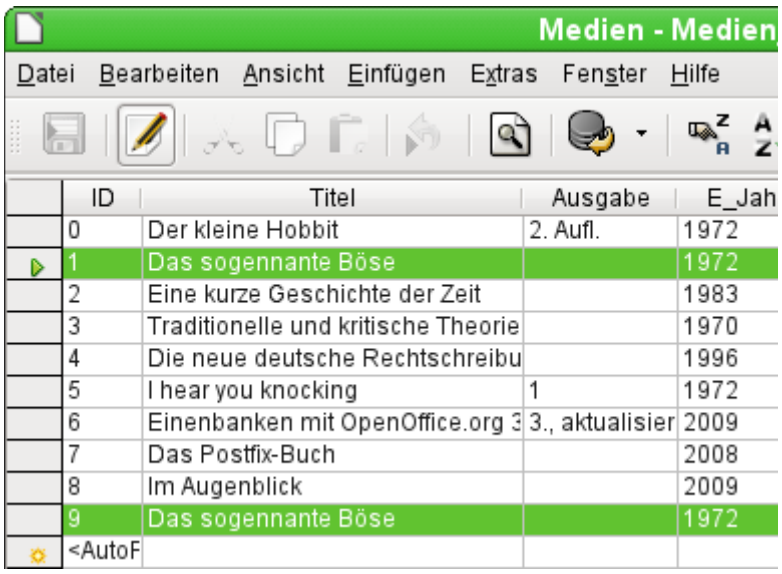
Die Änderung eines Datensatzes ist einfach möglich: Feld aufsuchen, geänderten Wert eingeben und Datenzeile wieder verlassen.

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

Recht gut versteckt gibt es auch die Möglichkeit, ganze Zeilen zu kopieren. Hierzu muss allerdings der Primärschlüssel der Tabelle als «AutoWert» definiert sein.



Der Zeilenkopf wird mit der linken Maustaste markiert. Die Maustaste wird gehalten. Das Cursor-Symbol verwandelt sich in ein Symbol mit einem Plus-Zeichen, das andeutet, dass etwas hinzugefügt wird. Sobald dieses Symbol erscheint, kann die Maustaste losgelassen werden.



ID	Titel	Ausgabe	E_Jah
0	Der kleine Hobbit	2. Aufl.	1972
1	Das sogennante Böse		1972
2	Eine kurze Geschichte der Zeit		1983
3	Traditionelle und kritische Theorie		1970
4	Die neue deutsche Rechtschreibu		1996
5	I hear you knocking	1	1972
6	Einenbanken mit OpenOffice.org 3	3., aktualisier	2009
7	Das Postfix-Buch		2008
8	Im Augenblick		2009
9	Das sogennante Böse		1972
<AutoF			

Der Datensatz mit dem Primärschlüsselwert '1' wird als neuer Datensatz mit dem neuen Primärschlüsselwert '9' eingefügt.

Wird nicht nur ein Datensatz markiert, sondern unter Zuhilfenahme der *Shift*- oder der *Strg*-Taste gleich mehrere Datensätze, so können auch mehrere Datensätze direkt kopiert werden. Das funktioniert auch zwischen Tabellen, sofern die Felder den entsprechenden Inhalt auch speichern können. Die Funktion fügt lediglich die Spalten ein, die einen übereinstimmenden Namen aufweisen. Andere markierte Spalten bleiben unberücksichtigt. Nur die aufnehmende Tabelle muss dazu einen eindeutigen Primärschlüssel erzeugen können (AutoWert).

Tipp	Die Spaltenköpfe können auf die für die Eingabe erforderliche Breite gezogen werden. Wird dies in der Tabelle gemacht, so wird die Speicherfunktion von Base aktiviert. Base speichert diese Spaltenbreiten aus den Tabellen. Die Spaltenbreiten der Tabellen beeinflussen die Spaltenbreiten in Abfragen. Sind Spalten in Abfragen zu klein, so hilft dort nur vorübergehend eine Breitenänderung. Die Einstellung der Abfrage wird nicht gespeichert. Hier hilft dann eine Breitenänderung in der Tabelle, sofern es sich in der Abfrage nicht um Felder handelt, die durch irgendwelche Funktionen beeinflusst wurden.

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

Sortieren von Tabellen

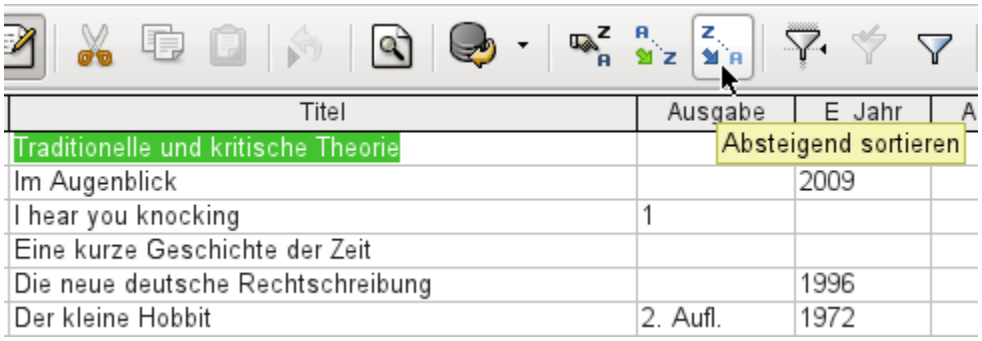


Abbildung 16: Schnelle Sortierung

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:

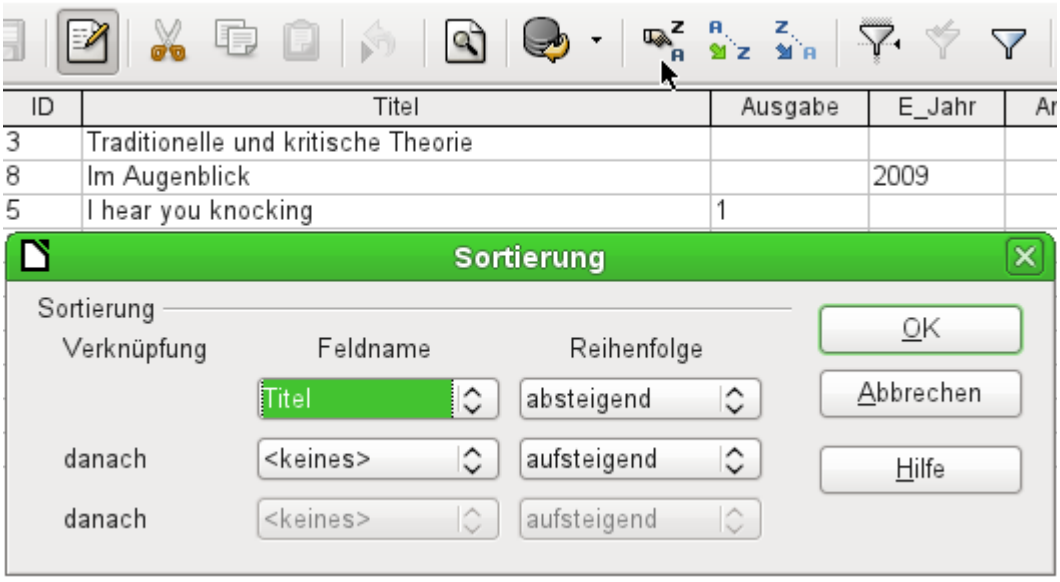
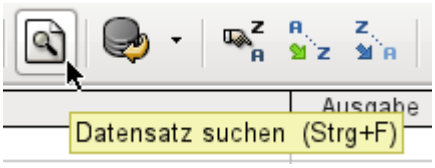


Abbildung 17: Sortierung nach mehreren Spalten

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 etwas kompliziert und für durch Suchmaschinen verwöhnte Nutzer nicht gerade die erste Wahl, um einen bestimmten Datensatz zu finden. Außerdem ist die Suchfunktion bei größeren Datenmengen äußerst langsam, da die Suche nicht mit SQL-Befehlen innerhalb der Datenbank erfolgt. Für eine schnellere Suche bietet sich stattdessen an, statt der Tabelle eine entsprechende Abfrage zu nutzen. Diese Abfrage kann dann mit Hilfe von Parametern gestartet und durchsucht werden. Siehe dazu: «Verwendung von Parametern in Abfragen».

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.

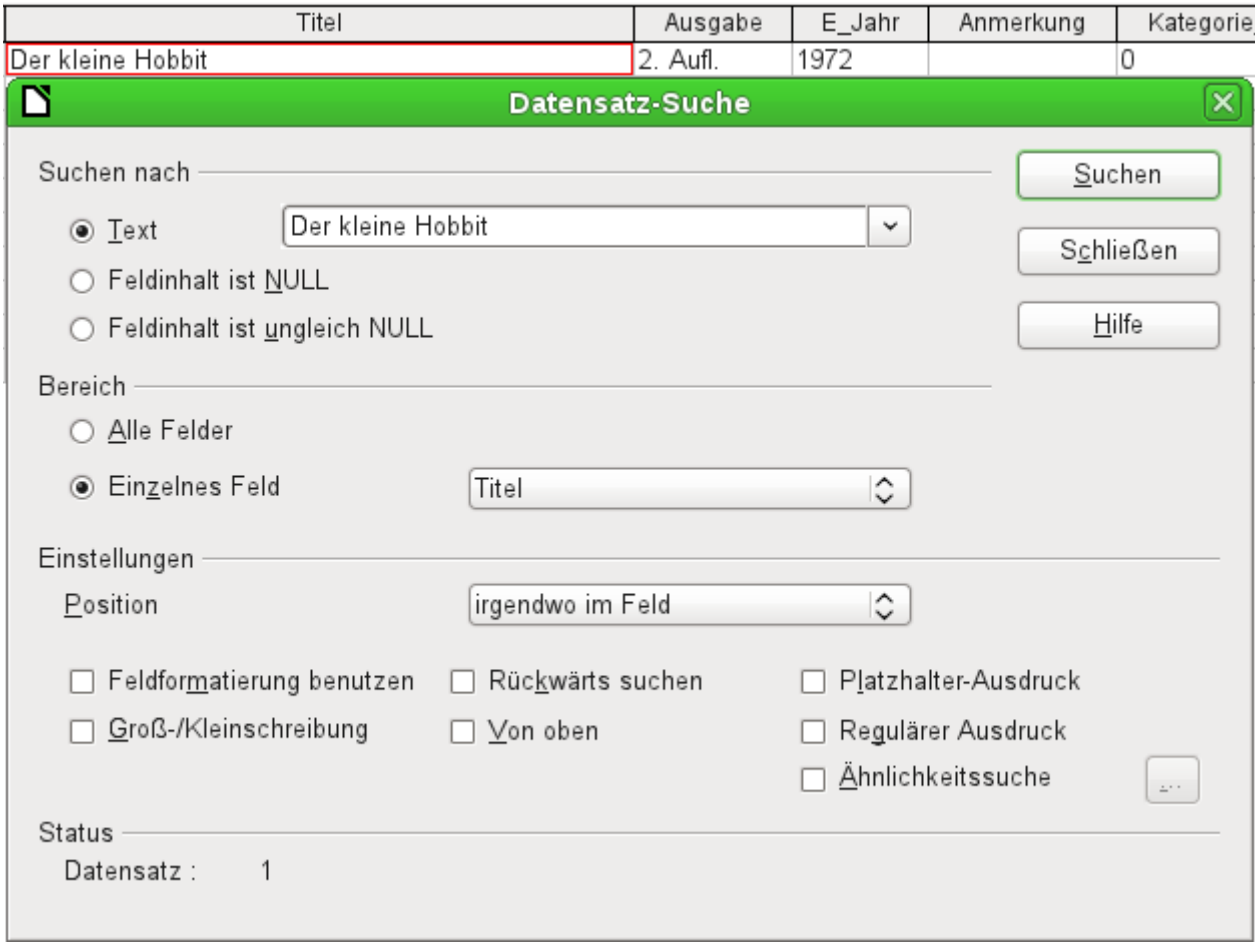


Abbildung 18: Eingabemaske zur Datensatzsuche

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 LO 7.4 auf einer leeren Seite.

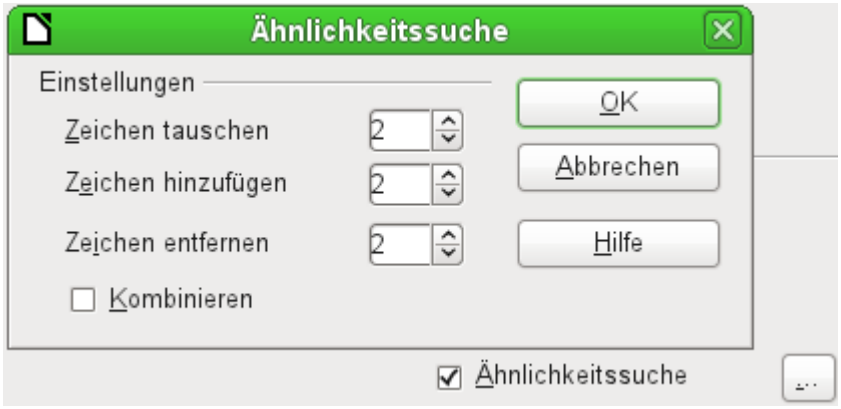


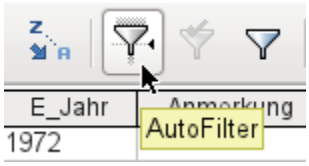
Abbildung 19: Eingrenzung der Ähnlichkeitssuche

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.

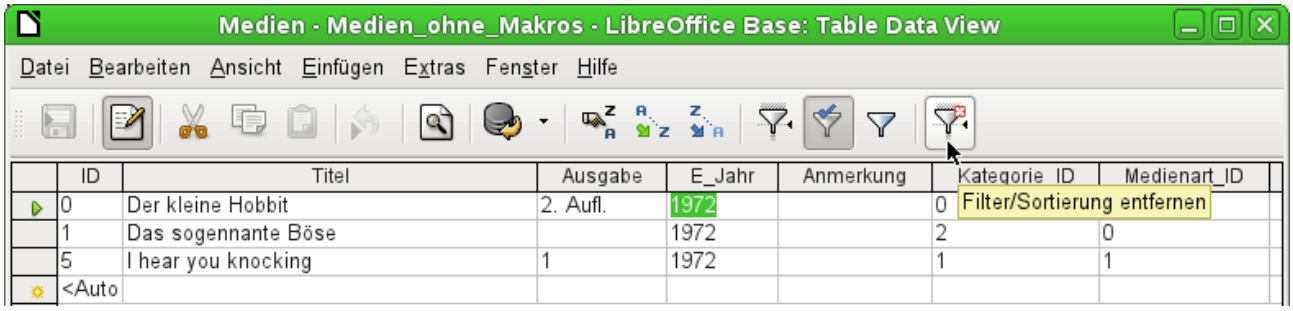
Insgesamt ist dieses Suchmodul eher etwas für Leute, die durch häufige Anwendung genau wissen, an welchen Stellen sie zum Erreichen eines Ziels drehen müssen. Für den Normaluser dürfte die Möglichkeit, Datensätze durch Filter zu finden, schneller zum Ziel führen.

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



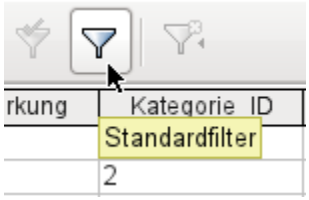
Die schnelle Filterung läuft über den AutoFilter. Der Cursor wird in ein Feld gesetzt, der Filter übernimmt nach einem Klick auf den Button diesen Feldinhalt. Es werden nur noch die Datensätze angezeigt, die dem Inhalt des gewählten Feldes entsprechen. Die folgende Abbildung zeigt die Filterung nach 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 20: 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 der GUI auch «*»; wird intern umgewandelt). 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

```
001 INSERT INTO "Tabellenname" [( "Feldname" [,...] )]
002 { 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.

```
001 INSERT INTO "Tabellenname" ("Feldname") VALUES ('Test');
002 CALL IDENTITY(); (HSQLDB, ~~FIREBIRD~~)
```

In die Tabelle wird in der Spalte "Name" der Wert 'Test' eingegeben. Das automatisch hoch zählende 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. Bei Firebird sollte **`RETURNING "ID"`** direkt mit dem Befehl angegeben werden, gibt aber zur Zeit (LO 7.4) keinen Wert für das Schlüsselfeld zurück.

Hinweis

Bei FIREBIRD ist es zur Zeit notwendig, die neu erstellte ID auf einem anderen Weg zu bestimmen, solange der **`RETURNING "ID"`** nicht funktioniert. Für Einzelbenutzer würde es ausreichen, die höchste ID nach dem Einfügen eines Datensatzes abzufragen. Bei mehreren Benutzern kann das aber schnell dazu führen, dass eben die ID eines anderen eingefügten Datensatzes ermittelt wird. deswegen hier ein komplizierterer Weg, bei dem die Abfragen nur im direkten SQL funktionieren.

```
001 SELECT RDB$FIELD_NAME, RDB$RELATION_NAME, RDB$GENERATOR_NAME
    FROM RDB$RELATION_FIELDS WHERE RDB$GENERATOR_NAME IS NOT
    NULL;
```

Zuerst muss der passende Generator für die entsprechende Tabelle ermittelt werden: **`RDB$GENERATOR_NAME`**

```
001 SELECT NEXT VALUE FOR RDB$1 FROM RDB$DATABASE;
```

Anschließend muss für diesen Generator die nächste freie "ID" geholt werden. Im obigen Code ist das der Generator **`RDB$1`**. Wird die "ID" ermittelt, dann ist sie für den Generator vergeben.

```
001 INSERT INTO "Tabellenname" ("ID", "Feldname") VALUES (<Wert
    aus vorheriger Abfrage>, 'Test');
```

Anschließend wird der ermittelte Wert für die "ID" in den kommenden INSERT-Befehl eingefügt. Sinnvoll nutzbar ist dies nur bei Makros.

Soll der neu vergebene Primärschlüssel im nächsten Schritt direkt wieder genutzt werden, so kann **`IDENTITY()`** direkt verwendet werden: HSQLDB, nicht ~~FIREBIRD~~

```
001 INSERT INTO "Ort" ("Ort") VALUES ('Buxtehude');
```

```
002 INSERT INTO "Person" ("Name", "Ort_ID") VALUES ('Hein', IDENTITY());
```

Bereits mit der zweiten, direkt folgenden Abfrage, werden 2 Werte auf einmal in die entsprechende Tabelle eingegeben.

```
001 INSERT INTO "Tabelle" ("Vorname","Nachname") VALUES ('Eva','Müller');
```

Hiermit werden die Werte für 2 Felder auf einmal eingegeben. Die Reihenfolge der Felder und die Reihenfolge der entsprechenden Werte muss dabei gleich sein. **Text** ist bei den Werten in **einfache Anführungszeichen** zu setzen: **'Text'**. Enthält der Text selbst einfache Anführungszeichen, so werden die einfachen Anführungszeichen wiederum durch einfache Anführungszeichen maskiert: **'Robert''s Datenbank'**.

```
001 INSERT INTO "Tabelle" ("Ware","Preis") VALUES ('Kaffee', 8.79);
```

Zahlenwerte können **ohne einfache Anführungszeichen** eingegeben werden. Werte mit Nachkommastellen werden nicht mit einem Komma, sondern dem Dezimalpunkt eingegeben.

```
001 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

```
001 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:

```
001 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:

```
001 UPDATE "Tabellenname" SET "Jahrgang" = "Jahrgang"+1 WHERE "Wiederholung" =
    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**.

Soll nachträglich für ein Feld bei allen leeren Feldern ein Standardwert eingetragen werden, so geht dies mit

```
001 UPDATE "Tabelle" SET "Feld" = 1 WHERE "Feld" IS NULL
```

Die Änderung mehrerer Felder erfolgt mit der direkten Zuordnung von Feld zu Wert. Angenommen es wären in einer Tabelle mit Büchern die Autoren direkt mit eingetragen. Jetzt ist aufgefallen, dass statt «Erich Kästner» häufig «Eric Käschtner» eingetragen worden ist.

```
001 UPDATE "Bücher"
002 SET "Autor_Vorname" = 'Erich', "Autor_Nachname" = 'Kästner'
003 WHERE "Autor_Vorname" = 'Eric' AND "Autor_Nachname" = 'Käschtner'
```

Auch 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:

```
001 UPDATE "Tabellenname"
002 SET "Preis" = "Preis"*0.9
003 WHERE "Preis" >= 150
```

Mit der Wahl des Datentyps 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

```
001 UPDATE "Tabellenname"
002 SET "Feldname" = TRIM(TRAILING FROM "Feldname")
```

Bestehende Datensätze löschen

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

Ohne einen eingrenzenden Bedingungsausdruck wird durch

```
001 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:

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

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

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

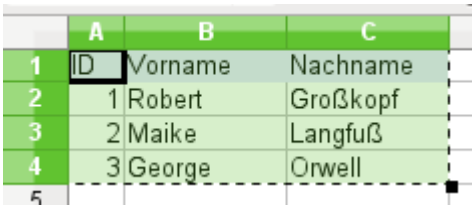
oder alternativ mit

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

Import von Daten aus anderen Datenquellen

Manchmal existieren schon komplette Datensammlungen in einem anderen Programm, die über die Zwischenablage in Base importiert werden sollen. Hier gibt es die Möglichkeit, beim Import eine neue Tabelle erstellen zu lassen oder auch Datensätze an eine bereits bestehende Tabelle anzufügen.

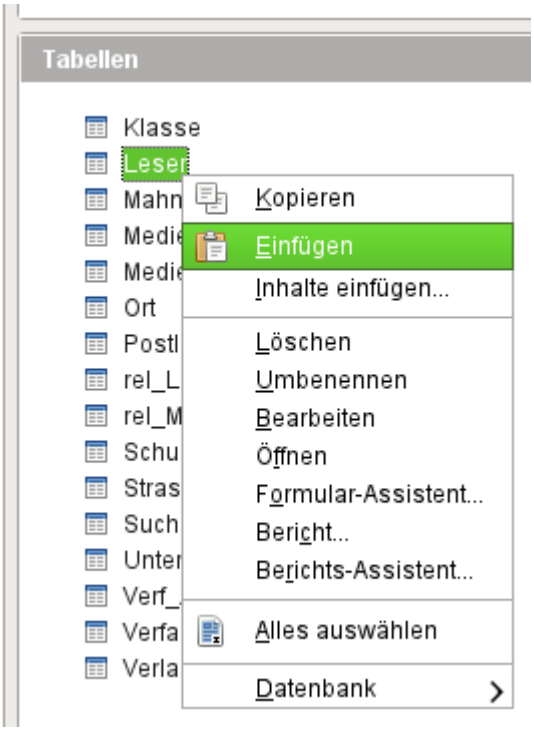
Hinweis	Um einen Import über die Zwischenablage zu gewährleisten, muss das Datenformat für Base lesbar sein. Dies ist bei allen Daten der Fall, die in LibreOffice geöffnet zur Verfügung stehen. Sollen also z. B. Tabellen aus einer externen Datenbank in eine *.odb-Datei eingelesen werden, so muss zuerst zu der externen Datenbank über LibreOffice ein Kontakt hergestellt werden. Diese Datenbank muss außerdem geöffnet oder als Datenquelle in LibreOffice angemeldet sein. Siehe hierzu auch das Kapitel «Zugriff auf externe Datenbanken».



	A	B	C
1	ID	Vorname	Nachname
2	1	Robert	Großkopf
3	2	Maike	Langfuß
4	3	George	Orwell
5			

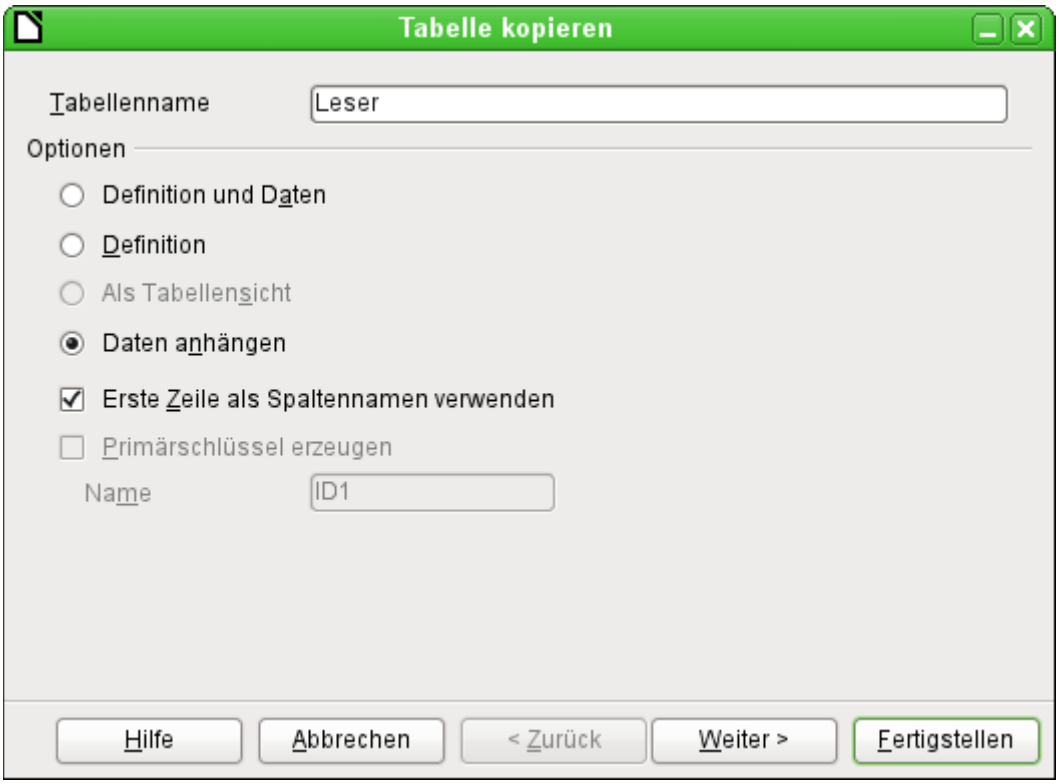
Eine kleine Beispieltabelle wird aus der Tabellenkalkulation «Calc» in die Zwischenablage kopiert.

Anschließend wird nach Base in den Tabellencontainer gewechselt. Dies kann natürlich auch über Markierung mit der linken Maustaste und anschließendem Ziehen bei gleichzeitig gedrückter Maustaste geschehen.

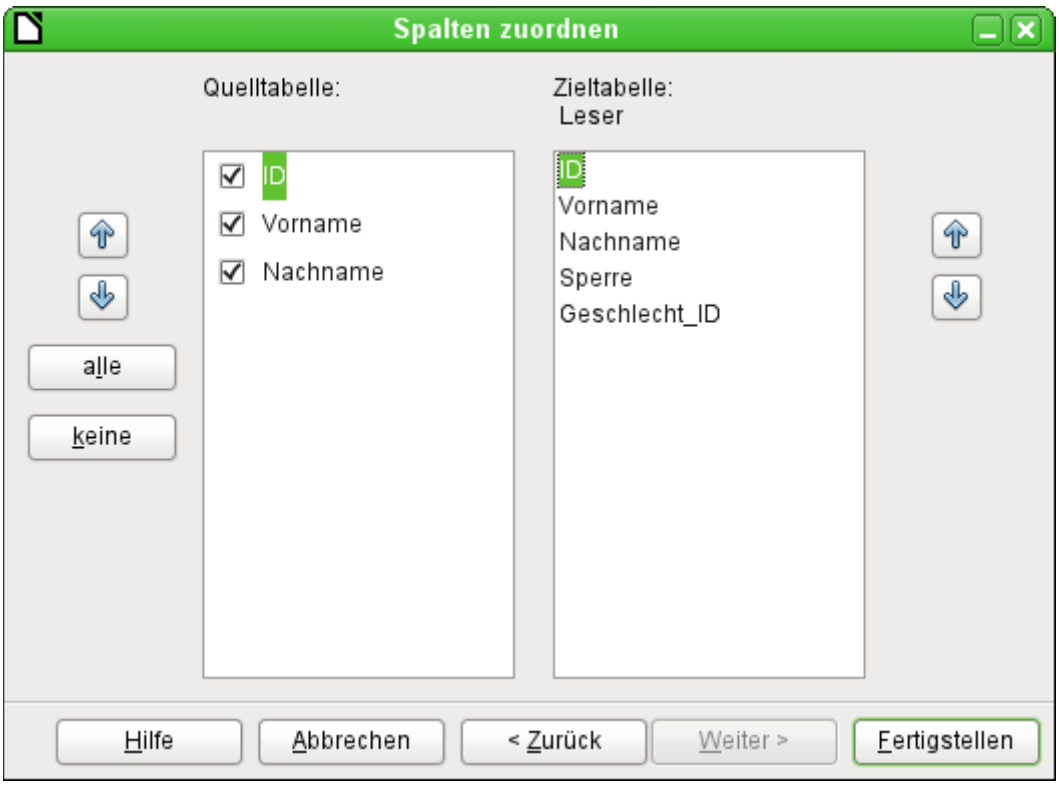


In dem Tabellencontainer wird mit der rechten Maustaste das Kontextmenü über der Tabelle aufgerufen, an die die Daten angehängt werden sollen und **Einfügen** gewählt

Importierte Daten an bestehende Daten einer Tabelle anfügen



Der Tabellenname erscheint im Importassistenten. Gleichzeitig ist **Daten anhängen** vorgewählt. **Erste Zeile als Spaltennamen verwenden** muss je nach LO-Version und Ausgangstabelle noch markiert werden. Wenn die Daten angehängt werden sollen, dann wird natürlich keine Definition der Daten übernommen. Ein Primärschlüssel muss auch bereits vorhanden sein.



Die Spalten der Quelltabelle aus Calc und der Zieltabelle in Base müssen nicht von der Reihenfolge her, vom Namen her oder von der Anzahl her übereinstimmen. Es werden nur die auf der linken Seite ausgewählten Elemente übertragen. Die Zuordnung zwischen Quelltabelle und Zieltabelle muss mit den danebenliegenden Pfeiltasten vorgenommen werden. Felder, die in der Zieltabelle in der gleichen Zeile liegen, erhalten die Daten aus der entsprechenden Zeile der Quelltabelle. Die Position in der jeweiligen Spalte ist also entscheidend, unabhängig davon, ob ein Element in der Quelltabelle markiert ist oder nicht.

Danach wird der Import vollzogen.

Der Import kann Probleme bereiten, wenn

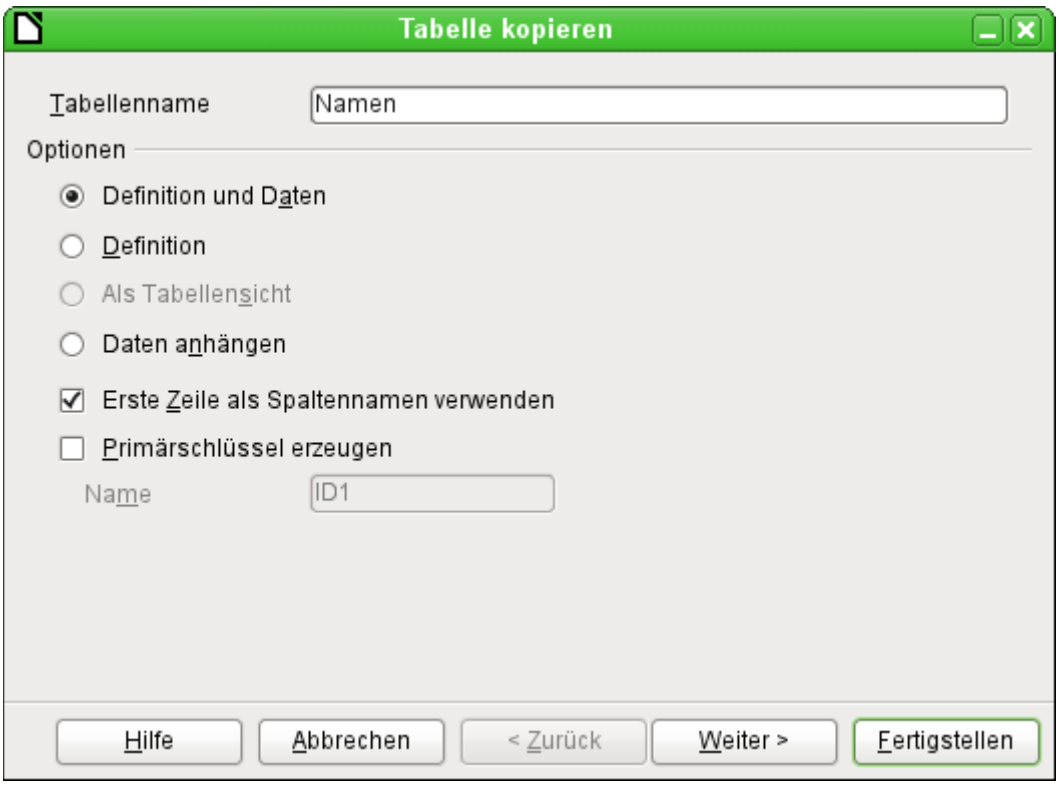
- Felder der Zieltabelle eine Eingabe erfordern, in der Quelltabelle hierfür aber keine Daten vorliegen,
- Felddefinitionen der Zieltabelle mit den Daten der Quelltabelle nicht vereinbar sind (wenn z. B. der Vorname in ein Zahlenfeld geschrieben werden soll oder das Feld der Zieltabelle zu wenig Zeichen zulässt) oder
- die Quelltabelle Daten vorgibt, die mit der Zieltabelle nicht vereinbar sind, wie z. B. eindeutige Werte beim Primärschlüssel oder in anderen Feldern, die so definiert wurden.

Tipp

Manchmal enthalten die zu importierenden Daten keinen Primärschlüssel. Der Primärschlüssel wird dann beim Import automatisch erzeugt. Das hat dann bei häufigerem Import oft den Nachteil, dass der Primärschlüssel bei der Zieltabelle, wie im obigen Screenshot "ID", ganz oben steht. Die Position bei der Zieltabelle hängt nämlich direkt mit der Position des Feldes in der Tabelle zusammen. Deshalb muss vor dem Import immer das Feld in der Zieltabelle erst einmal nach unten bewegt werden.

Wird stattdessen in der Quelltabelle eine Spalte "ID" eingefügt, die gar keinen Inhalt hat, so wird das beim Import als **NULL** interpretiert. Dadurch wird dann der automatisch generierte Primärschlüssel eingesetzt. So kann eine umständliche Sortierung verhindert werden.

Neue Tabelle beim Import erstellen



Zum Start des Importassistenten erscheint automatisch der vorher markierte Tabellenname. Dieser Tabellenname muss für die Gründung einer neuen Tabelle beim Import erst einmal geändert werden, da eine Tabelle mit gleicher Bezeichnung wie eine bereits bestehende Tabelle nicht existieren darf. Der Tabellenname ist "Namen". **Definition und Daten** sollen übernommen werden. Die erste Zeile enthält die Spaltennamen.

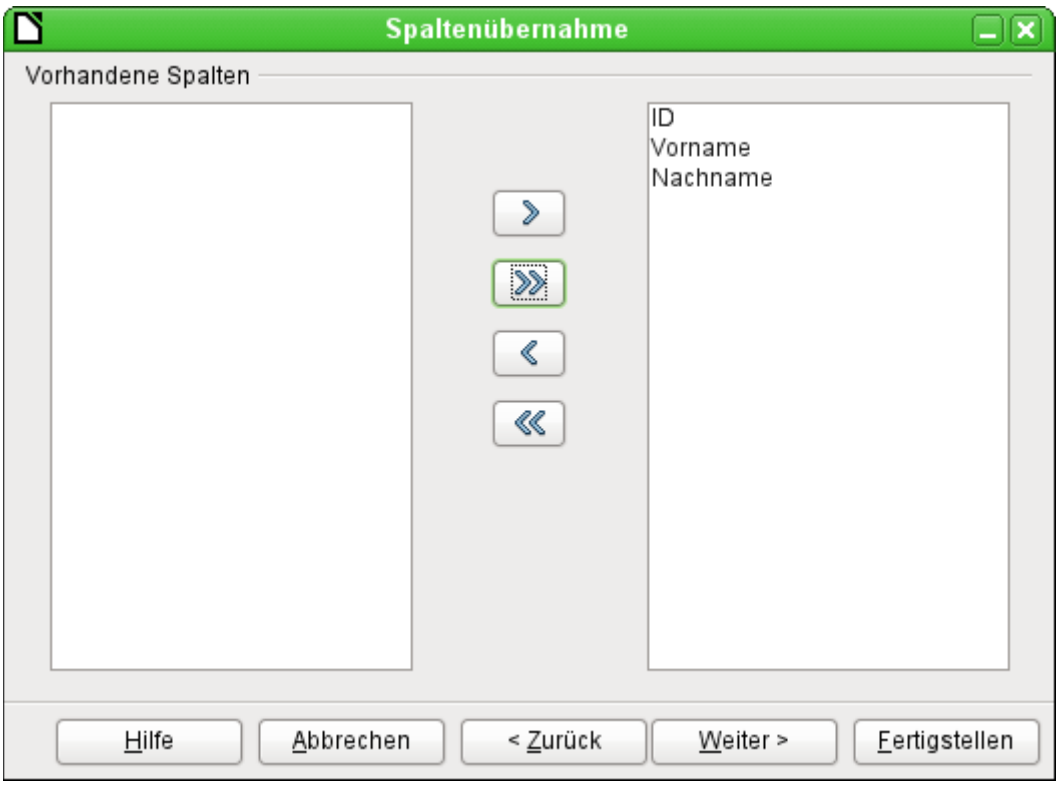
An dieser Stelle kann ein neues, zusätzliches Datenbankfeld für einen Primärschlüssel erzeugt werden. Der Name dieses Datenbankfeldes darf nicht schon als Spalte in der Calc-Tabelle existieren. Anderenfalls erscheint die Fehlermeldung:

'Es sind bereits folgende Felder als Primärschlüssel gesetzt : ID'.

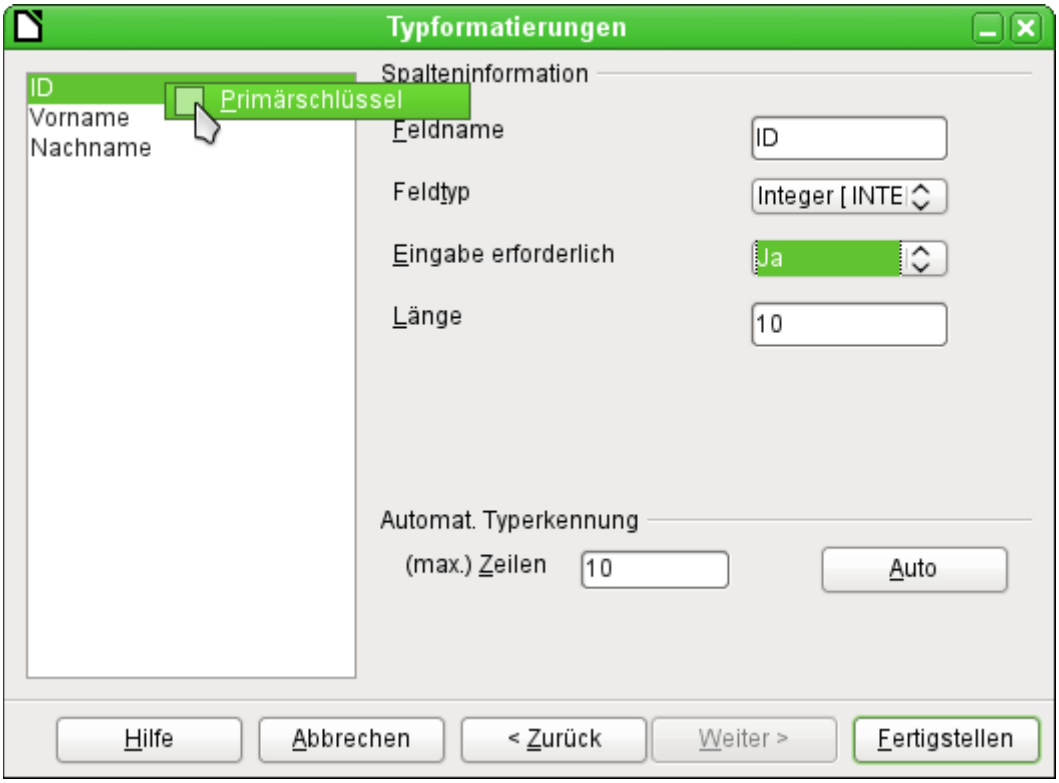
Diese Meldung gibt den Sachverhalt leider nicht ganz korrekt wieder.

Soll ein bereits vorhandenes Feld als Schlüsselfeld genutzt werden, lassen Sie **Primärschlüssel erzeugen** abgewählt. Das Festlegen als Primärschlüssel erfolgt in diesem Fall auf der dritten Dialogseite des Assistenten.

Beim Import soll die Definition der Tabelle erfolgen und die Daten übertragen werden.

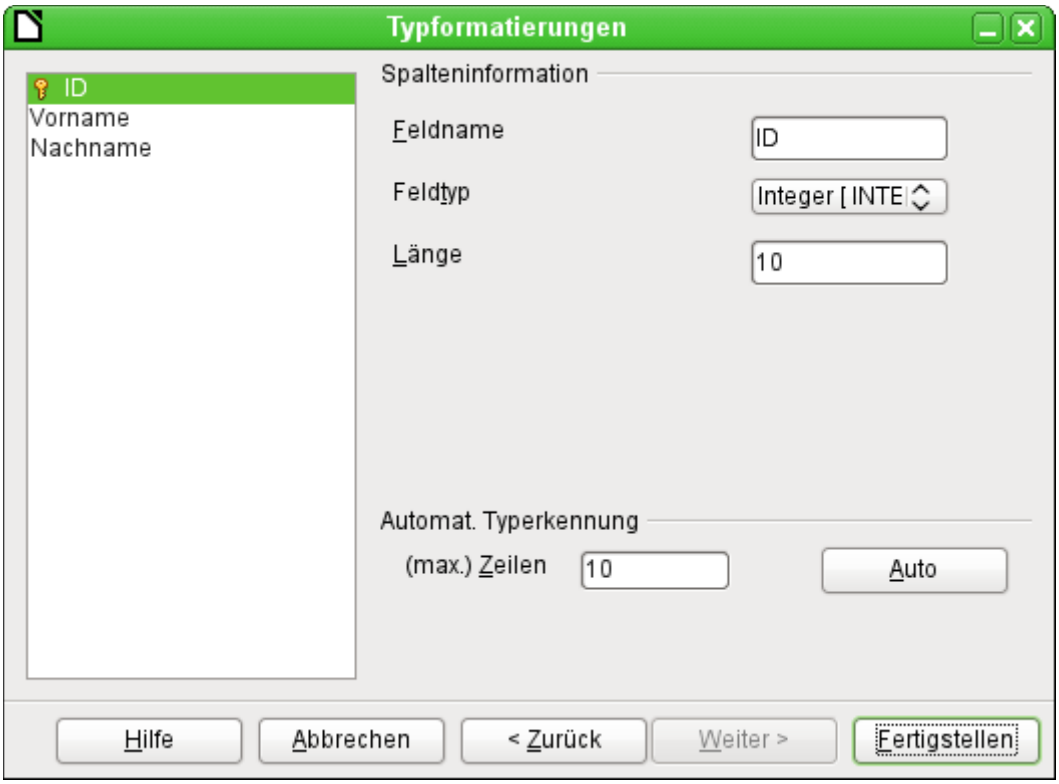


Alle vorhandenen Spalten werden übernommen.



Bei der Formatierung des Tabellentyps muss meist nachgebessert werden. In der Regel werden die Felder als Textfelder mit sehr hoher Länge vordefiniert. Zahlen- und Datumsfelder sollten also auf jeden Fall unter **Typformatierungen → Spalteninformation → Feldtyp** entsprechend eingestellt werden. Bei Dezimalzahlen mit Nachkommastellen ist außerdem auf die Anzahl der Nachkommastellen zu achten.

Die Auswahl des Primärschlüssels liegt etwas versteckt im Kontextmenü des Feldes, das den Primärschlüssel erhalten soll. Hier wird gerade das Feld "ID" formatiert, das als Primärschlüssel vorgesehen ist. Der Primärschlüssel muss hier noch einmal über das Kontextmenü des Feldnamens gesondert ausgewählt werden, wenn er nicht durch den Assistenten im Fenster **Tabelle kopieren** als zusätzliches Feld erstellt wurde.



Nach der Betätigung des Buttons Fertigstellen wird die Tabelle erstellt und mit dem kopierten Inhalt gefüllt.

Der neue Primärschlüssel ist kein «*Auto-Wert*»-Schlüssel. Um einen entsprechenden «*Auto-Wert*»-Schlüssel zu erzeugen, muss die Tabelle zum Bearbeiten geöffnet werden. Dort können dann weitere Formatierungen der Tabelle vorgenommen werden.

Hinweis	In FIREBIRD ist diese nachträgliche Bearbeitung nicht möglich. Hier hilft, **zuerst** nur die **Definition** zu erstellen, dann ein Feld für den Primärschlüssel als «Auto-Wert» an den Schluss der Tabellendefinition hinzuzufügen und **anschließend** mit einem Neustart des Assistenten **Daten anhängen** zu wählen. Die Position des Feldes kann bei Firebird später an den Beginn der Tabelle gesetzt werden. ALTER TABLE "Namen" ALTER "ID" POSITION 1; Dies setzt dann das Feld "ID", das als Primärschlüssel mit «Auto-Wert» in der GUI erstellt wurde, als erstes Feld der Tabelle.

Daten Aufsplitten beim Import

Manchmal liegen Daten in der Datenquelle nicht in der gewünschten normalisierten Form vor. Adressen, die z. B. in einer Tabellenkalkulation enthalten sind, enthalten häufig den kompletten Eintrag von Postleitzahl und Ort. Beim Import ist dann vielleicht gewünscht, diese Informationen in einer gesonderten Tabelle zu speichern und die Relationen zwischen den Tabellen zu gewährleisten.

Einen möglichen Weg, die Relation direkt zu erstellen, stellt die folgende Fassung dar:

1. Die komplette Tabelle mit allen Adressinformationen wird als Tabelle "Adresse" in Base importiert. Details dazu siehe in den vorherigen Kapiteln.
2. Die Felder PLZ und Ort werden daraus mit einer Abfrage ausgelesen, kopiert und als separate "PLZ_Ort"-Tabelle abgespeichert. Dabei wird für das Feld "ID" als Primärschlüssel der Autowert gewählt.
   Abfrage hierfür:
   `SELECT DISTINCT "PLZ", "Ort" FROM "Adresse"`
3. Der Tabelle "Adresse" wird ein Feld "PLZ_ID" hinzugefügt.
4. Über **Extras → SQL** wird ein Update für diese Tabelle ausgeführt:
   `UPDATE "Adresse" AS "a" SET "a"."PLZ_ID" = (SELECT "ID" FROM "PLZ_Ort" WHERE "PLZ"||"Ort" = "a"."PLZ"||"a"."Ort")`
5. Die Tabelle "Adresse" wird zum Bearbeiten geöffnet und die Felder "PLZ" und "Ort" gelöscht. Die Änderung wird gespeichert und anschließend die Tabelle wieder geschlossenen.

Damit sind die Tabellen so getrennt, dass eine 1:n-Beziehung zwischen der Tabelle "PLZ_Ort" und der Tabelle "Adresse" erstellt werden kann. Diese Beziehung wird anschließend über **Extras → Beziehungen** definiert.

Für Details zum SQL-Code siehe insbesondere «Abfrageerweiterungen im SQL-Modus» im Kapitel «Abfragen».

Mängel dieser Eingabemöglichkeiten

Eingaben mit einer Tabelle alleine berücksichtigen 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		

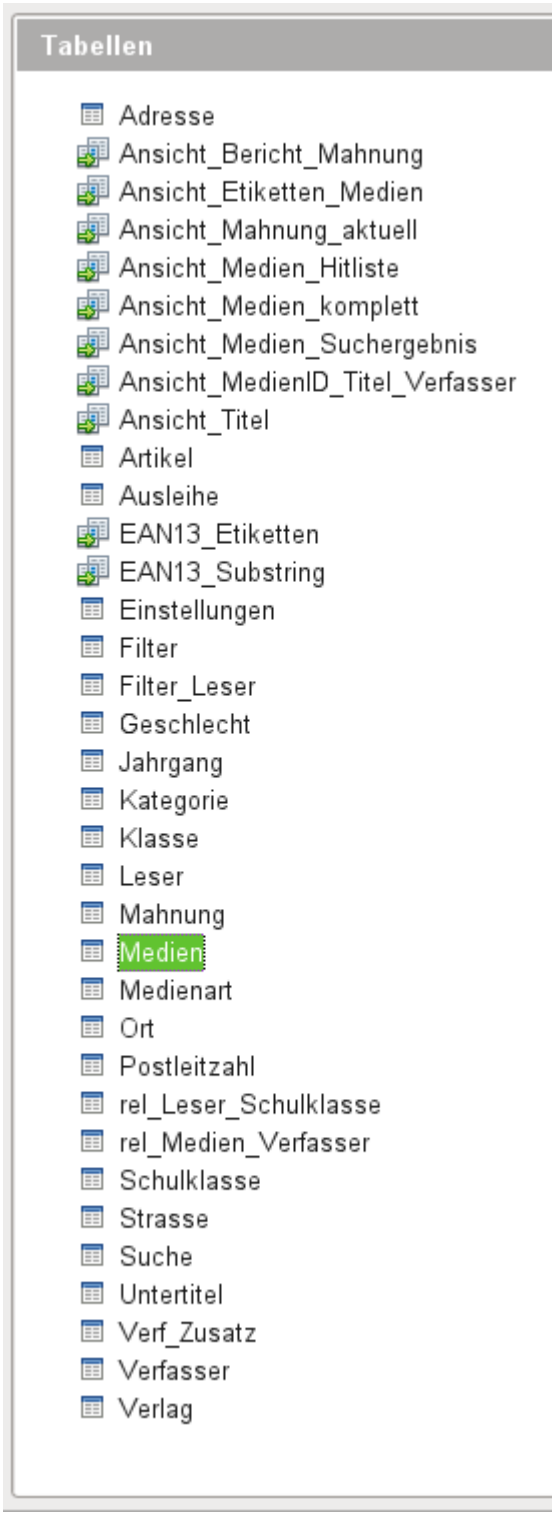
Die Ausleihtabelle besteht aus Fremdschlüsseln für das auszuleihende Medium "Medien_ID" und den entsprechenden Nutzer "Leser_ID" sowie einem Ausleihdatum "Leih_Datum". In die Tabelle werden also bei der Ausleihe zwei Zahlenwerte (Mediennummer und Benutzernummer) und ein Datum eingetragen. Der Primärschlüssel 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.

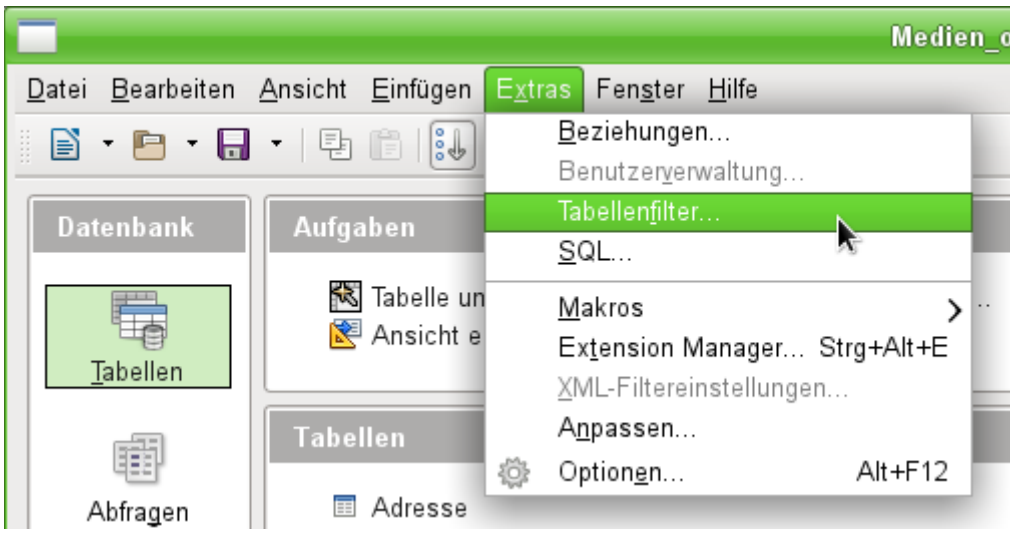
Die Eingabe in Tabellen ist in Datenbanken daher nur bei einfachen Tabellen sinnvoll. Sobald Tabellen in Relation zueinander gesetzt werden, bietet sich besser ein entsprechendes Formular an. In Formularen können diese Relationen durch Unterformulare oder Listenfelder gut bedienbar gemacht werden.

Tabellen verstecken

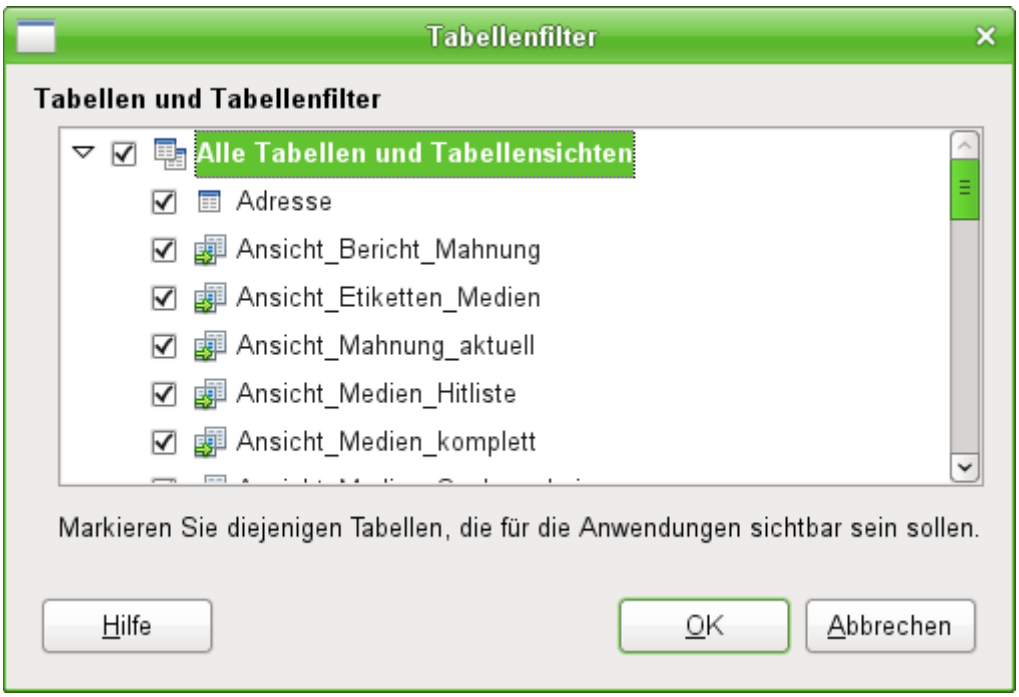
Bei einer entsprechend großen Datenbank erscheinen in der Tabellenübersicht sehr viele Tabellen und Tabellenansichten.



So eine Übersicht ist sicher beim Entwickeln der Datenbank sinnvoll. Mit Hilfe von Base ist es aber auch möglich, nur einen Teil dieser Tabellen überhaupt in der Ansicht aufzuführen. Sie erscheinen dann nicht in der Liste, sind aber sehr wohl für Eingaben und Abfragen über Formulare usw. verfügbar.



Über **Extras → Tabellenfilter** kann der Filter für Tabellen gestartet werden.



Würde hier der Haken bei **Alle Tabellen und Tabellenansichten** entfernt, so würde anschließend der Tabellencontainer komplett leer erscheinen. Die Tabellen ständen dann zur direkten Eingabe für den Normaluser nicht mehr zur Verfügung, solange er eben diesen Filter nicht entdeckt. Die Ausblendung schützt also nicht vor absichtlichem, vielleicht aber ein bisschen vor versehentlichem Missbrauch.

Ist eine entsprechende Auswahl getroffen worden, so sind erst einmal weiterhin alle Tabellen und Ansichten sichtbar. Hier muss über **Ansicht → Tabellen aktualisieren** die Ansicht neu eingelesen werden.

So könnte eine Filterung der Tabellen aussehen, die z.B. der Normaluser zu Gesicht bekommt. Alle Tabellen, die die Medien in der Datenbank betreffen, werden noch angezeigt. Alle weiteren Tabellen zur Ausleihe, zu den Lesern und auch alle Ansichten wurden einfach ausgeblendet.

Hinweis	Über den Tabellenfilter können auch Ansichten auf ganze Datenbanken ausgeschaltet werden. Bei manchen Treibern für MySQL/MariaDB oder PostgreSQL werden sämtliche Datenbanken angezeigt, auf die der angemeldete Nutzer zumindest lesend Zugriff hat. Für den Normalgebrauch irritieren diese Datenbanken aber nur.